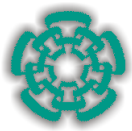

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Zacatenco

Departamento de Ingeniería Eléctrica
Sección de Computación

Modelos para Procesamiento Paralelo en Algoritmos Genéticos [1]

Tesis que presenta:
Marco Antonio Ortega García

Para obtener el grado de:
Maestro en Ciencias

En la especialidad de:
Ingeniería Eléctrica

Opción:
Computación

Director de tesis:
Dr. Arturo Díaz Pérez

Ciudad de México, DF. Octubre de 2003.

[1]Este trabajo fue parcialmente financiado mediante el proyecto CONACyT 31892A: Algoritmos y arquitecturas de computadoras con dispositivos reconfigurables.

Agradecimientos

Agradezco a mi asesor, el Dr. Arturo Díaz Pérez por el gran apoyo y conocimiento que aporto a éste trabajo de tesis, así como la paciencia que mostró durante todo el tiempo que llevo el desarrollo del proyecto.

Agradezco al Dr. Carlos Artemio Coello Coello y al Dr. José Oscar Olmedo Aguirre miembros del jurado, por el tiempo que dedicaron a la revisión de la tesis y por sus valiosos comentarios que hicieron de éste un mejor trabajo.

Agradezco a todos y cada uno de los profesores de la sección que con paciencia y dedicación aportaron su conocimiento haciendo de mis compañeros y de mí personas más competentes.

Agradezco a mis compañeros de la maestria por los consejos y conocimientos que me brindaron y por hacer más agradable el tiempo que permaneci en la sección de computación.

Agradezco a Sofia Reza (Sofy), la atensión y amabilidad que mostró siempre que necesite de su ayuda para efectuar algún tramite administrativo.

Al CINVESTAV, ésta gran institución de enorme calidad, que me brindó todo el apoyo durante mi estancia.

Reconocimientos

A mi madre y a mi padre:

A mi padre, por esa gran responsabilidad y fortaleza que siempre a mostrado por el trabajo y que me ha impulsado a seguir a delante.

A mi madre, por esa gran dedicación y atensión que ha tenido para mi y para mis hermanos. Por enseñarme con el ejemplo que nunca hay que darse por vencidos.

A ambos por esas llamadas de atensión necesarias, las cuales, me permitieron corregir mis errores he hicieron de mi una mejor persona. Por sus consejos y por seguirme enseñando cada día el valor de la familia y de saber que puedo contar con ellos y que ellos pueden contar conmigo.

A mis hermanos Norma, Jorge, Hector y Sandy por todo su apoyo y por comprender que aunque lejos de ellos, siempre pueden contar conmigo.

A mi abue Juanita y a mis demás familiares, por el gran apoyo que recibi al llegar a México.

A Carmen, por su confianza y animo que siempre me ha dado para concluir la maestria y por todo lo que ha representado en mi vida.

A Judith por el apoyo que me dió en las últimas etapas de mi tesis y a todas las personas que no he mencionado aquí, pero que me han ayudado directa o indirectamente a lograr dar éste gran paso.

Resumen

La complejidad de un problema es una medida que indica la dificultad inherente para poder resolverlo. Dos clases de complejidad que han sido ampliamente estudiadas son la clase P y la clase NP. A la clase de complejidad NP pertenecen los problemas intratables entre los que se encuentran los problemas de optimización combinatoria cuyo espacio de soluciones crece exponencialmente. Dado que no se conocen algoritmos polinomiales para resolver estos problemas de manera exacta y ya que es inviable hacer una búsqueda exhaustiva para instancias de tamaño moderado, se han propuestos diversas estrategias para obtener soluciones aproximadas. Los algoritmos genéticos, los cuales basan su estrategia en las teorías de evolución de Darwin, han mostrado ser un método eficaz para obtener soluciones buenas en algunas clases de problemas. Están basados en el uso de poblaciones de soluciones potenciales las cuales se combinan mediante operadores genéticos para obtener una nueva generación de soluciones. El método se itera hasta un número fijo de generaciones o hasta que se encuentra una solución razonablemente buena. Por lo anterior, los algoritmos genéticos requieren un alto consumo de recursos computacionales lo que representa una de sus mayores debilidades. Es posible, sin embargo, mejorar su desempeño usando técnicas de programación paralela las cuales reducen los tiempos de ejecución sin detrimento de la calidad de las soluciones obtenidas. En éste trabajo de tesis se presentan tres modelos de paralelización, *modelo global, modelo de islas y modelo celular*, para la implementación paralela de algoritmos genéticos. Para evaluar los beneficios del uso de paralelismo comparado con la versión secuencial del algoritmo se eligió un problema de optimización combinatoria muy conocido y fácil de entender llamado TSP o problema del agente viajero. Se presenta la implementación de un AG serial para el TSP. Posteriormente se presentan las versiones paralelas de dicho algoritmo genético serial usando los tres modelos de paralelización. Finalmente, se compara el rendimiento de cada uno de los modelos paralelos usando 4 y 8 procesadores contra la versión secuencial.

Abstract

The complexity of a problem is a measurement that indicates the inherent difficulty to solve it. Two classes of complexity that have been studied are the P and NP classes problems. Combinatorial optimization problems belong to the NP class whose space of solutions increases exponentially. Sice there are not known polinomial algorithms available to solve these problems in an exact way and since is nonviable to make an exhaustive search for instances of moderate size, diverse strategies have been proposed to obtaining approximated solutions. The genetic algorithms, which beses its strategy on Darwin's theory of evolution, have show to be an effective method to obtain good solutions in some types of problems. There are based on the use of potential populations of solutions which are combined by means of genetics operators to obtain a new group of solutions. The method is repeated until reaching a fixed number of generations or until one reasonably good solutions is found. Therefore, genetic algorithm requires a high consumption of computing resources which represent one of their weaknesses. It is possible, nevertheless, to improve his performance by using methods of parallel programming which reduce the execution or running time without harming the quality of the solutions obtained. In this tesis we present three parallel models, *global model, island model and cellular model*, each of which was adopted for the parallel implementation of genetic algorithms. In order to evaluate the benefits of using parallelism compared with the secuential version of the algorithm, we chose a of the solutions combinatorial optimization problem called TSP or travelling salesman problem. We start with the AG serial implementation of a genetic algorithm for the TSP. Later, we show the parallel we show the parallel version of this serial genetic algorithm using three parallel models. Finally, the performance of each one on the parallel models is compared using 4 and 8 processors with the sequential version.

Índice General

Índice de Tablas

Índice de Figuras

Introducción

Los algoritmos tienen características que los hacen adecuados o no a ciertos problemas, dichas características pueden ser velocidad, eficiencia, claridad, entre otras. Un parámetro que nos permite medir un algoritmo según su eficiencia y velocidad (para resolver un problema), es la complejidad. Dos clases de complejidad que han tomado un rol fundamental en ciencias computacionales, son las clases P y NP. La clase P consiste de los problemas conocidos como tratables o cuya complejidad se describe con un polinomio, a la clase NP pertenecen los problemas intratables, problemas que hasta ahora han resistido todos los esfuerzos de ser resueltos de manera exacta a través de algoritmos deterministas eficientes (ver apéndice A). Un área muy conocida y estudiada de los problemas NP está compuesta por los problemas de optimización combinatoria no lineales [3], para los cuales el número de soluciones posibles crece exponencialmente al aumentar el número de instancias involucradas; resultando inviable la búsqueda determinista de la solución óptima. Para este tipo de problemas se han diseñado una gran variedad de algoritmos no deterministas o heurísticas que trabajan en tiempo polinomial y que si bien, no garantizan encontrar soluciones exactas; sí permiten rastrear razonablemente el espacio de soluciones, aprovechando las particularidades de cada problema específico que se pretenda resolver. Ejemplo de lo anterior son los *algoritmos genéticos*, los cuales presentan excelentes características de flexibilidad, robustez y adaptabilidad para problemas donde el espacio de soluciones es extremadamente grande [30].

El contenido de este trabajo se centra en los *Algoritmos Genéticos* [2, 4] tanto secuenciales como paralelos (AGPs) cuyos métodos de búsqueda son no deterministas. Los algoritmos genéticos están basados en las teorías de evolución de Darwin [4] y su aplicación se ha expandido de los ámbitos académicos a las aplicaciones industriales y de negocios, entre otras. El algoritmo genético procesa una población de individuos encontrando a los más aptos por medio de la evaluación de cada uno de éstos. Cada individuo representa una solución potencial al problema bajo estudio, esto es, una solución del espacio de búsqueda. Los individuos más aptos son los que tienen más oportunidad de sobrevivir en la siguiente generación, de aquí la analogía con la teoría de selección natural. Es necesario agregar un operador de cruce que es la analogía a la reproducción sexual natural. Otro operador que se copia de la naturaleza es el operador de mutación, el cual es aplicado en un porcentaje menor que la cruza; pero con esto los individuos de la población experimentan cambios bruscos en su información genética; con ello se cubren áreas aún no exploradas del espacio de solución.

Los algoritmos genéticos han mostrado ser efectivos en obtener soluciones aceptables en problemas de optimización continua, lineal, no lineal y multiobjetivo. Sin embargo, su

efectividad para resolver problemas de optimización combinatoria ha sido cuestionada comparándolos con otras técnicas como son búsqueda local y recocido simulado.

A fin de lograr tener buenas soluciones, los algoritmos genéticos deben trabajar sobre poblaciones moderadamente grandes e iterarlas sobre un número de generaciones aceptable, de tal manera que al final han evaluado la aptitud sobre un número importante de soluciones del espacio de búsqueda. Una alternativa para mejorar el rendimiento de los algoritmos genéticos es su implementación mediante cómputo paralelo el cual es particularmente atractivo debido a la naturaleza inherentemente paralela para procesar la población de individuos de una generación. Esta es la estrategia directa de paralelización de un algoritmo genético. Sin embargo, la posibilidad de disponer de más de un procesador permite modificar ligeramente los modelos de manipulación de una sola población global. Particularmente es posible tener varias poblaciones que actúen de manera más o menos independiente y que en intervalos regulares de generaciones cada subpoblación pueda intercambiar información con otras subpoblaciones con el propósito de fomentar diversidad y movilidad de información genética que ayude a lozalizar una buena solución al problema planteado; esta posibilidad da origen a un nuevo operador conocido como migración. La aplicación de paralelismo debe mejorar los tiempos de ejecución de los algoritmos genéticos sin detrimento de la convergencia observada en las versiones secuenciales.
Este trabajo de tesis tiene como objetivo mostrar diversas alternativas que se obtienen al aplicar paralelismo para algoritmos genéticos sin menoscabo de la velocidad de convergencia. Por medio de un problema típico de optimización combinatoria, TSP, se presentan diversos modelos de paralelización de los algoritmos genéticos y se hace un estudio acerca de la convergencia de las versiones paralelas relacionada con el rendimiento observado en los tiempos de ejecución. Por la naturaleza estocástica del comportamiento de los algoritmos genéticos, se presentan tendencias generales en el desempeño de las variaciones paralelas las cuales son de utilidad para guiar en la elección del modelo de paralelización más adecuado para una aplicación y una arquitectura paralela. Por tanto, el objetivo del presente trabajo es mostrar que el rendimiento de los algoritmos genéticos puede mejorarse a través de técnicas paralelas y atacando al menos uno de los tantos problemas de optimización combinatoria existentes en esta área. Seleccionado por ser un problema clásico en optimización combinatoria, en el TSP se intenta minimizar la longitud total del recorrido que pasa por cada uno de los vértices de una gráfica sin tocar más de una vez cada vértice. El espacio de soluciones del TSP crece de forma factorial al aumentar el número de vértices involucrados; es decir, para n vértices el número de recorridos posibles es $(n-1)!$. Este problema es NP y son necesarias técnicas no convencionales para buscar su mejor solución.

En el presente trabajo se implementó un AG para el TSP, el algoritmo se compone de siete elementos básicos y debido a que se optó por una codificación donde cada una de las cadenas representa una solución al TSP fue necesario usar operadores diseñados para permutaciones y evitar con ello la generación de cadenas no válidas. Después de la implementación serial, se realizaron versiones en paralelo usando tres modelos que son: *el modelo global, modelo de islas y el modelo celular*. Para la implementación del esquema global primero se hicieron pruebas de rendimiento al algoritmo genético convencional con el fin de detectar las partes de código que requerían mayores recursos de procesador e implementar éstas en paralelo. Para el mode-

lo de islas se usaron cuatro esquemas que son: *topología de red, topología de estrella, topología de anillo y topología de hipercubo*. Para el esquema celular primero se realizó una simulación de este modelo en una computadora con un solo procesador y después se implementó una versión paralela de dicha simulación. Finalmente, se hizo el análisis y evaluación de resultados representados en gráficas y tablas, que nos muestran las ventajas de usar esquemas paralelos para los AG implementados en este trabajo.

Este trabajo de tesis está organizado en 5 capítulos de la manera siguiente: capítulo 1, se hace una descripción breve de los conceptos básicos de los AG y algunos de sus elementos principales así como los problemas que deben ser abordados para su implementación. Este capítulo se ha incluido con el propósito de que el reporte sea autocontenido, sin embargo, un lector experimentado sobre AG puede obviar su lectura.

En el capítulo 2 se mencionan algunos de los conceptos básicos de la programación paralela en general y de los AG paralelos.

En el capítulo 3 se explica en qué consiste el problema del agente viajero y se muestran algunos de los resultados obtenidos al intentar resolver el TSP con un algoritmo de búsqueda exaustiva. También se explican cada uno de los elementos del AG implementado para el problema del agente viajero y se muestra su pseudocódigo. El capítulo finaliza con un breve análisis de las limitaciones del AG implementado y se proponen algunas alternativas para mejorar su desempeño.

El capítulo 4 describe la manera en la que fueron implementados los tres esquemas de paralelización diseñados para los algoritmos genéticos descritos en el capítulo tres.

En el capítulo 5 se muestran las pruebas y resultados obtenidos de los códigos genéticos implementados y que confirmaron algunas de las hipótesis hechas en los primeros capítulos de este reporte.

La tesis finaliza con las conclusiones obtenidas de los resultados mostrados en el capítulo cinco.

Capítulo 1

Computación Evolutiva y los Algoritmos Genéticos

La evolución de las especies en la vida natural se produce como resultado de dos procesos primarios: la selección natural y la reproducción sexual. La primera determina qué miembros de la población sobrevivirán hasta reproducirse, la segunda garantiza la mezcla y combinación de sus genes entre la descendencia. Los individuos compiten entre sí por recursos tales como comida, agua, refugio y adicionalmente los animales de la misma especie normalmente compiten para obtener una pareja con la cual podrán crear descendencia. Esta es la teoría de evolución de las especies de Darwin, en ella los individuos luchan para adaptarse al medio que las rodea y aquellos individuos que tengan más éxito en tal adaptación tendrán mayor probabilidad de sobrevivir hasta la edad adulta y probablemente generar un gran número de descendientes, teniendo mayores probabilidades de que sus genes sean propagados a lo largo de sucesivas generaciones. La combinación de características de los padres bien adaptados en un descendiente puede producir muchas veces un nuevo individuo mucho mejor adaptado a las características de su ambiente que cualquiera de sus padres.

La idea de crear algoritmos con características similares a las de la teoría de la selección natural surgió en Michigan, Estados Unidos donde el profesor John Holland quien, consciente de la importancia de la selección natural, introdujo la idea de los planes reproductivos con el objetivo de que las computadoras aprendieran por sí mismas; esta técnica se hizo popular con el nombre de Algoritmos Genéticos. En términos generales para simular un proceso evolutivo en una computadora se requiere:

- Codificar las estructuras que se replicarán.
- Operaciones que afecten a los individuos.
- Una función de aptitud.
- Un mecanismo de selección.

1.1 Paradigmas en Computación Evolutiva (CE)

La computación evolutiva es un término relativamente nuevo, que intenta agrupar diferentes paradigmas los cuales estan relacionados entre sí por las teorías de selección natural. Computación Evolutiva se refiere al estudio de los fundamentos y aplicaciones de ciertas técnicas heurísticas de búsqueda basadas en los principios naturales de la evolución; teniendo una gran variedad de algoritmos evolutivos que han sido propuestos y que se clasifican de la siguiente manera: *Algoritmos Genéticos, Programación Evolutiva, Estrategias Evolutivas* los cuales se describirán de manera breve a continuación.

- **Algoritmos Genéticos.** Estrategias basadas en la teoria de evolución de Darwin y donde son implementados algoritmos de *selección, cruza y mutación* sobre cadenas de símbolos llamadas individuos. El operador principal en esta técnica es la cruza.
- **Programación Evolutiva.** Se aplica satisfactoriamente para problemas de optimización automática y aprendizaje de máquinas. Desarrolla programas usando analogías de los procesos biológicos tales como la evolución y la selección natural. A diferencia de los AGs esta técnica slo implementa el operador de cruza.
- **Estrategias Evolutivas.** Son algoritmos evolutivos enfocados hacia la optimización paramétrica que utilizan una representación a través de vectores reales, selección determinista y operadores específicos de mutación y cruce, el cual ocupa ahora un lugar secundario con respecto a la mutación.

Cada uno de los paradigmas anteriores se originó de forma independiente y algunas de sus aplicaciones son las siguientes: los algoritmos genéticos suelen utilizarse para resolver problemas de optimización en espacios discretos, las estrategias evolutivas para resolver problemas de ingeniería y la programación evolutiva se usa como un método eficiente para resolver cierto tipo de problemas que puedan ser representados con naturalidad a través de máquinas de estados finitos.

1.2 Definicin de Algoritmos Genéticos

Los Algoritmos Genéticos se inspiran en el proceso observado en la evolución normal de los seres vivos. Este proceso de evolución natural ha mostrado gran consistencia al tratar de resolver problemas cuyo espacio de solución es muy grande. Tal es el caso de los problemas de optimización combinatoria que constituyen parte de una amplia variedad de problemas difíciles de resolver y a los cuales se les ha denominado problemas no polinomiales o NP (ver apéndice A). Algunos ejemplos de este tipo de problemas son: optimización de funciones matemáticas, optimización combinatoria, problemas de optimización de rutas, optimización de circuitos, etc.

De acuerdo a John Koza [15], "El AG es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de la reproducción y supervivencia del más

Naturaleza	Algoritmos Genéticos
Gen	Parte elemental de una cadena binaria.
Alelo	Valor que puede tomar un gen.
Individuo	Codificación a la solución al problema.
Cromosoma	Cadena o palabra binaria.
Población	Conjunto de estructuras (individuos).
Hijos	Nuevas estructuras.
Fenotipo	Función de adaptación.
Genotipo	Características heredadas (genes).
Adaptación	Aptitud o fitness.
Medio ambiente	Función de evaluación.
Generación	Ciclo.
Herencia	Trasmisión de características.
Selección	Guía en la búsqueda de la solución.
Reproducción	Operador de cruza.
Mutación	Cambio brusco en el genotipo.
Deme	Subpoblación de individuos

Tabla 1.1: En esta tabla se muestran las analogías de los AGs con la evolución natural

apto, y tras haberse presentado de forma natural una serie de operaciones genéticas entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se asocian con una cierta función matemática que refleja su aptitud. Estos principios son imitados en la construcción de algoritmos computacionales que buscan la mejor solución para un determinado problema a través de evolución de poblaciones". Algunos de los términos que se manejan en AGs y que tienen su analogía con el sistema natural se muestran en la Tabla 1.1.

1.3 Estructura básica de los Algoritmos Genéticos

En los Algoritmos Genéticos (GAs) no es necesario disponer de un conocimiento profundo del problema a resolver para que éste pueda ser descompuesto en subproblemas; se parte de estructuras simples que interactúan entre sí, dejando que sea la evolución quien haga el trabajo. Así, los AGs utilizan una analogía directa del fenómeno de evolución de la naturaleza, trabajan con una población de individuos, cada uno representando una posible solución al problema dado. A cada individuo se le asigna una puntuación de adaptación, dependiendo de qué tan buena sea la respuesta al problema, a los mejores se les da mayor probabilidad de reproducirse mediante cruzamiento con otros individuos de la población produciendo descendientes con características de ambos padres; sin embargo, nada garantiza que dichos individuos producirán descendencia. Los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción, y pueden desaparecer. En ocasiones se producen mutaciones en los individuos, lo que origina cambios drásticos en las características

del individuo, evitando así estancamientos [31].

Se necesitan cinco componentes básicos para implementar un AG que resuelva un problema específico:

1. Encontrar una representación de soluciones potenciales al problema.
2. Crear un mecanismo que genere una población inicial de soluciones potenciales, esto se hace generalmente de forma aleatoria aunque es posible usar métodos determinísticos.
3. Determinar cuál será la función de evaluación (la cual representará el medio ambiente), calificando a las soluciones producidas en términos de su aptitud.
4. Elegir a los operadores que serán usados para alterar la composición de los individuos. Por lo general son usados operadores de cruza y mutación. Además, es necesario contar con un mecanismo de selección de individuos, que hará una elección de las cadenas (usando algún criterio) que serán modificadas por los operadores genéticos.
5. Finalmente, deben determinarse los valores que serán asignados a los parámetros usados por el AG.

En la Figura 1.1 se presenta el diagrama de un AG básico, el cual primero parte de una población (población inicial) de individuos y sobre ellos, una y otra vez se aplican una serie de operaciones como evaluar a la población, seleccionar los mejores individuos, aplicar operadores de reproducción y mutación y aplicar el criterio de reemplazamiento. Este proceso se repetirá hasta que se cumpla la condición que se haya establecido para la finalización del ciclo genético *Criterio de fin* o se llegue a un número máximo de repeticiones.

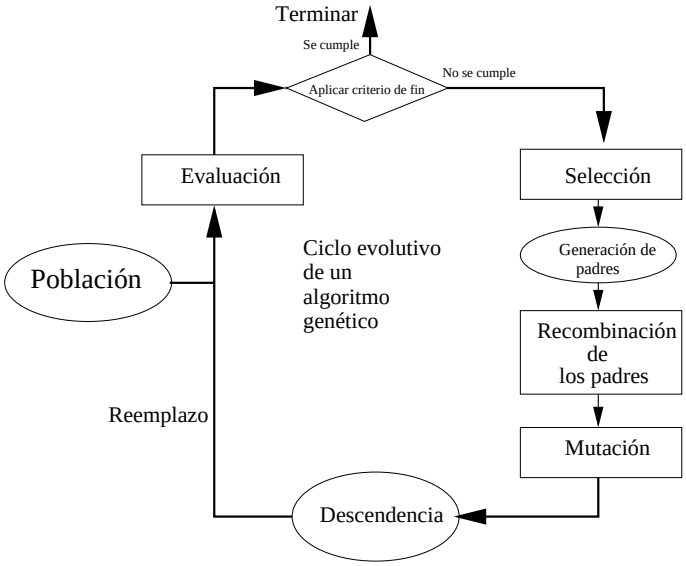

Figura 1.1: Diagrama de un AG convencional.

1.3.1 Representación de las cadenas genéticas

En los AGs un individuo es un cromosoma y es el código de información sobre el cual opera el algoritmo evolutivo. Ese código de información es una representación de las posibles soluciones dentro del espacio de búsqueda de un problema y define la estructura del cromosoma que va a ser manipulado por el código evolutivo.

En AGs la representación tradicional usada para codificar un conjunto de soluciones es el esquema binario en el cual un cromosoma o cadena de genes es una cadena de la forma $(p_1, p_2, p_3, ...p_k)$ donde $p_1, p_2, p_3, ...p_k$ se denominan *alelos* que pueden ser ceros o unos, ver Figura 1.2. Existen razones por las cuales suele usarse la codificación binaria en los AGs, aunque la mayoría de ellas se remontan al trabajo de Holland en el área quien dio en su libro una justificación teórica para usar codificaciones binarias [5]. Holland comparó dos codificaciones con una capacidad de informacin aproximadamente igual, (cadena de bits de longitud 100 y otra cadena decimal de longitud 30). Holland [5] argument que el primer tipo de codificacin permite mayor grado de paralelizacin implícita debido a que permite más esquemas[1] que la segunda (11_{30} contra 3_{100})[2].

0	1	1	1	0	1

Figura 1.2: Ejemplo de una cadena con representación binaria.

Sin embargo el uso de la representación binaria no siempre es lo más adecuado, debido a que diversos problemas se prestan de manera natural para la utilización de representaciones de mayor cardinalidad que la binaria. Por ejemplo, el problema del viajero se presta de forma natural para el uso de permutaciones de enteros decimales. La Figura 1.3 muestra el ejemplo de una representación entera decimal para el problema del Agente Viajero con 6 ciudades.

2	1	5	3	4	6

Figura 1.3: Ejemplo de una cadena con representación de enteros decimales para el TSP.

1.3.2 Población

Los AGs generan un conjunto de cadenas de información o hipótesis conocido comúnmente como *población*, la cual evoluciona paralelamente a través de k generaciones o ciclos adaptándose al medio que le rodea. Cada una de las cadenas de dicha población está asociada con una

[1] Un esquema es una plantilla que describe un subconjunto de cadenas que comparten ciertas similitudes en algunas posiciones a lo largo de su logitud [5].

[2] El número de esquemas de una cadena se calcula usando $(c+1)_l$, donde c es la cardinalidad del alfabeto y l es la longitud de la cadena.

aptitud o valor de adaptación al medio, el cual es usado por cada cadena para competir con las demás. Con ello se ataca un conjunto de puntos representativos de diferentes zonas del espacio de búsqueda y no un solo punto. Una pregunta que se plantea está relacionada por lo general con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional. No existe alguna regla que nos diga cual sería el tamaño de población idóneo para algún problema específico. Por tanto, es importante encontrar el equilibrio entre estos dos factores permitiendo a la población del AG evolucionar de forma adecuada sin caer en la excesiva generalidad o en la escasa diversidad.

La iniciación de la población determina el proceso de creación de los individuos para el primer ciclo del algoritmo. Normalmente, la población inicial se forma a partir de individuos creados aleatoriamente. También es posible que a la población inicial (creada aleatoriamente) le sean sembrados cromosomas buenos.

1.3.3 Evaluación

La evaluación es representada por la función de aptitud o función objetivo que es un modelo computacional adecuado al problema tratado y es el elemento crítico de todo programa evolutivo. El objetivo principal de la función de aptitud es calcular un valor o costo de aptitud para cada uno de los individuos contenidos en la población. El método más simple de atribuir un valor de aptitud a un individuo es utilizar el valor numérico del resultado de la evaluación de ese cromosoma por la función de evaluación. Sin embargo, al intentar calcular el valor de evaluación es necesario tener claro que en muchos problemas de optimización combinatoria donde existen diversas restricciones, los puntos obtenidos por los operadores del AG pueden llegar a formar individuos no válidos requiriendo por ello métodos que corrijan este problema, algunas técnicas usadas son las siguientes:

1. Enfoque basado en la penalización de la función de aptitud. La idea general consiste en dividir la aptitud del individuo por una cantidad (la penalización) que guarda una relación con las restricciones que dicho individuo viola. Con ello se espera disminuir la probabilidad de que dicho individuo sea seleccionado para su reproducción.

2. Otra posibilidad consiste en reconstruir o reparar aquellos individuos que no verifiquen las restricciones. Dicha reconstrucción suele llevarse a cabo por medio de un nuevo operador que se acostumbra denominar *reparador*.

3. Se diseñan operadores genéticos específicos para el problema tratado Por ejemplo para el caso del TSP son necesarios operadores genéticos especiales para permutaciones, evitando con ello la generación de individuos no válidos.

1.3.4 Operadores genéticos

Los operadores genéticos son los diferentes métodos u operaciones que se pueden ejercer sobre los individuos de una población y que nos permiten obtener poblaciones nuevas. Después

de haber evaluado cada individuo con una función de aptitud se aplican los operadores genéticos, la elección de éstos afectará de forma significativa el desempeño del algoritmo. La elección del tipo de operadores a usar dependerá del problema que se esté atacando. En algoritmos genéticos se destacan los siguientes:

- Operador de selección.
- Operador de recombinación.
- Operador de mutación genética.
- Operador de reemplazamiento de población.

A continuación se describen brevemente cada uno de estos operadores.

Técnicas de selección

El paso siguiente a la evaluación es escoger los miembros de la población que serán utilizados para la reproducción. A estos individuos los conoceremos como padres. Veremos a la selección de padres como una tarea de distribución de oportunidades de reproducción para cada individuo y se espera que al igual que los individuos mejor adaptados, los individuos con baja adaptación tengan una probabilidad mayor a cero de ser seleccionados. La selección de los individuos suele realizarse de forma probabilística. Las técnicas de selección se clasifican generalmente en tres grupos:

- **Selección proporcional al ajuste dado por la función de evaluación**. En este método se eligen individuos de acuerdo a su contribución de aptitud con respecto al total de la población. Algunas de las técnicas que se usan son:

  1. Método de la Ruleta [28].
  2. Sobrante estocástico [16, 17].
  3. Jerarquías [18].

  El método de la ruleta tiene una complejidad de $O(n^2)$ y al igual que las otras dos técnicas tiene el problema de la convergencia prematura.

- **Selección mediante torneo**. Esta técnica fue inventada por Wetzel [10] y su idea básica es seleccionar con base en comparaciones directas de los individuos. La idea de este método es muy simple. Se permuta la población y se hacen competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente compiten en parejas) en un torneo del que resultarán ganadores aquellos que tengan valores de aptitud más altos. Si se efectúa un torneo binario (o sea, competencia por parejas), entonces la población se debe de permutar dos veces. Nótese que esta técnica garantiza la obtención de múltiples copias del mejor individuo entre los progenitores de la siguiente generación (si se efectúa un torneo binario, el mejor individuo será seleccionado dos veces), ver figura 1.4.

El método agrupa individuos en subconjuntos creados al azar y selecciona a uno de ellos; para la elección del individuo existen dos estrategias:

1. La primera es seleccionar al mejor de cada subconjunto, llamado *método de los torneos determinísticos.*
2. La segunda es elegir al mejor o al peor individuo del subconjunto a través de una probabilidad p, llamado *método de los torneos probabilísticos.*

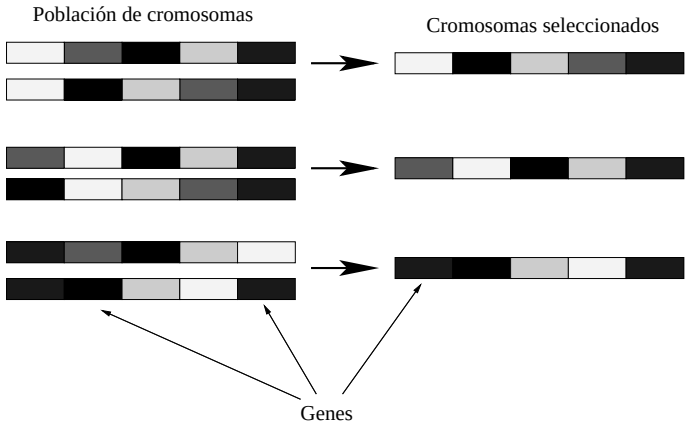

Figura 1.4: Selección de individuos mediante torneo binario.

Esta técnica es eficiente y fácil de implementar, su complejidad es *O(n).* Para una revisión más detallada de este método ver [1, 10].

- **Selección de estado uniforme**. Técnica propuesta por Whitley usada en AG no generacionales en los cuales algunos individuos son reemplazados en cada ciclo generacional. Un uso común de esta técnica es cuando se evolucionan sistemas basados en reglas en los que el aprendizaje es incremental o cuando los miembros de la población resuelven colectivamente un problema y no de forma individual. Para más detalles ver [1].

Reproducción y mutación

Una vez realizada la selección, se procede a la reproducción sexual o cruce de los individuos seleccionados. En esta etapa los supervivientes intercambiarán material cromosómico y sus descendientes formarán la población de la siguiente generación.

Operador de cruza

Consiste en unir de alguna forma los cromosomas de dos cadenas de información que han sido previamente seleccionadas de la generación anterior para formar nuevos descendientes. Existen diferentes técnicas y variaciones que son empleadas para hacer que dos individuos combinen su material genético y formen con él nuevos descendientes. Algunas de las más empleadas son:

- **Cruza con un punto de corte**. Se genera un punto aleatorio dentro de las cadenas genéticas que serán combinadas (llamado punto de cruce). El punto dividirá las cadenas en dos subcadenas las cuales serán combinadas entre sí para formar una o más cadenas genéticas nuevas.

- **Cruza con dos puntos de corte**. El operador de cruzamiento de dos puntos consiste en generar dos puntos aleatorios dentro de las cadenas genéticas que se combinarán. Dichos puntos dividirán cada cadena genética en tres subcadenas, que serán combinadas entre sí para formar una o más cadenas genéticas.

- **Cruza uniforme**. Se atribuye generalmente a Syswerda [13] y consiste de una cruza de n puntos, aunque no se fija previamente el número de puntos de cruza. En este operador cada gen, en la descendencia se crea copiando el correspondiente gen de uno de los dos padres, escogido de acuerdo a una máscara de cruce (por lo general aleatoriamente). Cuando existe un 1 en la máscara de cruce, el gen es copiado del primer padre; mientras si existe un 0 en la máscara de cruce, el gen se copia del segundo padre.

Estas técnicas aunque son de uso común para cadenas binarias, muchas veces no son útiles en alfabetos de mayor cardinalidad o para determinado tipo de representación como por ejemplo algunos tipos de representación como es el caso del problema del agente viajero, donde las cadenas genéticas están formadas por las ciudades en el orden en que se recorren para formar un tour. Estas cadenas formarán cadenas de enteros para representar permutaciones (ver capítulo 4 para más detalles) debido a que una ciudad no puede aparecer más de una vez en un recorrido o tour, como en la siguiente cadena:

1 2 3 4 5 6

que representa un tour de 6 ciudades, y donde el ltimo recorrido va de la ciudad 6 a la ciudad 1.
Al efectuar cualquiera de los operadores de cruza entre dos cadenas que usan una permutación como representación se formarán inevitablemente cadenas no válidas. Para poder hacer uso de los operadores de recombinación es necesario hacer algunas modificaciones para evitar la generación de cadenas no válidas. Ejemplo de ello es el diseño de operadores de cruza para permutaciones, que son usadas frecuentemente en problemas de optimización combinatoria. Algunas de esas técnicas son:

- **Position-based Crossover.** Esta técnica fue propuesta por Syswerda [13] como una adaptación a la cruza uniforme para permutaciones. Los pasos del algoritmo son los siguientes:

  1. Seleccionar al azar un conjunto de posiciones en P1.
  2. Producir un hijo borrando de P1 todos los valores, excepto aquéllos que hayan sido seleccionados en el paso anterior.
  3. Borrar los valores seleccionados de P2. La secuencia resultante de valores se usará para completar el hijo.

4. Colocar en el hijo los valores faltantes de izquierda a derecha, de acuerdo a la secuencia de P2.
5. Repetir los pasos del 1 al 4, pero ahora tomando la secuencia de P2.

Un ejemplo de esta técnica se puede ver en [1].

- **Order-based Crossover.** Propuesta por Syswerda [13] como una variante de Position-based Crossover en la que cambia el orden de los pasos del algoritmo.
  En este caso, primero se selecciona una serie de valores de P1. Luego, removemos de P2 esos valores. A continuación se genera a un hijo a partir de P2'. Finalmente, completamos los hijos con los valores de la secuencia obtenida de P1 (insertada de izquierda a derecha en el orden impuesto por P1).

  Un ejemplo de esta técnica se puede ver en el capítulo 3.

Operador de mutación

Uno de los objetivos de este operador es dar diversidad a la población. Un problema muy común en AG's es que al generar aleatoriamente la población inicial o después de varias generaciones se pueden generar en posiciones similares de los cromosomas, esquemas también iguales. Esto supone un estancamiento debido a que el operador de reproducción (cruza) no cambia ese tipo de elementos, aun después de muchos ciclos. El aplicar operadores de mutación con una correcta probabilidad pueden ayudar a salir de ese tipo de estancamientos locales y permitir la correcta construcción de bloques. El operador de mutación origina variaciones elementales en la población y garantiza que cualquier punto del espacio de búsqueda pueda ser alcanzado.
Una forma de aplicar el operador de mutación si usamos una representación binaria es, sustituyendo un bit por su complemento (un cero cambia en uno y viceversa) permitiendo con ello la introducción de nuevo material cromosómico en la población como sucede con sus equivalentes biológicos. Al igual que el cruce, la mutación se maneja como un porcentaje que indica con qué frecuencia se efectuará, aunque se distingue de la primera en que ocurre mucho más esporádicamente.

Mutación para Permutaciones

Al igual que para el operador de cruza, existe una representación para permutaciones (para operadores de mutación) que es usado en problemas de optimización combinatoria (como el del viajero). Algunos de los métodos más usados son los siguientes:

- **Mutación por Inserción**. Se selecciona un valor en forma aleatoria y se le inserta en una posición arbitraria. Un ejemplo de esta técnica se puede ver en [1].

- **Mutación por Desplazamiento**. Es una generalización de la mutación por inserción en la que en vez de mover un solo valor, se cambian de lugar varios a la vez. Un ejemplo de este método se da en el capítulo 3 al aplicarlo al problema que se trata de resolver.

Técnicas de reemplazamiento y criterios de paro

Después de aplicar los operadores de selección, cruza y mutación a las cadenas de la población de padres y obtener con ello la nueva población de individuos (hijos), es necesario determinar la forma en la que será reemplazada esta población de padres por la nueva población de descendientes. Esto ha dado lugar a una clasificación de los AG's basada el tipo de reemplazamiento usado. Estos dos tipos de genéticos son:*Algoritmos Genéticos Generacionales y Algoritmos Genéticos no Generacionales o de Estado uniforme.*

Algoritmos Genéticos Generacionales. Se asemejan a la forma de reproducción de los insectos, donde una generación pone huevos, se aleja geográficamente o muere y es sustituida por una nueva. Este modelo es conocido como Algoritmo Genético Canónico.

Algoritmo Genético no Generacoinal. Utilizan el esquema generacional de los mamíferos y otros animales de vida larga, donde coexisten padres y sus descendientes, permitiendo que los hijos sean educados por sus progenitores, pero también a la larga se genere competencia entre ellos.
En este modelo, no sólo se deben seleccionar los dos individuos a ser padres, sino también cuáles de la población anterior serán eliminados, para dar espacio a los descendientes. La diferencia esencial entre el reemplazo generacional y el modelo de estado uniforme es que en éste último las estadísticas de cada descendiente en la población se recalculan al finalizar el cruce de los padres, por tanto los nuevos individuos estarán disponibles inmediantamente para la reproducción. Esto permite al modelo utilizar las características de un individuo prometedor tan pronto como es creado.

Finalmente, los pasos descritos antes se repiten evaluando en cada ciclo a la nueva generación obtenida terminando las evaluaciones de acuerdo al criterio de paro. Existen varios criterios, algunos de ellos son los siguientes:

- Repetir los AG's un número determinado de generaciones o ciclos.
- Repetir el proceso hasta que la población total converja a un cierto valor.
- Repetir hasta que algún individuo este muy cerca de un valor específico.

1.4 Algoritmos Relacionados

Existen una gran cantidad de técnicas que permiten resolver diversos problemas. Por ejemplo, en los problemas donde es preciso optimizar una función lineal los métodos Simplex [7] son una opción viable. Para ploblemas con funciones no lineales hay métodos directos, como *la búsqueda aleatoria* [7] y métodos no directos como el *método del gradiente conjugado* [7]. Las técnicas clásicas de optimización requieren de información que en muchas ocasiones es muy difícil de obtener. Por ejemplo, el método de gradiente conjugado requiere de la primera derivada de la función objetivo, lo que en ocasiones no es posible debido a que la función no es derivable o peor aún, no se tiene una función objetivo para el problema.

Otras técnicas conocidas como heurísticas son usadas al enfrentarse con problemas cuyo espacio de soluciones es muy grande, por ejemplo, como el problema del agente viajero, donde los algoritmos determinísticos más eficientes que existen para resolver el problema requieren tiempo exponencial. Algunos ejemplos de estas técnicas heurísticas se describen a continuación.

Búsqueda Aleatoria

La aproximación por fuerza bruta para funciones difíciles es una búsqueda aleatoria o enumerativa. Los puntos en un espacio de búsqueda y la evaluación de su aptitud son seleccionados aleatoriamente o de manera sistemática. Esta no es una estrategia muy inteligente por lo que es muy poco usada.

Búsqueda tabú

La búsqueda tabú es una meta-heurística, porque es un procedimiento que debe acoplarse a otra técnica, ya que no funciona por sí sola. La búsqueda tabú usa una memoria para guiar la búsqueda, de tal forma que algunas soluciones examinadas recientemente son memorizadas y se vuelven tabú (prohibidas) al tomar decisiones acerca del siguiente punto de búsqueda. La búsqueda tabú es determinística, aunque se le pueden agregar elementos probabilísticos [9].

Recocido Simulado

Esta técnica fue propuesta por Kirkpatrick en 1982. Una buena descripción de esta técnica se encuentra en [34, 8]. Es esencialmente una modificación de la versión del Hill Climbing. Inicia de un punto aleatorio en el espacio de búsqueda y se hace a su vez un movimiento también aleatorio. Si este movimiento es hacia un punto alto (que mejora la solución actual), ese punto es aceptado; si el movimiento es hacia un punto muy bajo (que empeora la solución actual), dicho punto se acepta con una probabilidad de $p(t)$, donde t es el tiempo. La función $p(t)$ inicia con valor uno, pero gradualmente se reduce hacia cero, análogamente al enfriamiento de un sólido. Inicialmente la probabilidad de aceptar puntos con valores diversos es alta. Sin embargo, al reducirse la temperatura, la probabilidad de aceptar movimientos negativos se reduce. Como la búsqueda aleatoria, sin embargo, el recocido simulado opera con una solución a la vez, reduciendo la probabilidad de explorar la mayoría de las regiones del espacio de búsqueda. Además no guarda información de los movimientos previos para guiar la selección de los nuevos movimientos. Esta técnica es usada en muchas aplicaciones, por ejemplo, en diseño de circuitos VLSI.

1.5 Dificultades en el uso de Algoritmos Genéticos

A la hora de llevar a la práctica el modelo de los Algoritmos Genéticos para resolver problemas no triviales de optimización se plantean varias dificultades, algunas de los cuales son las siguientes:

1. *La opacidad.* El AG serial es un algoritmo de búsqueda ciega. Esto es, sólo está guiado por la aptitud de los individuos y no incorpora ningún otro conocimiento específico del problema en cuestión.

2. *La finitud.* Debido a limitaciones propias del hardware, el AG sólo puede trabajar con poblaciones finitas no muy grandes de individuos. Lo contrario provocaría un bajo rendimiento en velocidad del AG, generando en muchos casos un algoritmo muy lento.

El primer punto puede llevar al *devío* de la búsqueda. Es decir, el AG serial realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud de los individuos para recombinar internamente los bloques constructores; en ocasiones ocurre que la información proporcionada (e.d., la aptitud) resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A esto se le llama *desorientación* o *decepción.* A nivel interno esto se concreta en que no se verifica la hipótesis de los bloques constructores, es decir, ciertas combinaciones válidas de buenos bloques constructores originan individuos de baja aptitud. En el peor de los casos puede ocurrir que este fenómeno tenga tanta fuerza como para impedir que el AG converja al óptimo global, desviándolo finalmente hacia un óptimo local. Entonces uno de los aspectos que debemos de considerar para un funcionamiento aceptable de un AG es la de proporcionarle una mínima cantidad (y calidad) de información. Si no hacemos esto el AG puede evolucionar incorrectamente.

La finitud de la población trae problemas de diversidad. La necesidad de que haya diversidad de individuos es entendible: con poca variedad de individuos hay poca variedad de esquemas. A causa de ello el operador de cruce pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. La necesidad de tener controlada la diversidad de aptitudes radica en la imposibilidad práctica de trabajar con una población infinita. Es fácil darse cuenta de que no es conveniente tener poca diversidad de aptitudes ya que en tal caso todos los individuos tendrían más o menos las mismas posibilidades de sobrevivir, la selección reproduciría la situación anterior y todo el peso de la búsqueda recaería en los operadores genéticos, lo que a la larga sería una búsqueda aleatoria. Por otra parte, cuando la población es finita -es decir, siempre- tampoco se puede tener gran disparidad de aptitudes, pues ello puede afectar muy negativamente a la diversidad de la población.

Un problema que también viene dado por la falta de diversidad ocurre cuando un individuo o un grupo de ellos obtengan una aptitud notablemente superior a los demás. Esto es especialmente probable en las fases tempranas de la evolución, en las cuales de entre una población de individuos mediocres suele surgir un buen candidato por aplicación de los operadores genéticos. En tal circunstancia existe el riesgo de que se produzca una *evolución en avalancha:* al incrementar los individuos más aptos su presencia en la población, por ser ésta finita la diversidad disminuye, ello hace que en la siguiente generación se favorezca aun más a los individuos más aptos hasta que éstos acaban dominando por completo la población. A esto se le conoce como el fenómeno de los **superindividuos.** Habitualmente ocurre que tales superindividuos sólo son los más aptos en cierto momento, pero no lo más aptos absolutos -la falta de diversidad estanca la búsqueda- lo que en último término provoca una **convergencia**

prematura del AG, habitualmente hacia un subóptimo. Este fenómeno es deseable en las fases tardías de la evolución, cuando el AG ha localizado correctamente la solución óptima, pero nunca antes.

Capítulo 2

Procesamiento paralelo en AGs

En este capítulo se describen inicialmente algunos conceptos básicos de programación paralela. Posteriormente, se propone una descomposición general de AG paralelos.

2.1 Nociones de paralelismo

La utilidad de las computadoras para describir y modelar de forma aceptable el mundo real, ha provocado la demanda creciente de potencia computacional a costos cada vez más bajos. Aunque hay grandes esfuerzos que se están realizando por conseguir velocidades de cómputo mucho mayores, existen problemas que son muy difíciles de abordar sobre arquitecturas secuenciales debido a que la densidad de integración tienen cierto límite. La computación paralela ofrece un camino para abordar estos problemas en un tiempo aceptable usando múltiples procesadores. El problema a resolver se divide en un conjunto de subproblemas que son tratados en varios procesadores. El objetivo de crear un código paralelo eficiente es utilizar el mínimo tiempo de cómputo, lo cual se obtiene si se dividen las cargas de trabajo entre todos los procesadores (balanceando la carga) mientras se minimiza la cantidad de información que se transfiere entre procesadores (que suele ser una penalización común en el procesamiento paralelo). Dos de los modelos creados para la implementación de algoritmos paralelos son: High Performance Fortran (HPF) para el modelo de programación de datos paralelos y el Message Passing Interface (MPI) para el modelo de programación de paso de mensajes.

Hace algunos años se tenía un esquema de paralelización muy conocido como *taxonomía de Flynn*, que se basaba en la indentificación de los flujos de control y datos entre las unidades de memoria y los elementos de procesamiento [35]. Esta clasificación estaba compuesta de las siguientes arquitecturas: *SISD, MISD, SIMD y MIMD*. Actualmente los estudios se basan solamente en la arquitectura MIMD *(Multiple Instruction Multiple Data)*, que consta típicamente de un número pequeño de procesadores independientes capaces de ejecutar flujos de instrucciones individuales, con posibilidad de que cada procesador ejecute un programa diferente. La arquitectura se suele dividir, dependiendo de la relación de los procesadores y la memoria. Esta división conduce a tres tipos importantes de arquitecturas MIMD: *memoria compartida, memoria distribuida y memoria compartida-distribuida.*

Memoria compartida

En la arquitectura con memoria compartida (ver Figura 2.1) hay un número pequeño de procesadores, cada uno de los cuales tiene acceso a una memoria global vía alguna red de interconexión. Un procesador se comunica con otro procesador escribiendo datos en una posición de memoria a la que el otro procesador puede acceder y leer. La ventaja de este tipo de arquitecturas es que es fácil de programar y no hay comunicaciones explícitas entre procesadores. El peor inconveniente de esta arquitectura es su pobre escalabilidad. Esto es debido a que a partir de una docena de procesadores, la red de interconexión no puede absorber el elevado número de interconexiones que producen las peticiones a memoria generadas desde los procesadores. Un método para evitar este conflicto de acceso a memoria consiste en dividir la memoria en múltiples módulos de memoria, cada uno conectado a los procesadores vía una red de interconexión. Sin embargo, este desarrollo tiende a desplazar el problema a la red de comunicaciones.

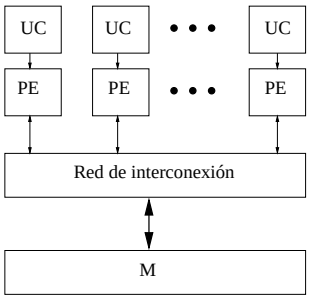

Figura 2.1: Arquitectura MIMD con memoria compartida, donde, UC-*unidad de control,* PE-*Elemento de proceso* y M-*Memoria compartida*

Memoria Distribuida

La arquitectura con memoria distribuida (ver Figura 2.2), evita los inconvenientes de la arquitectura de memoria compartida asignándole a cada procesador su propia memoria. Un procesador sólo puede acceder a la memoria que está conectada directamente a él. Si un procesador necesita datos que están en la memoria de un procesador remoto los adquiere pidiéndole una copia de ellos a dicho elemento remoto. Claramente el acceso a la memoria local es más rápido que el acceso a los datos de un procesador remoto. De ahí la importancia de la distancia entre procesadores, ya que cuanto más lejos esté el procesador remoto, más durará el intercambio de los datos entre procesadores.

Este tiempo de acceso no uniforme a memoria puede estar afectado por el modo en el que los procesadores están conectados. De ahí que surjan estudios de las diferentes topologías de interconexión como anillo, malla, hipercubo, estrella, etc.

Memoria Compartida/Distribuida

Por lo general las máquinas actuales son mezclas de los diferentes tipos de arquitecturas. Un ejemplo de esto es la así llamada arquitectura de memoria compartida/distribuida, que

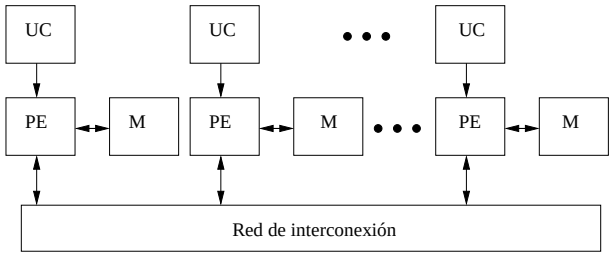

Figura 2.2: Arquitectura MIMD con memoria distribuida.

trata de combinar las ventajas de los modelos anteriores, esto es, la escalabilidad del modelo distribuido y la facilidad de uso del modelo compartido. Como en las máquinas de memoria distribuida, cada procesador tiene una memoria local, pero puede realizar accesos directos a memoria remota mediante el uso de un espacio de dirección global. Este acceso remoto es posible debido a la incorporación de un nivel de hardware intermedio de enrutado que se encarga de realizar las conexiones de forma automática y casi transparente para el usuario.

2.1.1 Construcción de programas paralelos

Para la construcción y desarrollo de algoritmos paralelos pueden distinguirse 4 etapas que se explicarán de manera breve.

- **Descomposición.** Es necesario dividir el cómputo en tareas. Así se identificará la concurrencia y se decidirá el nivel con el cual se explorará. También es importante tener suficientes tareas para mantener a los procesadores ocupados aunque no en exceso. Se pueden mencionar dos tipos de descomposición: *de dominio y funcional.*

- **Asignamiento.** En este mecanismo se especifica cómo dividir el trabajo entre procesos. Junto con la descomposición se le conoce como particionamiento. Es necesario tener un buen balance de carga para reducir los costos de comunicación y sincronización. El algoritmo resultante de las etapas anteriores es abstracto en el sentido de que no es especializado para que se ejecute en una arquitectura en particular.

- **Mapeo.** Se especifica dónde se ejecuta cada tarea. Cada proceso se asigna a un procesador de manera que se trata de maximizar la utilización de los procesadores y minimizar los costos de comunicación y sincronización. Se debe tratar de ubicar tareas que se pueden ejecutar concurrentemente en distintos procesadores para mejorar la concurrencia o poner las tareas que se comunican con frecuencia en el mismo procesador para incrementar la localidad. El problema del mapeo no aparece en computadoras con un solo procesador o en computadoras con memoria compartida.

- **Orquestación.** Marca la transferencia de datos entre tareas, por ello se necesitan mecanismos para acceder a los datos y mecanismos de comunicación y sincronización entre procesos. La arquitectura, el modelo de programación y el lenguaje de programación juegan un papel importante. En este paso se debe reducir el trabjo adicional *overhead* para controlar el paralelismo reduciendo costos de comunicación y sincronización en

una arquitectura real. Se encuentra por tanto mayor dependencia de la arquitectura usada.

Se describirán más a detalle los dos tipos de descomposición. Cuando se intenta paralelizar un algortimo secuencial, normalmente se han de detectar las partes inherentemente secuenciales y las potencialmente paralelas de dicho algoritmo. Frecuentemente, la descomposición introduce una sobrecarga adicional sobre las comunicaciones entre los procesadores. Dependiendo del tipo de la división del programa secuencial que se realice se obtienen dos diferentes clases de descomposición que se indican a continuación.

Descomposición Funcional

Un tipo de descomposición es el conocido *pipeline* que describiremos a continuación.

Pipeline. En esta estrategia cada conjunto de datos que funcionan como parámetros de entrada de cada una de las tareas en las que se divide el programa secuencial, atraviesa cada uno de los subprogramas en un orden dado. El paralelismo se introduce cuando se obtienen simultáneamente diferentes conjuntos de datos, de modo que cada uno de ellos sea una entrada para cada una de las etapas del pipeline.
El paralelismo en un pipeline está limitado por el número de etapas que se hayan podido definir. Para mantener la mayor eficiencia habría que mantener todas las etapas ocupadas, esto requiere que los tiempos de procesamiento de las diferentes etapas sean idénticos, de modo que el pipeline esté balanceado. Una posible extensión a la descomposición funcional consiste en repartir el trabajo de las etapas más costosas entre varios procesadores. A esta técnica se le conoce como *task farmming* [36].
En general la descomposición funcional es muy dependiente del problema y por lo tanto la capacidad de paralelización está limitada por el programa secuencial a paralelizar.

Descomposición de Datos

En este modelo de paralelismo se realiza una división de los datos asociados al problema a paralelizar, de modo que puedan asignarse subconjuntos de datos diferentes a cada procesador. Una vez realizada la división de los datos, hay que dividir la carga computacional de modo que cada procesador realice las operaciones relacionadas principalmente con su conjunto de datos. Existen numerosas técnicas de descomposición de datos. Una muy común es la que consiste en dividir el conjunto de datos inicialmente entre los procesadores. Esta división se ha de realizar de modo que las cargas computacionales de los procesadores sean prácticamente iguales, lo cual no implica que los subconjuntos de datos tengan por qué ser del mismo tamaño [27].
Existe también una descomposición dinámica de los datos, de modo que el reparto se realiza dependiendo de la evolución del algoritmo, asegurándose así un mejor balanceo de la carga computacional.

2.1.2 Alternativas para la programación paralela

A continuación se verán brevemente algunas de las alternativas más utilizadas y extendidas en la programación paralela, orientadas hacia la aplicación objeto de esta tesis.

Lenguajes de programación

Actualmente existen gran cantidad de lenguajes para programación paralela de datos, algunos de los más populares son el CM-Fortran, Fortran D, High Performance Fortran (HPF), entre otros. Con ellos emerge un método de programación en el que el programador y el compilador producen un código paralelo eficiente en poco tiempo de desarrollo. La idea principal bajo el paradigma de paralelismo de datos, heredada de las máquinas SIMD, es soportar completamente las operaciones sobre arreglos en paralelo. En general, la distribución de los datos y los cálculos es una tarea que implementará el compilador, guiado por las indicaciones que le proporciona el programador mediante ciertas directivas o anotaciones. Para códigos bien estructurados, estas herramientas pueden generar programas paralelos eficientes utilizando simples distribuciones de datos y cálculos. En este caso las ventajas de este paradigma son: facilidad y simplicidad de las implementaciones, portabilidad de código existente o futuro.

Bibliotecas de uso general

La programación paralela involucra una cooperación entre procesadores para resolver una tarea común. El programador primero tiene que definir los procesos que serán ejecutados por los procesadores especificando cuántos de aquellos procesadores tienen que sincronizarse e intercambiar información. Dentro de las bibliotecas de uso general tenemos a las bibliotecas de paso de mensajes donde los procesos se comunican y sincronizan enviando mensajes a otros procesos. Las arquitecturas paralelas que soportan un modelo de paso de mensajes suelen ser arquitecturas de memoria distribuida. Las operaciones de paso de mensajes se realizan mediante bibliotecas de paso de mensajes que son las responsables de establecer la comunicación física de la red en las que están conectadas las computadoras. Con las bibliotecas de paso de mensajes, el programador se olvida del compilador y se encarga de paralelizar el código de forma completamente manual, asumiendo un control total sobre éste. Para ello dispone de unas primitivas de paso de mensajes que se encargan de comunicar procesadores, el que envía los datos y el que los recibe. Actualmente existen diferentes bibliotecas de paso de mensajes para implementar códigos paralelos, algunos de ellos son: Message Passing Interface (MPI) [37, 40] y Parallel Virtual Machine (PVM) [38, 40] entre otros.

Bibliotecas de aplicación general

Se trata de la aproximación más sencilla desde el punto de vista del usuario, por cuanto el compilador genera de una forma sencilla la versión paralela del código secuencial. Dado que la programación es más complicada en las arquitecturas de memoria distribuida, su automatización ha estado tradicionalmente ligada a estos sistemas. Esta idea constituye el punto clave de la computación paralela, y ha sido perseguida con mucho entusiasmo por la comunidad científica. Sin embargo, los resultados no han sido muy buenos. Antes de que el

compilador pueda explotar el paralelismo inherente a un programa, necesita saber cómo se comporta ese programa, siendo un problema de difícil solución. Es por ello que a pesar de los esfuerzos que se han invertido y la multitud de estrategias que se han ideado para analizar, transformar y descomponer interactivamente aplicaciones secuanciales, sean tan pocas las que hoy día pueden paralelizarse de forma automática. Ejemplo de ello es el software creado para paralelizar aplicaciones en las que se usan Algoritmos Genéticos que aunque no son muy eficientes son muy simples de usar [1].

Hay diversas bibliotecas de aplicación general que facilitan la implementación de programas paralelos, únicamente es necesario adaptar algunas variables y elementos propios del problema que se pretende resolver.

Un conjunto de bibliotecas de propósito general es un paquete llamado **PGAPack** [39] que permite el desarrollo de AGs paralelos. Algunas de sus principales características son:

- Permite diferentes tipos de datos, binarios, enteros, reales y caracteres. Esto es importante ya que dependiendo del problema atacado se elegirá una determinada representación de cadenas genéticas. A su vez permite agregar nuevos tipos de datos.
- Proporciona diferentes niveles de acceso para usuarios novatos o expertos. Es posible generar un código genético muy general usando únicamente las funciones que proporciona el mismo paquete o bien generar un código más específico modificando y agregando nuevas funciones y operadores genéticos.
- Soporta diferentes tipos de operadores de selección, cruza y mutación.
- Soporta redes de estaciones de trabajo, arquitecturas en paralelo y uniprocesadores. El paquete contiene un compilador que permite la ejecución en paralelo de un algoritmo PGAPack.
- También cuenta con una gran cantidad de ejemplos y se dispone de un manual para el usuario.
- Es posible encontrar versiones de libre distribución en Internet.

2.2 Paralelismo en Algoritmos Genéticos

Dada la importancia de los Algoritmos Genéticos, se han utilizado numerosos modelos paralelos para mejorar su rendimiento tanto en velocidad como la velocidad de convergencia. Antes de describir en qué consisten estos modelos paralelos, se definirán algunos conceptos básicos propios de los algoritmos genéticos paralelos.

2.2.1 Descomposición de datos

Es la forma más común de paralelizar un AG y consiste en dividir la población total de individuos en subpoblaciones las cuales se asignan a procesadores diferentes. En cada procesador se ejecuta entonces un AG que tiene los tres operadores básicos de los genéticos secuenciales: *reproducción, cruce y mutación.* El algoritmo busca mejorar la subpoblación

con la ayuda de una función de evaluación (fitness) como en el caso secuencial. En un AG paralelo, la aptitud de un individuo es relativa a las aptitudes de su propia subpoblación mientras que en un AG secuencial la aptitud de un individuo es relativa a la población completa. En las versiones paralelas de los AG sólo pueden emparejarse entre sí los individuos pertenecientes a una misma subpoblación. Así diferentes procesadores hacen su investigación en diferentes espacios de búsqueda. Los procesos descritos antes paralelizan un AG mediante la *descomposición de datos.*

2.2.2 La migración y sus parámetros

En un AG formado por una gran población, un individuo con un valor muy bueno de la función de aptitud, mejorará el valor medio de la población. Este efecto ocurre porque el mecanismo de supervivencia del más apto provoca que este individuo apto se reproduzca con mayor frecuencia que uno menos apto. En un AG paralelo, la población se divide en distintas subpoblaciones, pero si no hay comunicación entre las subpoblaciones un buen individuo sólo podrá mejorar el valor de la subpoblación a la que pertenece, pero es incapaz de mejorar el resto de las subpoblaciones. Así, cada subpoblación intenta mejorar su propio valor sin afectar la ejecución del resto; las poblaciones están totalmente aisladas entre sí. En este caso no hay comunicación entre las subpoblaciones.

Tanese [14] experimentó con un AG paralelo dividido, lo comparó con un AG secuencial examinando los resultados sobre cinco ejecuciones; hizo pruebas de sus algoritmos usando un gran número de funciones test (polinomios Walsh generados aleatoriamente [4]). Para estudiar su comportamiento escogió dos tipos de medidas: *(1) Mejor Individuo y (2) Valor Medio Final.* El mejor individuo es el mejor valor que el algoritmo ha encontrado de entre todas las ejecuciones realizables. El valor medio final es el mejor valor medio encontrado por el algoritmo en cualquier ejecución. Esto significa que el valor de los individuos se mediatiza en la última generación de cada ejecución y se compara con el valor final medio de otras ejecuciones. Tras la observacin de todas las medias, resultó que un AG paralelo es capaz de encontrar mejores soluciones que un AG secuencial, ya que el valor Mejor Individuo era mejor. Sin embargo, el Valor Medio Final es menor. Una posible explicación de estos resultados es que las poblaciones más pequeñas tienden a converger antes. En una implementación paralela las subpoblaciones son pequeñas, comparadas con la gran población utilizada en un AG secuencial. Las subpoblaciones aisladas de un AG paralelo se fijan pronto y sólo la mutación puede causar algún cambio en esas subpoblaciones. Esta fijación temprana provoca una ejecución media bastante mala, pero debido al particionamiento, se investigan muchos esquemas diferentes y es posible que se encuentre una solución muy buena en una subpoblación. Esto se debe a que cada subpoblación evoluciona en un camino único y aunque cada subpoblación puede estar dominada por ciertos individuos con una aptitud muy alta, estos individuos son diferentes en cada subpoblación. Dados estos resultados Tanese investigó un AG paralelo con migración. Enviaba individuos escogidos pseudo aleatoriamente de cada subpoblación hacia subpoblaciones vecinas. Esta migración se hacía regularmente. El AG con migración mostró que era posible combinar lo mejor de ambas técnicas, secuencial y dividida. La adición de la migración al AG paralelo incremento el Valor Medio Final de las subpoblaciones, y el mejor

individuo era mejor que en los AGs secuenciales.

Paramétros importantes para implantar un operador de migración son: *tamaño de la población, número de comunicaciones, frecuencia de intercambio y número de intercambios.*

El **tamaño de la población**, donde debe existir un compromiso entre la cantidad de búsqueda genética (una gran población) y el tiempo de ejecución.

El modo de **comunicación de las subpoblaciones** es un parámetro que puede controlar de alguna forma la convergencia prematura, debido a que una restricción en la cantidad de individuos comunicables intenta evitar la dominancia de individuos casi óptimos, es decir, la diversidad de la población es más alta si los canales de comunicación están limitados de algún modo. De este modo la exploración funciona mejor ya que diferentes subpoblaciones investigan diferentes picos de la función objetivo.

La **frecuencia de intercambio** es un parámetro importante y se define como el inverso o recíproco del número de generaciones entre dos migraciones. Si la migración de individuos ocurre muy seguido, la dominancia de un único individuo amenaza la diversificación de la población. Por otro lado, si la frecuencia de intercambio es demasiado baja, ciertos procesadores pueden estar ocupados explorando áreas prometedoras mientras que otros pueden estar perdiendo su tiempo con una población de individuos poco apta. Para determinar la mejor proporción de intercambio, deben ejecutarse diferentes pruebas en el AG paralelo, teniendo en cuenta que la frecuencia de intercambio tiene un gran impacto sobre la cantidad de comunicación en el algoritmo paralelo.

Otro parámetro es el **número de intercambios** que representa la cantidad de individuos que pueden migrar desde cada subpoblación. Un gran número de intercambios puede entorpecer la comunicación debido a la sobrecarga de las comunicaciones, provocando que el AG paralelo se comporte como un AG secuencial. Las subpoblaciones paralelas deben tener la posibilidad de desarrollarse sin demasiadas interferencias y por el contrario el número de intercambios, en comparación con el tamaño de la población no debe ser demasiado pequeño tampoco.

En las siguientes secciones se describirán los esquemas paralelos más utilizados para Algoritmos Genéticos así como algunas de las razones por las que se hace necesaria la paralelización en este tipo de algoritmos.

2.2.3 Modelos para la paralelización de los AGs

Los AGs secuenciales han tenido mucho éxito en diferentes dominios de aplicación, pero existe una gran cantidad de desventajas en su utilización, algunas de ellas son las siguientes:

- Son propensos a perderse en la búsqueda de la solución óptima.
- Las evaluaciones y los operadores genéticos consumen mucho tiempo de cómputo.

- El tamaño de las poblaciones necesitan ser grandes en la mayoría de los problemas.

Con el fin de resolver estas deficiencias se ha diseñado una gran diversidad de modelos paralelos para los Algoritmos Genéticos, entre los cuales se han propuesto los siguientes [26]:

1. Modelo de paralelización global.
2. AG paralelo de islas.
3. AG paralelo celular.

2.2.4 Paralelización global

El método más directo de paralelizar cualquier algoritmo secuencial consiste en hacer una paralelización **global**. En este tipo de algoritmos paralelos existe un procesador maestro y varios esclavos (ver Figura 2.3). Lo más usual en este tipo de paralelización es que el procesador maestro sea el encargado de ejecutar el algoritmo de optimización secuencial básico, o bien de tomar decisiones globales, y los procesadores esclavos sean los encargados de realizar búsquedas locales a partir de los puntos indicados por el procesador maestro, o bien simplemente de evaluar tales puntos. Si el costo de las comunicaciones no domina sobre el costo de cálculo se puede esperar una aceleración significativa. En un principio las implementaciones de este tipo solian ser síncronas de modo que el procesador maestro necesita conocer los resultados de los procesadores esclavos antes de tomar alguna decisión o comenzar alguna otra iteración; pero recientemente también se están diseñando estrategias asíncronas que permiten que los tiempos de comunicación no influyan tanto en el proceso paralelo. No obstante, estos algoritmos paralelos son más complicados de diseñar y no son tan parecidos a los correspondientes secuenciales.

En este tipo de genético paralelo solo hay una población, como en el AG convencional, pero la evaluación de los individuos y los operadores genéticos se paralelizan de forma explícita. Puesto que sólo hay una población, la selección considera a todos los individuos y cada individuo tiene oportunidad de aparearse con cualquier otro. Por lo tanto, el comportamiento del AG global permanece sin cambios. El esquema se muestra en la Figura 2.3.

2.2.5 AG paralelo de islas

Se les llama AGs de isla haciendo alusión a un modelo poblacional usado en genética en el cual se consideran subpoblaciones relativamente aisladas (ver Figura 2.4). También son conocidos como AGs de grano grueso[1]. En este caso cada procesador ejecuta un programa de optimización que evoluciona independientemente del resto durante la mayor parte del tiempo, aunque ocasionalmente intercambia información (normalmente puntos óptimos

[1] El tamaño del grano en paralelismo se refiere a la razón entre el tiempo empleado en cómputo y el tiempo de comunicación. Cuando esta razón es grande (mayor tiempo gastado en cómputo que en comunicación) el procesamiento es llamado de grano grueso.

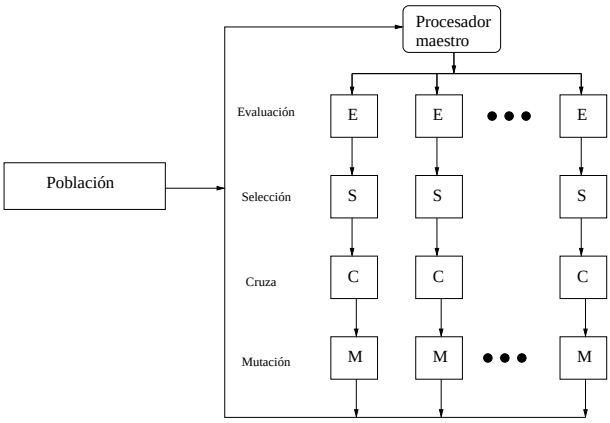

Figura 2.3: Modelo de paralelización global para un AG.

encontrados hasta ese momento) con algún otro procesador de modo que la información recibida por cada procesador puede influir el desarrollo del algoritmo. En el contexto de los AGs paralelos de grano grueso al intercambio se le conoce como migración, considerado como un nuevo operador genético. En el caso de algoritmos que trabajen con poblaciones de puntos, en este tipo de estrategias de paralelización, se suele dividir la población en múltiples subpoblaciones (**demes**) que evolucionan independientemente unas de otras mientras que no haya intercambios de puntos. Estos algoritmos de grano grueso (basados en subpoblaciones) introducen cambios fundamentales en las operaciones del algoritmo secuencial y tienen un comportamiento diferente con respecto a la versión secuencial. Algunas de las razones por la que los AGs de grano grueso son muy usados son:

1. Se implementan de forma simple: se toman unos cuantos AGs secuenciales, se ejecuta cada uno de ellos en un procesador diferente y, con ciertos intervalos de tiempo se hace intercambio de individuos. Se requiere de rutinas adicionales para hacer las comunicaciones entre subpoblaciones y efectuar con ello el proceso de migración.

2. Si no se tiene una arquitectura paralela, pueden implementarse a través de una simulación efectuada en una red de estaciones de trabajo. Para ello se pueden usar bibliotecas de paso de mensajes como MPI o PVM.

Los parámetros que requieren los AGs paralelos de islas son:

- Número y tamaño de los demes, lo cual puede estar determinado por el hardware disponible.
- Estructura de interconexión o topología.
- Intervalo de migración o frecuencia de intercambio, que indican cada cuántas generaciones se efectúa la migración.

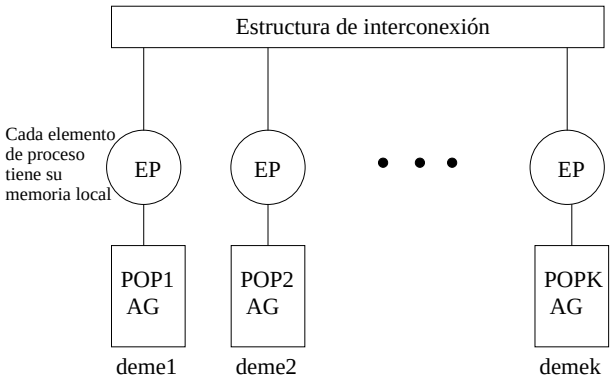

Figura 2.4: Modelo de paralelización de grano grueso para un AG, donde, EP-*Elemento de proceso*, POP_k-*Polación del* EP_k

- Tasa de migración o número de intercambios, que indica cuántos individuos se deben de migrar.

- Radio de selección, generalmente el radio es cero indicando con ello que la selección se hará sólo en su misma subpoblación.

- Radio de migración, que también puede estar determinado por la topología.

Algunos de estos parámetros como por ejemplo la topología, juegan un papel muy importante en el desempeño del AG ya que determina qué tan rápido se disemina una buena solución hacia los otros demes. Si se usa una topología dispersamente conectada, las soluciones se diseminarán más lentamente y los demes estarán más aislados entre sí, permitiendo la aparición de soluciones diferentes, favoreciendo probablemente la diversidad. Si usa una topología densamente conectada puede promover una mejor mezcla de individuos, pero a un costo computacional más alto. Por ejemplo, una topología densa tiende a encontrar soluciones globales con un menor número de evaluaciones de la función de aptitud que si se usa una topología dispersa.
Cuatro de las topologías usadas en el presente trabajo se muestran en la Figura 2.5 y se describen brevemente a continuación.

- **Topología de estrella.** En esta topología uno de los nodos está conectado a todos los demás (nodo central). El costo básico de este sistema es lineal respecto del número de nodos. El costo de comunicaciones es también bajo, ya que un mensaje que va de un nodo a otro (considerando que ninguno de ellos es el nodo central) requiere de dos transferencias. Sin embargo esta velocidad puede no ser tal ya que el nodo central puede convertirse en un cuello de botella. Mientras haya pocos mensajes la velocidad se mantendrá alta. En algunos sistemas estrella el nodo central está totalmente dedicado al pasaje de mensajes. El número de ligas que contiene el nodo central es $n-1$ y el resto contiene una sola liga.

- **Topología de red totalmente conectada.** En esta topología cualquier nodo en la red está conectado a todos los otros nodos de la red. El costo básico de esta configuración es alto, ya que debe existir una conexión directa entre cada dos nodos. El costo de anexar un nuevo nodo crece según crece la cantidad de nodos que contiene la red. Sin embargo, en este entorno los mensajes entre nodos pueden entregarse muy rápidamente, debido a que hay una sola conexión para viajar entre dos nodos. El número de ligas que contiene cada nodo es $n-1$, donde n es el número total de nodos.

- **Topología de anillo.** Cada nodo está conectado a sólo otros dos nodos. El anillo puede ser unidireccional o bidireccional. Si es unidireccional un nodo puede transmitir información hacia uno solo de sus vecinos. Todos los nodos entregan información hacia la misma dirección. En una arquitectura bidireccional un nodo puede trasmitir información hacia cualquiera de sus vecinos. El costo básico de un anillo es nuevamente lineal con respecto a la cantidad de nodos. Sin embargo el costo de comunicación puede ser bastante alto y que un mensaje de un nodo a otro debe viajar alrededor del anillo hasta que llegue a su destino. En un anillo unidireccional debe recorrer a lo sumo $n-1$ nodos, en tanto que en uno bidireccional a lo sumo $n/2$.

- **Topología de hipercubo.** En la Figura 2.5 se ilustra un cubo tridimensional. Este concepto de cubo puede extenderse a un espacio de dimensión n obteniéndose el $n-$ *cubo* con n bits para cada vértice. Una red $n-$ *cubo* corresponde con N nodos donde $n = log_2 N$. En un $n-$ *cubo* cada nodo está conectado con exactamente n vecinos. Esos vecinos difieren exactamente en un bit. La comunicación entre los procesadores es mediante caminos redundantes, por ello pueden ocurrir sobrecargas internas. Para un cubo $n-$ *dimensional* con 2^n nodos, la máxima distancia de comunicación entre dos nodos es $n = log_2 N$ y es a su vez el número total de ligas que puede tener un nodo.

También es posible usar topologías dinámicas, en las que los demes no están limitados a poder comunicarse sólo con un cierto conjunto predefinido de subpoblaciones, sino que envía sus migrantes a aquellas subpoblaciones que satisfacen ciertos criterios. La idea de este esquema es que puedan identificarse las subpoblaciones donde los migrantes tienen mayores probabilidades de producir algún efecto. Usualmente, se usa la diversidad como criterio principal para definir qué tan adecuada es una subpoblación.

Un concepto relacionado con las topologías es el de vecindario, que se refiere al área dentro de la cual puede moverse un migrante de una subpoblación. Finalmente, asociado al concepto de vecindario tenemos al radio de selección, que se refiere a la cantidad de vecinos entre los cuales se puede efectuar la selección. Es común usar un radio de selección de cero, es decir, efectuar la selección sólo dentro de la misma subpoblación.

2.2.6 AG paralelo celular

También conocido como AG paralelo de grano fino, es el tercer desarrollo en la paralelización de los algoritmos de optimización global y utiliza un paralelismo de **grano fino** (ver Figura 2.6), donde el número de procesadores es mucho más elevado y éstos establecen comunicaciones con todos los procesadores vecinos. En algoritmos de optimización que sólo

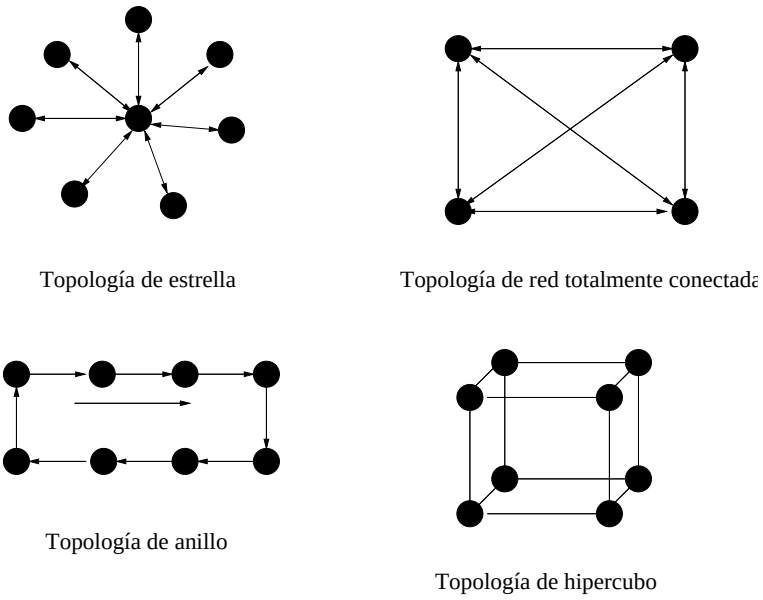

Figura 2.5: Algunas topologías usadas para AG de grano grueso.

guardan información de un único punto, este paralelismo se reduce a una división de tareas, en la que cada procesador hace una tarea (no muy costosa computacionalmente). En algoritmos que trabajen con poblaciones de puntos se establece una descomposición de datos de modo que se divide la población total en un gran número de pequeñas subpoblaciones. De hecho el caso ideal es tener sólo un individuo en cada elemento de proceso disponible. Este modelo está adaptado para computadoras masivamente paralelas, pero puede ser implementado en un multiprocesador.

Este modelo también introduce cambios fundamentales en el modo de trabajar de los algoritmos con respecto a la versión secuencial.

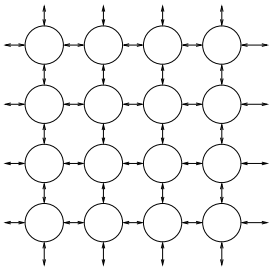

Figura 2.6: Esquema de la estrategia de paralelización de grano fino.

En un AG de grano fino los individuos se pueden distribuir en una malla bidimensional (ver Figura 2.6), donde, cada uno de los nodos es un elemento de proceso al cual corresponde una sola solución al problema. Con este esquema la comunicación es intensiva entre las subpoblaciones (individuos), y es esta comunicación masiva la que proporciona un modo de

diseminar las soluciones buenas o malas por toda la población.

Así mismo, los individuos en la malla se organizan formando vecindarios conocidos comunmente como demes; los individuos sólo pueden migrar hacia otro procesador que pertenezca al mismo deme, lo cual dependerá del tipo de deme que se esté trabajando. Por lo regular las demes constan de 4 a 10 procesadores (aunque pueden contener más elementos) y existen dos tipos: *lineales o compactos*. En los vecindarios lineales L_r, los vecinos de un punto dado incluyen a las $r-1$ estructuras más cercanas elegidas sobre los ejes horizontal y vertical (Figura 2.7). En el caso del vecindario compacto C_r, el vecindario incluye a los $r-1$ individuos más cercanos. El tipo de elección puede dar lugar a vecindarios compactos cuadrados o diamante. Esta distinción puede extenderse a vecindarios que cambian en el tiempo o en los que los individuos eligen su pareja tras un paseo aleatorio hasta ella.

Puede observarse en la Figura 2.7 que atendiendo al radio y a qué vecinos se incluyan distinguimos entre distintos vecindarios. Es de esperarse que los vecindarios pequeños como L5 sobrecarguen poco al sistema. En cuanto a los vecindarios grandes, es interensante observar, como ocurre con el vecindario C25 del ejemplo, que algunos pueden llegar a incluir a todos los elementos de la malla. En el caso extremo de vecindarios muy grandes y mallas pequeñas el efecto es similar al de usar una única población.

La difusión de estructuras (Figura 2.8) se lleva a cabo por el solapamiento de vecindarios, ya que cada una de las estructuras pertenece al vecino de sus cuatro puntos adyacentes de la malla. De esta forma, la aparición de una solución en un punto de la malla se difunde a sus vecinos en el primer paso siguiente al de su descubrimiento y éstos, a su vez, la difunden en pasos sucesivos, ya que la política de reemplazo suele requerir que la nueva estructura calculada por los operadores genéticos sea mejor que la considerada actualmente para reemplazar a esta última.

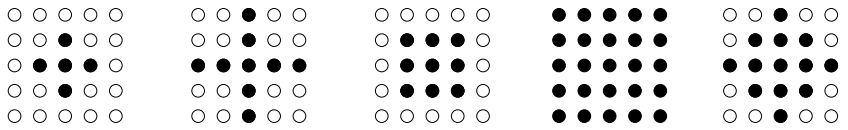

Figura 2.7: Dos posibles tipos de vecindarios: lineales y compactos.

2.2.7 Esquema híbrido

Otras formas de implementar los AGP es combinando los esquemas descritos antes, que son conocidos como híbridos. Un posible híbrido consiste en usar un AG de grano fino a bajo nivel y otro de grano grueso a alto nivel, tal y como se muestra en la Figura 2.9. Un ejemplo de este híbrido es el AG propuesto por Gruau [29], en el cual la población de cada deme se coloca en una malla bidimensional y los demes se conectan entre sí en forma de toroide

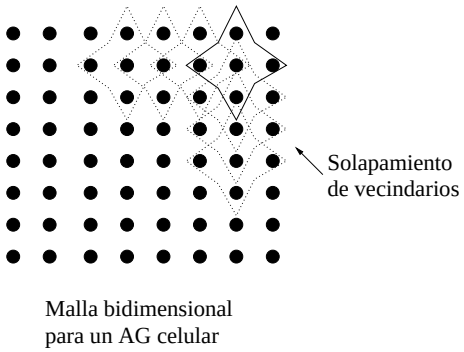

Figura 2.8: Solapamiento entre vecindarios dentro de una malla bidimensional.

bidimensional. La migración entre los demes vecinos ocurre a intervalos regulares.

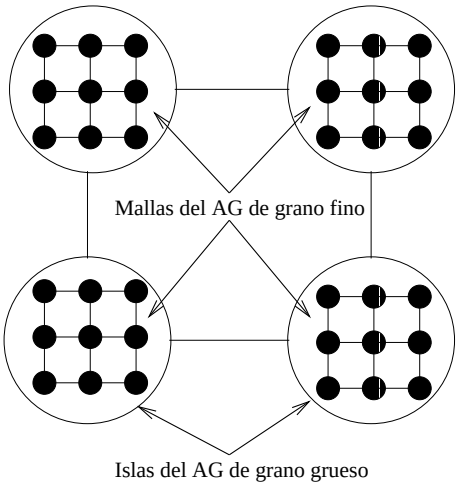

Figura 2.9: Esquema híbrido donde se combina un AG de grano grueso a alto nivel con un AG de grano fino a bajo nivel.

A continuación se describen los elementos de un AG implementado para resolver el problema del agente viajero (TSP) y finalmente se hará un breve análisis de las limitaciones de dicho algoritmo, proponiendo algunas alternativas para mejorar su desempeño.

Capítulo 3

Algoritmos genéticos para el TSP

Dada la dificultad para resolver problemas NP (ver apéndice B) a través de métodos convencionales se han diseñado una gran cantidad de técnicas no comunes, entre ellas los algoritmos genéticos los cuales han mostrado una efectividad relativa para problemas de optimización combinatoria. En estos problemas los AGs tienen problemas debido a que el espacio de búsqueda aunque es finito es sumamente grande y los algoritmos genéticos trabajan con poblaciones de soluciones cuyo tamaño es mucho menor si se compara con el espacio de soluciones del problema. Generalmente la función de evaluación del problema es relativamente fácil de evaluar aunque en algunas ocasiones el principal problema se puede encontrar en los operadoradores genéticos, en los cuales puede requerirse mucho tiempo de cómputo. Por lo tanto, dos de los factores principales a tomar en cuenta al atacar un problemas de optimización combinatoria son, la convergencia y la velocidad.

Dada la importancia de los problemas NP-Completos y sabiendo que la solución de un problema que sea considerado NP-Completo repercute en todos los demás del mismo tipo, el problema del agente viajero (TSP) se seleccionó en este proyecto para el análisis de rendimiento de los algoritmos genéticos por ser un problema clásico.

3.1 Los problemas de optimización combinatoria

Un proceso de optimización tiene como objetivo hallar el valor máximo o mínimo de una cierta función, definida en un dominio. En los problemas de decisión que generalmente se presentan en la vida empresarial, por lo general existen una serie de recursos escasos (personal, presupuesto, tiempo), o de requisitos mínimos a cumplir (producción, horas de descanso), que condicionan la elección de la estrategia más adecuada. Por lo general, el objetivo al tomar la decisión consiste en llevar a cabo el plan propuesto de una manera óptima: mínimos costos o máximos beneficios. La resolución de este tipo de problemas atrajo la atención de numerosos investigadores, principalmente desde la Segunda Guerra Mundial, con el florecimiento de la investigación de operaciones.

Existe un tipo concreto de problemas de optimización, especialmente interesantes denominados como problemas de optimización combinatoria. En ellos, las variables de decisión son enteras y por lo general el espacio de soluciones está formado por permutaciones o subconjunto de números naturales (de ahí su nombre). En el caso de la optimización combinatoria, se

trata de hallar el mejor valor entre un número finito de posibilidades, pero la enumeración de este conjunto de posibilidades tomaría un tiempo considerable por lo que sería prácticamente imposible, aún para instancias de tamaño moderado.

3.1.1 El problema del agente viajero

Uno de los problemas más famosos, y quizá el más estudiado, en el campo de la optimización combinatoria, es el problema del agente viajero (TSP, por sus siglas en inglés: Traveling Salesman Problem). La historia del TSP desde el punto de vista algorítmico ha evolucionado gracias a que este problema ha sido una excelente plataforma de prueba para la introducción de buenos algoritmos aunque no necesariamente óptimos en la solución, tales como *recocido simulado, búsqueda tubú, algoritmos genéticos, entre otros.*
La variante más habitual del TSP, el TSP asimétrico (al cual nombraremos simplemente TSP en este trabajo), consiste en una distribución de n ciudades en un plano bidimensional. Cada ciudad es alcanzable desde cualquier otra ciudad mediante un camino cuya longitud o costo debe estar definido. Retomando la definición de [19], el TSP se formula de la siguiente manera: un agente viajero, partiendo de su ciudad de origen, debe visitar exactamente una vez cada ciudad de un conjunto de ellas (previamente especificado el número de ciudades) y regresar al punto de partida. Un recorrido con estas características, es llamado dentro de este contexto un *tour*. El problema consiste en encontrar el tour para el cual la distancia total recorrida sea mínima. Se asume que se conoce para cada par de ciudades, la distancia entre ellas. La Figura 3.1 ilustra un tour en una instancia de ocho ciudades, representada por un grafo donde cada nodo del grafo corresponde a una ciudad y cada arista que une a un par de nodos representa la parte del tour que pasa por dichos nodos. En la Figura 3.1 se ilustra el tour que visita las ciudades 1, 2, 3, 8, 5, 4, 7, 6 y 1, en ese orden.

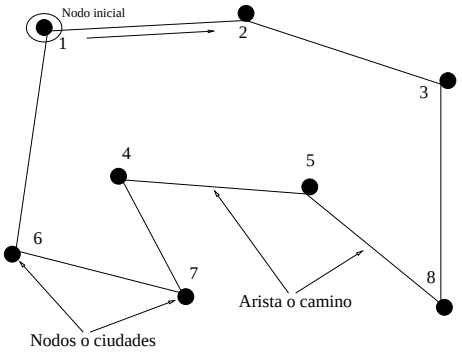

Figura 3.1: Ejemplo de un tour para el TSP de ocho ciudades.

Una definición más formal del TSP sería la siguiente: podemos plantear el TSP como un problema en el que cada solución se representa como una permutación, dado un número entero $n \geq 3$ y dada una matriz $C = (c_{ij}) \in M(n, n)$, con elementos c_{ij} enteros no negativos, se trata de encontrar la permutación cíclica π de los enteros de 1 a n que minimiza $\sum_{i=1}^{n} c_{i\pi(i)}$.

Ciudades	Soluciones	Tiempo
2	1	0.0053
4	6	0.0.0584
6	120	0.3360
8	5040	1.6190
10	362880	5.3770
12	39916800	37.8298
14	6227020800	2096.5054
16	1.3077E+12	35825.4870
18	3.5569E+14	171899.7670
20	1.216E+17	347835283.8565

Tabla 3.1: Crecimiento del espacio de búsqueda para el problema del Agente Viajero. La columna Tiempo se representa en segundos

3.1.2 Complejidad del TSP

El TSP es un problema NP-Completo, donde el espacio de estados o posibles soluciones crece de forma factorial ante incrementos pequeños en el tamaño del problema [19]. Una función factorial es muy similar a una función de complejidad exponencial lo que da una idea más clara de lo difícil que resulta una búsqueda exhaustiva de la solución óptima al problema. En general se puede ver que para un problema de n nodos o ciudades el número de recorridos es $(n-1)!$ La representación abreviada de cualquier recorrido para el problema del TSP de n nodos, se escribiría de la siguiente manera:

$$[p_0, p_1, p_2, ..., p_n, p_0]$$

el cual representa una de las $(n-1)!$ posibles soluciones.

La Tabla 3.1 y la Figura 3.2 muestran el crecimiento en tiempo para explorar todo el espacio de búsqueda del TSP para diferentes tamaños de problema. Se tomaron tiempos experimentales para problemas menores a 12 nodos y para problemas mayores se extrapolaron los resultados.

La implicación directa de un problema difícil de resolver es que cualquier algoritmo empleado para encontrar la solución óptima tomaría un tiempo de cómputo que crece exponencialmente ante incrementos lineales en el tamaño de los datos del problema. Aparece entonces la necesidad de emplear heurísticas para tratar este tipo de problemas. Las heurísticas son procedimientos que aunque no garantizan una solución óptima al problema, obtienen soluciones aproximadas de alta calidad en un tiempo de ejecución razonable.

3.1.3 Aplicaciones del TSP

Los problemas básicos de optimización de rutas son muy diversos y tienen además múltiples aplicaciones a la solución de otros problemas similares. Un modelo como el agente

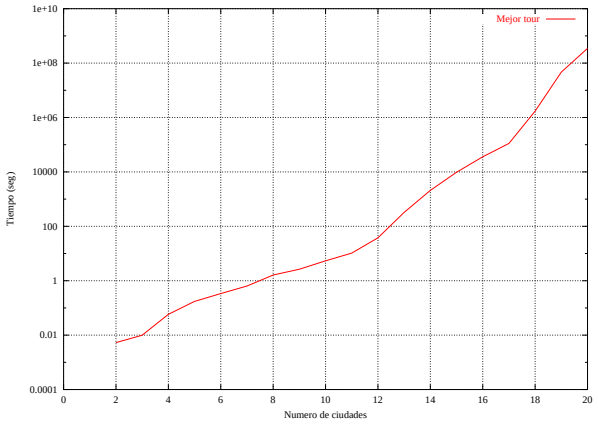

Figura 3.2: Tiempo requerido por un algoritmo convencional para resolver el TSP.

viajero se puede aplicar a situaciones útiles, semejantes al TSP o aplicaciones diferentes. Algunas de las aplicaciones que se pueden observar son las siguientes: *recorrido de recolectores, perforación de tarjetas de circuitos impresos y control de robots.*

3.2 Implementación del TSP con AGs

El Algoritmo 1 muestra el pseudocódigo del AG secuencial general para el problema del TSP y en las siguientes secciones se describen cada uno de los procedimientos que integran el AG para el TSP.

Algoritmo 1 AG secuencial general para el TSP

Entrada: $totalGeneraciones > 0, numeroCiudades > 0, tamañoPoblacion > 0$
Salida: $mejorIndividuo, peorIndividuo, numGeneracion, tiempoBusqueda$
Crear la primera población de individuos
while $totalGeneraciones <> 0$ **do**
  $totalGeneraciones \Leftarrow totalGeneraciones - 1$
  Evaluar a todos los individuos de la población
  Aplicar esquema elitista
  if $totalGeneraciones <> 0$ **then**
    Aplicar operador de selección
    Aplicar operador de cruza
    Aplicar operador de mutación
  end if
end while

El pseudocódigo 1 muestra 6 pasos principales que conforman el AG secuencial, sin embargo como se verá más adelante hay otros puntos que es necesario considerar si se espera obtener un AG adecuado.

Al pensar en una implementación que resuelva el TSP aparecen inmediatamente dos problemas: el primero de ellos surge de la definición del TSP y es que no puede permitirse que una ciudad se repita, y no deben permitirse los números de ciudad fuera del rango especificado. Por lo tanto, deberan revisarse la manera en la que serán generados los individuos de la primera población, evitando las cadenas inválidas. El segundo problema se relaciona con el primero y es que, aun forzando a la primera población para que cumpla con las dos restricciones anteriores, se debe ser cuidadoso al aplicar cruzamiento y mutación ya que las cadenas resultantes pueden inducir a ciudades repetidas o no válidas.
Las alternativas para enfrentar estos problemas son variadas (descrito en el primer capítulo), pero la usada en este trabajo, es aquella donde se usan operadores especialmente diseñados para cadenas de permutación.

En las siguientes secciones se describen las estrategias y algoritmos usados para la implementación del AG para el TSP. Al finalizar la descripción de cada algoritmo se muestra un breve análisis del orden de complejidad.

3.2.1 Codificación de las cadenas genéticas

La representación usada para los individuos de la población en el problema del agente viajero es formar cadenas de identificadores (números enteros) sin repetición o permutaciones. Cada identificador constituye un alelo, un ente indivisible que no puede ser partido ni alterado internamente, el análogo de cada bit en una representación binaria. Se establecen las restricciones de que en un individuo debe aparecer cada identificador de ciudad una sola vez, y que todos los individuos tienen el mismo primer identificador.

Una ruta puede ser representada como una permutación, la cual se escribirá como una lista de las n ciudades en el orden en el que se recorren. El Algoritmo 2 describe la forma de generar la población inicial de cromosomas, donde P es el tamaño de la población y n representa el número total de ciudades consideradas en el problema.

Este algoritmo proporciona una población inicial de n cadenas válidas para el problema del agente viajero.

Análisis. En el algoritmo se observan dos estructuras importantes para el análisis, la primera de ellas, el ciclo **for** más interno que se ejecuta $n-1$ veces. La segunda, el ciclo **for** externo que se ejecuta P veces. Dado que el primer ciclo **for** se encuentra dentro del segundo ciclo, ambos se ejecutan $(n-1)*P$ veces. A n se resta un valor contante (-1), pero este valor resulta poco significativo al incrementar el número de ciudades, por lo tanto, la función de tiempo es: $n*P$ y el orden de complejidad es $O(n*P)$.

Ahora se debe asegurar que los operadores genéticos mapeen individuos válidos en individuos válidos. Para ello se utilizan aquí operadores adecuados para codificación de permutaciones. Antes de los operadores se analizará la función de evaluación.

Algoritmo 2 Generar la primera población de cromosomas

Entrada: P-tamaño de la población, n-número de ciudades
Salida: *poblacionInicial*
for $i = 0$ to P **do**
  Generar de forma aleatoria un número (*alelo*), que será la ciudad inicial para el tour i
  $alelo \Leftarrow rand()\ MOD\ n$
  Colocar el *alelo* en la primera posición del código genético del *i-ésimo* individuo (primer alelo)
  Generar los siguientes $n - 1$ alelos
  for $j = 1$ to n **do**
    Generar el siguiente alelo de manera aleatoria y diferente a los ya generados
    $alelo \Leftarrow (alelo + NUMEROPRIMO)\ MOD\ n$
    Colocar el *alelo* obtenido en la posición *j-ésima* del individuo *i-ésimo*
  end for
end for

3.2.2 Función de Evaluación

Se pretende minimizar el costo de un viaje redondo que pase por todas las ciudades, es decir, para un problema de n ciudades, donde se representa un tour cualquiera, digamos el tour *j-ésimo* como:

$$[k_1, k_2, k_3, ..., k_n]$$

donde cada k_i es una ciudad que es visitada sólo una vez en cada recorrido. El costo asociado al recorrido j se calcula de la siguiente forma:

$$aptitud_j = f(k_{jn}, k_{j1}) + \sum_{i=1}^{n-1} f(k_{ji}, k_{j(i+1)}) \tag{3.1}$$

Generalmente para los problemas en los que son aplicados los AGs se requiere maximizar el resultado y obtener así los individuos mejor adaptados. No es el caso del TSP, donde se requiere obtener la solución de mínimo costo, pero, este problema de minimizar se puede traducir en un problema de maximizar obteniendo el inverso multiplicativo del valor de evaluación de un individuo, es decir,

$$aptitud_j = \frac{1}{fa_j} \tag{3.2}$$

donde fa_j es el valor de evaluación del individuo *j-ésimo*. Así, al maximizar esta función se está minimizando implícitamente la función objetivo. El algoritmo 3 describe brevemente el código implementado para obtener el valor de aptitud de cada uno de los individuos en la población, donde P es el tamaño de la población, n representa el número total de ciudades consideradas en el problema y $aptitudCromosoma_i$ es el valor de evaluación del individuo *i-ésimo* en la población.

Algoritmo 3 Obtener el valor de aptitud de cada individuo en la población

Entrada: P-tamaño de la población, n-número de ciudades
Salida: Valor de aptitud de cada uno de los elementos de la población
for $i = 0$ to P **do**
  $aptitudCromosoma_i \Leftarrow 0$
  for $j = 0$ to $n - 1$ **do**
    $aptitudCromosoma_i \Leftarrow$ **(costo de ir de la ciudad** j **a la ciudad** $j + 1$**) +** $aptitudCromosoma_i$ **acumulada**
  end for
  Se agrega el costo de la última arista de la gráfica que cierra el ciclo del tour
  $aptitudCromosoma_i \Leftarrow$ **costo de ir de la última ciudad del tour a la ciudad de inicio.**
  $aptitdCromosoma_i \Leftarrow 1$ / $aptitudCromosoma_i$
  Almacenar valor de aptitud del cromosoma i
end for

Análisis. En el algoritmo se observan dos estructuras importantes para el análisis, la primera de ellas, el ciclo **for** más interno que se ejecuta $n - 1$ veces. La segunda, el ciclo **for** externo que se ejecuta P veces. Debido a que el primer ciclo **for** se encuentra dentro del segundo, ambos se ejecutan $(n-1) * P$ veces. Nuevamente el valor contante (-1) que se resta a n resulta poco significativo al incrementar el número de ciudades, por lo tanto, la función de tiempo es: $n * P$ y el orden de complejidad es $O(n * P)$.

3.2.3 Mecanismos para la selección de cadenas

Una vez calificados todos los individuos de una generación, el algoritmo genético debe seleccionar a cuales de ellos se les permitirá la reproducción. En este respecto un primer enfoque podría ser la elección de los individuos mejor calificados incrementando con ello la probabilidad de tener individuos buenos en la población y hacer que el algoritmo converja. Esto no es siempre cierto, ya que puede ocurrir que la población se acumule alrededor de algún individuo que sea bueno comparativamente con el resto, pero que esté muy lejos del óptimo gobal; lo anterior ocasiona convergencia prematura.

No se puede garantizar pero sí procurar que lo anterior no ocurra, asegurando que el AG tome un poco de todo, principalmente al inicio y en las últimas generaciones disminuya la cantidad de individuos seleccionados que no son tan buenos. En el capítulo 1 se describieron algunas estrategias de selección. La implementada en este trabajo es la conocida como *selección por torneo probabilístico*, el cual permite la selección de individuos con características diversas variando únicamente el valor de p_s, que es la probabilidad de seleccionar una cadena genética de acuerdo a su evaluación. El algoritmo 4 describe brevemente los pasos para seleccionar los padres que formarán nuevos descendientes. En el algoritmo, P es el tamaño de la población, $flip(p_s)$ es una función que devuelve 1 ó 0 con una cierta probabilidad p_s y T es el tamaño del torneo probabilístico.

Algoritmo 4 Operador de selección

for $h = 0$ to T **do**
  Realizar una permutación a la población
  for $i = 0$ to P **do**
    $cromosomaMayor \Leftarrow 0$
    $cromosomaMenor \Leftarrow \infty$
    for $j = 0$ to T **do**
      if $aptitudCromosoma_i \rangle cromosomaMayor$ **then**
        $cromosomaMayor \Leftarrow aptitudCromosoma_i$
        Almacenar índice i del $cromosomaMayor$
      end if
      if $aptitudCromosoma_i \langle cromosomaMenor$ **then**
        $cromosomaMenor \Leftarrow aptitudCromosoma_i$
        Almacena índice i del $cromosomaMenor$
      end if
      $i \Leftarrow i + 1$ /*Se incrementa el número de individuos seleccionados*/
    end for
    if $flip(p_s)$ **then**
      Almacena el índice del $cromosomaMayor$
    else
      Almacena el índice del $cromosomaMenor$
    end if
  end for
end for

Descripción. El Algoritmo 4 entra a un ciclo que se ejecutará T veces, el primer paso es hacer una permutación a la población, después, se inicia un segundo ciclo en el cual se tomarán T individuos de la población los cuales competirán entre sí obteniendo al mejor y al peor de ellos (tercer ciclo). La elección del mejor o peor individuo en la población se hará con una probabilidad p_s. Al término del segundo ciclo se habrán seleccionado P/T individuos y al terminar el primer ciclo se habrán elegido P individuos.

Análisis. En el algoritmo se observan tres estructuras importantes para el análisis, la primera de ellas, el ciclo **for** más interno que se ejecuta T veces. La segunda, el ciclo **for** que contiene al ciclo más interno, se ejecuta P/T veces. Finalmente la tercer estructura, el ciclo más externo y que contiene a las otras dos, se ejecuta T veces. Debido a que los tres ciclos están anidados, eśtos se ejecutan $(T * (P/T) * T)$ veces, es decir, $(T * P)$ veces. Por lo tanto, el orden de complejidad de este algoritmo es $O(T * P)$.

3.2.4 Operadores

Al definir los operadores que se utilizan en el AG será preciso tomar en cuenta el tipo de representación usada y evitar así la generación de cadenas no válidas.

Operador de cruza

Es posible diseñar operadores genéticos de cruza y mutación que siempre generen cadenas válidas. Ejemplo de ello son los operadores de cruza para permutaciones descritos en el capítulo 1. En este trabajo se usó el operador de cruza llamado "Order Based Crossover", el cual es una variante de la cruza uniforme para permutaciones. Conocido como operador **OX2**, selecciona al azar varias posiciones en la cadena de uno de los padres, para a continuación imponer en el otro padre, el orden de los elementos en las posiciones seleccionadas. Por ejemplo considere los padres

$$[1\ 2\ 3\ \mathbf{4}\ \mathbf{5}\ \mathbf{6}\ 7\ 8]$$

$$[1\ \mathbf{4}\ \mathbf{6}\ 8\ 7\ \mathbf{5}\ 3\ 2]$$

Suponga que en el segundo padre se seleccionan las posiciones segunda, tercera y sexta. Los elementos en dichas posiciones son 4, 6 y 5 respectivamente. En el primer padre dichos elementos se encuentran en las posiciones cuarta, quinta y sexta. El descendiente coincidirá con el primer padre si exceptuamos las posiciones cuarta, quinta y sexta:

$$[1\ 2\ 3\ X\ X\ X\ 7\ 8]$$

.

A continuación se rellenan los huecos del descendiente teniendo en cuenta el orden con el que aparecen en el segundo padre. Como resultado se obtiene

$$[1\ 2\ 3\ 4\ 6\ 5\ 7\ 8]$$

.

Cambiando el papel entre el primer y segundo padre, y utilizando las mismas posiciones seleccionadas, se obtiene el segundo descendiente:

$$[1\ 4\ 3\ 8\ 7\ 5\ 6\ 2]$$

El algoritmo 5 describe brevemente los pasos para la reproducción de los padres formando con ello la nueva población de descendientes. En el algoritmo, p_c es la probabilidad con la que se aplicará el operador de cruza a una pareja de individuos, k es el total de posiciones que serán generadas de forma aleatoria para la combinación de genes y n es el número de ciudades.

Análisis. El bloque principal es una estructura **if**. En ella se encuentran las siguientes subestructuras: Un ciclo **for** que se ejecuta k veces, k se genera de forma aleatoria y se encuentra en el rango $[1, n]$, por lo que el peor caso será cuando $k = n$. Después se encuentran dos ciclos **for** anidados, el ciclo interno se ejecuta k veces, es decir, en el peor caso se ejecutará

Algoritmo 5 Operador de recombinación

if $flip(p_c)$ **then**
Obtener aleatoriamente el número de posiciones k que serán generadas, tal que, $0 \leq k \leq n$
$k \Leftarrow (rand() + NUMEROPRIMO)\ MOD\ n$
Ahora se generan dichas posiciones
$nuevaPosicion \Leftarrow k$
for $i = 0$ to k **do**
$nuevaPosicion \Leftarrow nuevaPosicion + NUMEROPRIMO\ MOD\ n$
$bufferPosiciones_i \Leftarrow nuevaPosicion$
end for
Ver si el *i-ésimo* elemento de la cadena o cromosoma del primer padre es igual a alguno de los k elementos del segundo padre seleccionados y cuyas posiciones se encuentran en el **buffer de posiciones**. Esto se realiza con el siguiente ciclo.
for $i = 0$ to n **do**
for $j = 0$ to k **do**
if *alelo* que esta en la posicion $bufferPosiciones_j$ del primer padre $=$ *alelo* que está en la posición i del segundo padre **then**
Se coloca una marca X en la posición i en el $nuevoHijo$
else
Se copió el alelo i del segundo padre en el $nuevoHijo$
end if
end for
end for
Se copian los *alelos* del primer padre, cuyas posiciones están en $bufferPosiciones$ en el $nuevoHijo$
for $i = j = 0$ to n **do**
if *alelo* en la posición i del $nuevoHijo = X$ **then**
Posición i del $nuevoHijo \Leftarrow$ *alelo* del padre uno cuya posición esta en $bufferPosiciones_j$
end if
end for
else
Los dos padres pasan a la siguiente generación sin recombinarse
end if

n veces y el ciclo externo se ejecuta también n veces, por tanto, en el peor caso estos dos ciclos serán ejecutados $n * n$ veces. Finalmente, se tiene otro ciclo **for** que se ejecutará n veces. Este algoritmo se aplica a los individuos de la población con una probabilidad p_c, así, en el peor caso será aplicado a todos los individuos de la población. Los individuos se toman en parejas, por lo tanto todo el proceso se ejecutaría $P/2$ veces en el peor caso. Entonces, la función de tiempo en el peor caso del algoritmo 5 es aproximadamente: $(2n + n^2) * P/2$, y el orden de complejidad es $O(n^2 * P/2)$.

Elección de padres

La elección de los padres a los cuales será aplicado el operador de cruza se hace de la siguiente forma: se toman por parejas y como aparecen dentro de la población de individuos seleccionados, esto se hace hasta terminar con todos los individuos. Cada pareja de individuos se cruza con una probabilidad p_c. En caso de que los individuos no se recombinen pasarán directamente a la siguiente generación.

Operador de mutación

El objetivo de un operador de mutación es generar nuevos individuos, que exploren regiones del dominio del problema que probablemente no se han visitado aún. Así se buscan aleatoriamente nuevas soluciones que quizá superen las encontradas hasta el momento. Con lo que respecta a nuestro problema, no se usa el operador de mutación tradicional ya que puede generar cadenas inválidas. Se usa un operador de mutación que garantiza la obtención de cadenas válidas y se describió en el capítulo 1 (su nombre es *operador de mutación por desplazamiento*). El algoritmo se describe a continuación.

1. Generar de forma aleatoria k valores distintos, donde $k \leq n$ siendo n el número de instancias en el problema. Cada elemento d_i generado cumplirá que $1 \leq d_i \leq n$.
2. Generar de forma aleatoria k posiciones dentro de la cadena genética.
3. Buscar cada elemento generado en (1) dentro de la cadena cromosómica e insertarlo en la siguiente posición generada en (2).

Considere el ejemplo siguiente: Sea P tal que

$$P = 1\ 7\ 6\ 3\ 5\ 4\ 2$$

1. Se supone que se han generado los siguientes valores:

   $$\{7\ 6\ 2\}$$

   Se generaron 3 valores, entonces $k = 3$. Los valores obtenidos se resaltan en la cadena genética:

   $$P = 1\ \mathbf{7}\ \mathbf{6}\ 3\ 5\ 4\ \mathbf{2}$$

2. Ahora se supone que se generan las siguientes posiciones:

   $$\{6\ 4\ 2\}$$

3. Los movimientos a realizar son los siguientes:

   (a) Valor 7 se inserta en la posición 6. Los elementos se corren hacia la izquierda.

   $$[1 \leftarrow 6\ 3\ 5\ \mathbf{7}\ 4\ 2]$$

(b) Valor 6 se inserta en la posición 4. Los elementos se corren hacia la izquierda.

$$[1 \leftarrow 3\ \mathbf{6}\ 5\ 7\ 4\ 2]$$

(c) Valor 2 se inserta en la posición 2. Los elementos se corren hacia la derecha.

$$[1\ \mathbf{2}\ 3\ 6\ 5\ 7\ 4\ \rightarrow]$$

4. Se insertan los valores generados en (1) en las posiciones generadas en (2). La cadena genética queda de la siguiente forma:

$$P' = 1\ 2\ 3\ 6\ 5\ 7\ 4$$

Como se puede observar la nueva cadena generada es un cromosoma válido. El algoritmo 6 describe brevemente el operador de mutación implementado. En el algoritmo 6, p_m es la probabilidad con la que será aplicada la mutación a un individuo, k es el número de posiciones y cromosomas que serán generados de forma aleatoria para aplicar el proceso de mutación y n es el número de ciudades considerados en el problema.

Algoritmo 6 Operador de mutación

if $flip(p_m)$ **then**
  Obtener aleatoriamente el número de posiciones que serán generadas, tal que, $0 \leq k \leq n/2$
  $k \Leftarrow (rand() + NUMEROPRIMO)\ MOD\ n/2$
  Generar k índices en la segunda mitad del *nuevoHijo*, los alelos en dichas posiciones serán insertados. Almacenar los índices en un buffer de posiciones
  for $i = 0$ to k **do**
    $nuevaPosicion \Leftarrow (nuevaPosicion + NUMEROPRIMO\ MOD\ n/2) + (n/2)$
    $indicesDeValoresAInsertar_i \Leftarrow nuevaPosicion$
  end for
  Generar aleatoriamente las posiciones donde serán insertados los alelos cuyas posiciones estan en el buffer *indicesDeValoresAInsertar*. Almacenar estos índices en el buffer *indicesDondeSeInsertaran*
  Formar *hijoMutado* insertando en el los alelos cuyas posiciones están en *indicesValoresAInsertar*
  El nuevo hijo mutado formará parte de la siguiente generación
else
  El nuevo hijo pasa a la siguiente generación sin sufrir mutación
end if

Análisis. En el algoritmo 6 hay un ciclo for que se ejecuta k veces, donde k está en el rango $[1, n/2]$ por lo que en el peor caso $k = n/2$. Este proceso será aplicado a cada uno de los individuos de la población con una probabilidad p_m, por esto en el peor caso la mutación será aplicada a todos los individuos, ejecutándose el algoritmo P veces. La función de tiempo de este algoritmo es aproximadamente: $n/2 * P$ en el peor caso y el orden de complejidad es $O(n/2 * P)$.

Algoritmo	Orden de complejidad
Población inicial	$O(n * P)$
Función de evaluación	$O(n * P)$
Función de selección	$T * P$
Operador de recombinación	$O(n^2 * P/2)$
Operador de mutación	$O(n/2 * P)$

Tabla 3.2: Orden de complejidad en el peor caso de los algoritmos genéticos

3.2.5 Fase de reemplazamiento y condición de paro

Fase de reemplazamiento. Después de aplicar los operadores de selección, cruza y mutación a las cadenas de la población de padres y obtener con ello la nueva población de individuos (hijos), es necesario determinar la forma en la que será reemplazada esta población de padres por la nueva población de descendientes. Para esta implementación se optó por usar un esquema generacional, por ser una implementación sencilla y a pesar que la población actual es sustituida por la nueva de descendientes es también aplicado un mecanismo comúnmente llamado *elitismo*. Este mecanismo permite localizar al mejor individuo de una generación y mantenerlo presente en el siguiente ciclo, permitiendo con ello que el algoritmo genético encuentre individuos mejor adaptados en sucesivas generaciones.

Condición de paro. Finalmente, los pasos descritos antes se repiten evaluando en cada ciclo a la nueva generación obtenida terminando las evaluaciones después de un número g de generaciones o ciclos. Al final de estas generaciones se espera obtener una población que se encuentre muy cerca del óptimo global. De esta población se elige al individuo más adaptado el cual será la mejor solución encontrada por el AG.

3.3 Análisis del AG implementado

Terminada la definición del algoritmo genético para el TSP se procederá a calcular su orden de complejidad partiendo del análisis efectuado a cada uno de los procesos descritos en las secciones anteriores.

Los resultados de dichos análisis se muestran en la Tabla 3.2 donde se observan las principales funciones que componen al AG, cada una de ellas tiene un orden de complejidad diferente y con ellas se calculará el orden de complejidad total. Los cinco procesos se encuentran dentro de un ciclo que se ejecuta g veces, donde g es el número de generaciones o ciclos en los cuales serán aplicados los operadores genéticos. Por lo tanto la función de tiempo de todo el AG es aproximadamente:

$$f(t) = g * (n * P + n * P + T * P + n^2 * P/2 + n/2 * P)$$

Tomando el término de mayor grado, el orden de complejidad del AG es:

$$O(\frac{g * n^2 * P}{2})$$

El valor del tamaño de la población y del número de generaciones se puede definir en terminos del número de ciudades, de forma que:

$$P = n$$

y

$$g = \frac{n}{2} * n$$

Por lo que el orden de complejidad del AG es:

$$O(\frac{n^5}{4})$$

Es decir, el polinomio que describe el grado de complejidad del AG es de grado 5. El grado del polinomio puede incrementarse si se incrementa el número de generaciones o el tamaño de la población.

Capítulo 4

Modelos paralelos para el AG serial

Posiblemente una de las propiedades más importantes de los AGs es el paralelismo implícito lo que promueve a realizar implementaciones explícitamente paralelas aprovechando las propiedades que da la propia estructura del AG.
En las siguientes secciones se describirán algunos de los procesos desarrollados para la implementación paralela del AG convencional visto en el capítulo 3.

4.1 Necesidad de la paralelización de los AG

Los algoritmos genéticos secuenciales son muy lentos al tratar problemas relativamente grandes, es decir, tardan mucho para encontrar soluciones cercanas al óptimo global. Generalmete la ejecución de los operadores genéticos así como de la función de evaluación consumen mucho tiempo de cómputo, llegando a ser algoritmos sumamente lentos. Otro aspecto a considerar es el tamaño de la población que necesita crecer en número de individuos si se requiere de cierta diversidad en la población, consumiendo con ello gran cantidad de memoria. Con el objeto de resolver estas deficiencias y también con el fin de estudiar nuevos modelos, se han estudiado esquemas para diseñar *algoritmos genéticos paralelos*. Se espera que un Algoritmo Genético Paralelo (PGA) sea más eficiente, tanto en velocidad como en la calidad de resultados encontrados, debido a que su trabajo de búsqueda es diferente y también porque intervienen varios procesadores.

4.2 Análisis de rendimiento

Se realizó un análisis de los tiempos requeridos por los procesos que conforman al AG convencional con el fin de detectar los procesos que consumen mayor tiempo de cómputo. Las funciones del algoritmo serial son las siguientes:

- **crearPoblacion().** Se encarga de crear la primera población del algoritmo genético.
- **evaluación().** Se encarga de obtener el valor de aptitud de los individuos en la población.

- **aplicaElitismo().** Se encarga de obtener al individuo mejor adaptado en la población y de conservarlo para la siguiente.
- **selección().** Se encarga de seleccionar a los individuos que serán candidatos a reproducirse.
- **mutación().** Aplica mutación a un individuo con una probabilidad p_m.
- **cruza().** Recombina dos individuos con una probabilidad p_c.
- **main().** Es la función principal del AG serial.
- **Otras.** Son el resto de las funciones que componen al AG.

Se realizaron diversas ejecuciones variando los parámetros del algoritmo genético, pero sólo se eligió una de las más representativas tomado el conjunto de valores que se muestra en la Tabla 4.1.

Parámetros	**Ejecución 1**
Número de ciudades	30,40 y 50
Número de generaciones	$\#Ciudades * 30$
Tamaño de la población	$\#Ciudades$

Tabla 4.1: Parámetros dados al AG para las pruebas del análisis de rendimiento.

La Figura 4.1 muestra los resultados de las pruebas realizadas usando los parámetros de la Tabla 4.1. Se muestran porcentajes del tiempo de ejecución requeridos por las funciones principales del agoritmo genético serial.

En la gráfica 4.1 se tienen tres series diferentes, que se describen a continuación:

1. Serie izquierda (barras con relleno a cuadros). Muestra la ejecución del algoritmo genético con: *30 ciudades, 900 generaciones y un tamaño de población igual a 60 individuos.*
2. Serie central (barras con relleno de líneas verticales). Muestra la ejecución del algoritmo genético con: *40 ciudades, 1200 generaciones y un tamaño de población igual a 80 individuos.*
3. Serie derecha (barras con relleno de lineas diagonales). Muestra la ejecución del algoritmo genético con: *50 ciudades, 1500 generaciones y un tamaño de población igual a 100 individuos.*

Como se ve en estas pruebas, la función que requiere el mayor porcentaje de tiempo de ejecución del AG es el operador de cruza, después el operador de mutación y finalmente la función de evaluación, éste puede ser un comportamiento especial de los problemas de optimización combinatoria en donde la función de evaluación no es computacionalmente más costosa que la aplicación de los operadores de mutación y cruza. El requerimiento de estas

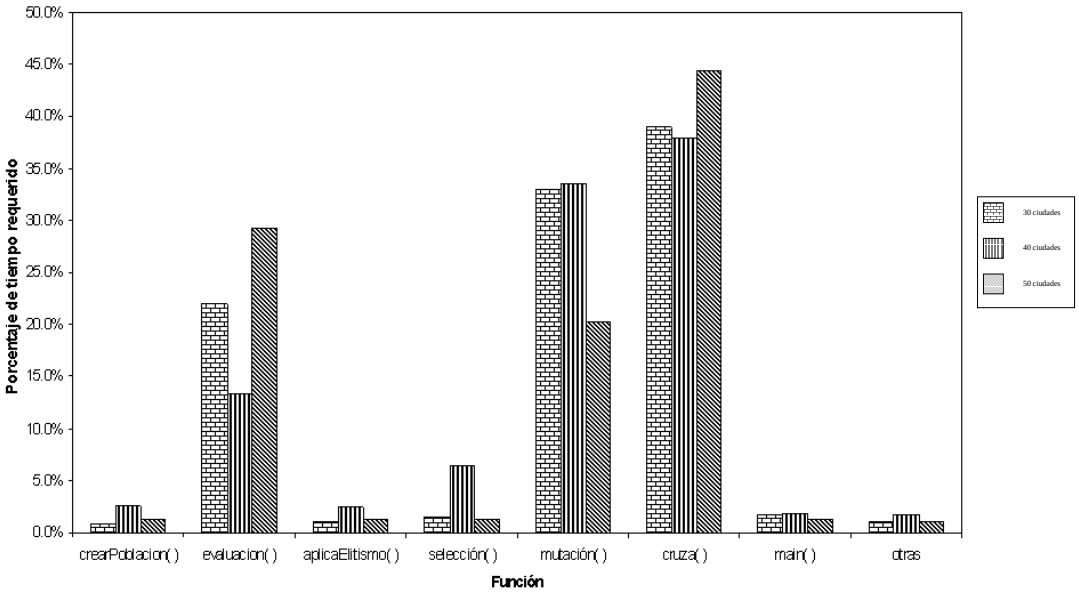

Figura 4.1: Tiempos requeridos por las funciones principales del AG convencional.

tres funciones es claramente mayor en comparación con el resto de las funciones, por lo que son claros candidatos a ser paralelizados.
En las siguientes secciones se describe la manera en la que fueron implementados de forma paralela los procesos que requieren altos recursos de memoria y/o procesador según la gráfica 4.1.

4.3 Paralelización del AG serial

Cuando se hacen versiones paralelas de algoritmos secuenciales es preciso tener presente que cuantos más procesadores se usen el costo de las comunicaciones crece, pero también decrecen los costos de cálculo que cada procesador tiene que realizar. No es conveniente sin embargo, agregar procesadores de forma indefinida pensando que el tiempo requerido para obtener la solución al problema tratado va en proporción con el número de elementos de procesamiento. En algún momento cuando estamos agregando procesadores, el costo de comunicación supera el costo de cálculo que debe realizar cada elemento, haciendo poco efectivo el paralelismo.

4.3.1 Esquemas de paralelización implementados

Son tres los esquemas de paralelización usados para los AGs secuenciales que fueron descritos en el capítulo 2. Cada modelo tiene características propias, las cuales van ligadas a la manera en que son aplicados los operadores genéticos sobre la población de individuos.

1. Primero se tiene al *modelo de paralelización global*, este aplica de forma paralela algunos procesos genéticos (los que consumen más tiempo de cómputo) y otros los ejecuta de

forma serial. Su comportamiento es muy similar al que tiene el AG serial, sin embargo la ganancia se encuentra en el tiempo de cómputo requerido.

2. La segunda implementación paralela se conoce como *modelo de paralelización de grano grueso*, la cual observó un comportamiento muy diferente con respecto del AG serial debido a que cada elemento de procesamiento ejecuta de forma independiente un AG sobre su conjunto de datos y esporádicamente intercambian información a tráves del operador de migración. Esto último lleva implícito un proceso de comunicación entre procesadores, siendo importante por o tanto determinar de qué forma se conectan los procesadores (topología).

3. Finalmente se implementó el *modelo de paralelización de grano fino*, en el que a diferencia de los modelos anteriores, la población en cada procesador se distribuye en una malla bidimensional en cual se forman subgrupos de individuos llamados vecindarios. Los vecindarios varian en la forma y número de individuos y determinan los límites en los cuales son aplicados los operadores genéticos.

En el desarrollo de los tres modelos paralelos para el AG secuencial se detectaron aspectos generales, los cuales son similares en las tres implementaciones efectuadas. Cada uno de ellos se describirán a continuación.

Tipo de descomposición de datos

El modelo conocido como *descomposición de datos* ha sido usado en este trabajo. Se realizó una división de la población de modo que cada procesador de la red realiza las mismas operaciones pero relacionadas con su conjunto de datos por lo que las cargas computacionales son prácticamente iguales. La figura 4.2 muestra un ejemplo del tipo de distribución de datos usado.

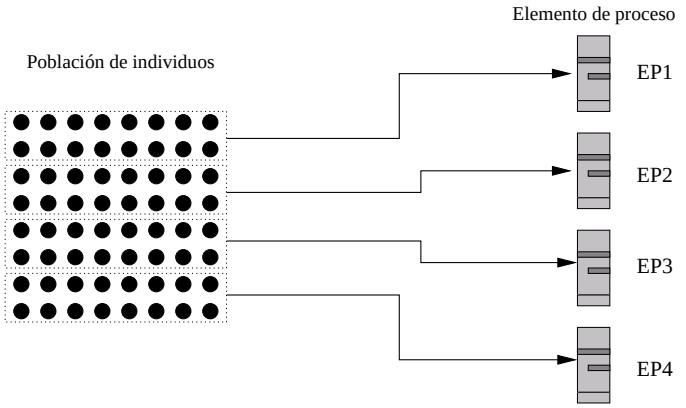

Figura 4.2: Estrategia de distribución de datos para los elementos de proceso.

A cada elemento de proceso de la figura 4.2 le corresponde una parte de la subpoblación de individuos. La distribución es muy simple y se asignan a cada procesador N/N_p, donde N es el número de cromosomas en la población y N_p es el número de procesadores.

Codificación de las cadenas genéticas y población inicial

La codificación de las cadenas es la misma que en el algoritmo serial. También la población inicial de individuos se obtiene de forma semejante, para los esquemas global y de grano fino el procesador central será el encargado de obtener los N individuos de la población y es el mismo proceso central el que hará una distribución de los cromosomas a los demás procesos e inclusive a él mismo, es decir, para N individuos en la población a cada procesador le corresponde N/N_p, donde N_p es el número total de procesadores incluidos en el cálculo paralelo. En el esquema de grano grueso cada procesador crea su propia población inicial. Sin embargo, a cada elemento de procesamiento le corresponde la misma cantidad de indiviuos que en los modelos global y de grano fino.
La Figura 4.3 muestra la manera en la que se distribuyen los datos por el procesador maestro. El algoritmo 2 del capítulo 3 muestra la manera en la que se genera la primera población de individuos.

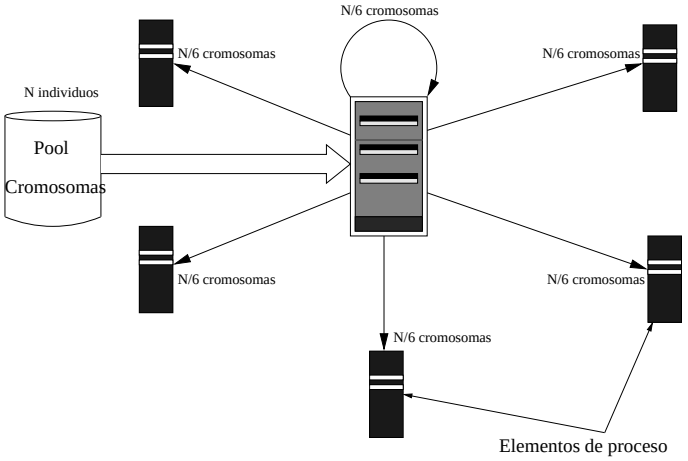

Figura 4.3: El proceso maestro distribuye los datos a los demás incluido él mismo.

Función de evaluación

Cada procesador evaluará los individuos que pertenecen a la sección de la población que tiene asignada. No hay cambios sustanciales con respecto al algoritmo convencional, esto debido a que sólo fue necesario considerar que la población de cada procesador tiene un tamaño máximo de N/N_p individuos.

Criterio de reemplazamiento y de paro

Para el caso de los esquemas global y de grano fino, es el proceso maestro el encargado de llevar a cabo el proceso de reemplazamiento y de verificación de la condición de paro. En el esquema de grano grueso cada elemento de procesamiento se encargará de realizar su propio proceso de reemplazo y de determinar si se ha cumplido o no la condición de paro. La estrategia usada para la sustitución de la población en cada ciclo genético y la finalización

del AGP es la misma que en el modelo secuencial.

A continuación se describirán los esquemas paralelos implementados y procesos que son particulares a cada uno de ellos.

4.4 Modelo de paralelización global

La primera forma de paralelismo implementada que se describirá es donde se utilizó el mismo algoritmo genético serial, pero se dividió a la población entre los N_p procesadores que intervendrán en el proceso paralelo y cada elemento de proceso aplica los algoritmos genéticos a su propio conjunto de datos. La comunicación entre procesadores sólo se da al finalizar cada ciclo genético y es para aplicar el algoritmo de selección y la técnica elitista.
Como en el AG secuencial, cada individuo compite con todos los restantes miembros de la población y también tiene una oportunidad de asociarse con cualquier otro individuo. El programa espera la recepción de las evaluaciones de toda la población antes de proceder con los siguientes procesos, por tanto es un algoritmo síncrono.

4.4.1 Topología y comunicación entre procesadores

Para la implementación de este tipo de AGP's se hace uso de una topología de estrella en la cual se identifica un procesador central llamado generalmente proceso maestro y que mantiene el control de las comunicaciones en la red. Además del proceso maestro hay un conjunto de procesadores que reciben el nombre de procesos esclavos. La Figura 2.5 del capítulo 2 muestra este tipo de topología. Para que la comunicación se dé correctamente, es necesario que exista una sincronización que permita a los procesadores enviar y recibir datos en el momento correcto, por lo que se deben identificar los puntos del algoritmo genético donde es preciso colocar las instrucciones de envío y recepción de datos ya que es ahí donde los elementos de proceso se sincronizan.

4.4.2 Operadores genéticos

Los operadores genéticos se aplican de forma independiente por cada elemento de proceso a su conjunto de datos, de forma que el número de individuos a los cuales se les aplicarán los operadores es N/N_p. La Figura 4.4 da una idea más clara de esto.

4.4.3 Estructura de los modelos globales implementados

Se hicieron dos implementaciones paralelas con el modelo global que se diferencian por el número de procesos que son paralelizados. En la primera de ellas se paralelizaron tres de los procesos principales del algoritmo genético, que son: *función de aptitud, operador de recombinación y el operador de mutación*. El esquema de la Figura 4.5 a) muestra de manera más clara el diseño del primer código paralelo para el modelo global.

A la izquierda del primer modelo se encuentra el pool de la población inicial. El proceso maestro se encarga de distribuirla entre los procesadores antes de entrar al ciclo genético

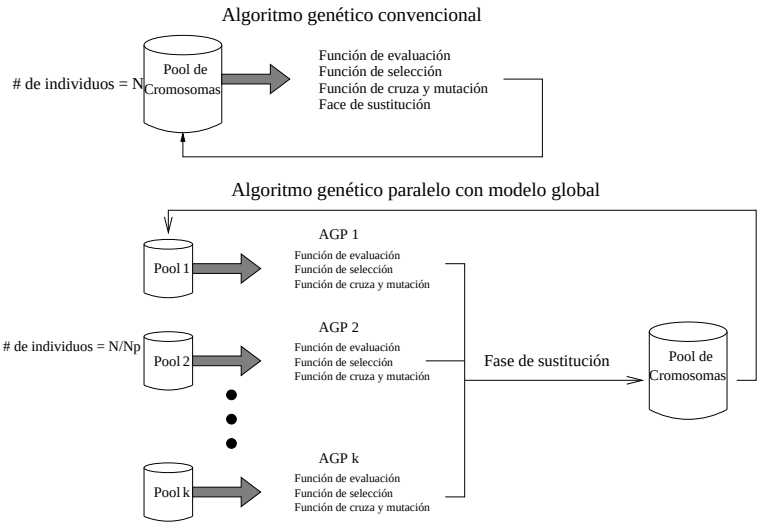

Figura 4.4: Proceso de ejecución del AGP global.

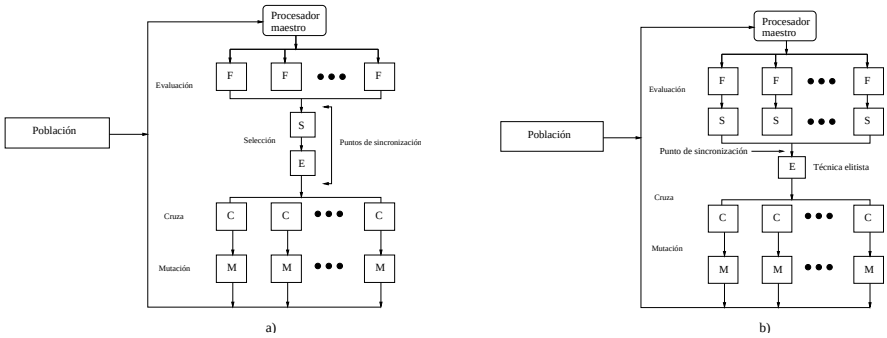

Figura 4.5: Esquemas implementados para el modelo global.

principal. Cuando se inicia dicho ciclo cada procesador calcula el valor de aptitud de los individuos de la subpoblación que le fue asignada y cada elemento de proceso envía la parte que le corresponde de los valores de aptitud calculados al elemento maestro. Lo anterior se hace para ejecutar de forma secuencial dos módulos: la *selección* de padres y también la aplicación de la técnica *elitista* (este es el primer punto de sincronización). El segundo punto de sincronización se da al término de estos dos procesos secuenciales, porque nuevamente se distribuye la población, aunque ahora son los individuos que fueron seleccionados por el operador de selección. Cada elemento de proceso recibe su parte y aplica a ella los dos operadores restantes, el operador de *cruza* y el operador de *mutación*; cada procesador obtiene una nueva subpoblación finalizando el ciclo genético e iniciando uno nuevo. El pseudocódigo se omite debido a que es muy similar al algoritmo 7 del segundo modelo global implementado y que se describe a continuación.

Segunda implementación. En la segunda implementación se paralelizaron cuatro de los procesos principales del algoritmo genético, que son: *función de aptitud, operador de se-*

lección, operador de recombinación y el operador de mutación. El único cambio que hay con respecto a la paralelización anterior es que fue paralelizado el operador de selección, que según el análisis de rendimiento, es uno de los procesos que requieran menos tiempo de cómputo pero, con el objetivo de ver su comportamiento se paralelizó. Se obtuvo un mejor rendimiento en esta implementación que en la efectuada antes, la cual paraleliza sólo tres procesos, los resultados se muestran en el capítulo 5. Esta mejora se observa más claramente cuando el tamaño de la población se incrementa notablemente y es debido en parte a que es requerido sólo un punto de comunicación.

El esquema de la Figura 4.5 b) muestra de manera más clara el diseño del segundo código paralelo para el modelo global. A la izquierda está el *pool* de datos. El proceso maestro la distribuye entre los procesadores antes de entrar al ciclo genético principal. Cada procesador calcula el valor de aptitud de los individuos de la subpoblación que le fue asignada. Después cada elemento de proceso envía al proceso maestro la parte que le corresponde de los valores de aptitud calculados al elemento maestro, lo anterior para ejecutar de forma secuencial un módulo: la aplicación de la técnica *elitista* (punto de sincronización). En este caso no se requiere distribuir nuevamente la población, debido a que cada procesador tiene la población que le corresponde. Por lo tanto, cada elemento de proceso aplica a su conjunto de individuos los operadores, obteniendo una nueva subpoblación de descendientes reiniciando con ellos un nuevo ciclo genético.
El algoritmo 7 muestra la función principal implementada del algoritmo genético paralelo con modelo global que paraleliza cuatro operadores genéticos.

4.5 Modelo de paralelización de grano grueso

El segundo enfoque de paralelismo implementado, es el que se conoce como algoritmo genético de grano grueso que consiste en crear de forma paralela múltiples subpoblaciones, las cuales evolucionan en diferentes procesadores con un algoritmo genético serial. Una diferencia fundamental con el AG con modelo global es que se comunican entre sí intercambiando individuos después de un número preestablecido de ciclos *(operador de migración).*

4.6 Operador de migración

La migración es un operador que se aplica al AG paralelo de grano grueso con el fin de mejorar la diversidad en cada uno de los demes[1] en la red. La migración se refiere específicamente a los intercambios de individuos que realizará cada elemento de proceso a lo largo de los ciclos evolutivos. Para la migración se consideraron los siguientes parámetros:

- Frecuencia de intercambio. Es el intervalo de generaciones que espera el AGP para realizar una migración
- Tasa de migración. Es el número de individuos que se migrarán.

[1] Una deme es una subpoblación que evoluciona en un procesador

Algoritmo 7 Función principal del segundo AGP con modelo global.

```
creaPrimeraPoblacion()
tamanoSubpoblacion ⇐ tamanoPoblacionTotal/numeroDeProcesadores
Iniciar biblioteca de paso de mensajes
/* Maestro envia matriz de distancias a todos los procesadores*/
broadCast(matrizDeDistancias, numeroDeCiudades*numeroDeCiudades,MAESTRO)
/* Proceso maestro divide la población y la distribuye a todos los procesadores */
scatter(poblacionADistribuir, tamanoSubpoblacion, subPoblacionDistribuida,MAESTRO)
while numGTmp do
  numGTmp ⇐ numGTmp − 1
  /* Obtener la aptitud de los individuos en la población*/
  funcionDeEvaluacion()
  /* Establecemos una barrera de sincronización */
  barrier()
  /* Los procesos envían sus evaluaciones y cromosomas al proceso maestro */
  gather(aptitudDeSubPoblacion, tamanoSubPoblacion, aptitudDePoblacion,MAESTRO)
  gather(subPoblacionDistribuida, tamanoSubPoblacion, poblacionTotal,MAESTRO)
  if myid = 0 then
    /* Aplica técnica elitista */
    FuncionAplicarElitismo()
  end if
  if numGTmp then
    /* Cada procesador aplica el operador de selección a su población*/
    funcionDeSeleccion()
    /* Cada procesador aplica operador de cruza y mutación a su conjunto de elementos*/
    funcionDeRecombinacion()
    funcionDeMutacion()
    /* Final de la generación, inicia nuevamente con la nueva población*/
  end if
end while
Finaliza la biblioteca de paso de mensajes
```

- Radio de migración. Está relacionado con la topología usada y se refiere a la distancia máxima que deberá recorrer un mensaje para llegar a su destino.
- Probabilidad de migración. Es la probabilidad de que un individuo determinado sea migrado.

Para determinar la **frecuencia de intercambio** se realizaron diez pruebas. En cada una de ellas se asignó un valor en el rango [0,100] que crecía a intervalos de 20. El valor seleccionado para la frecuencia de intercambio es 20. Para la **tasa de migración** se hicieron pruebas variando el valor del porcentaje de intercambios en el rango de [0,100] a intervalos de 10, el valor que fue seleccionado es 30. Finalmente se definió el **radio de migración**, el cual se determina por la topología usada. La **probabilidad de migración** es la probabilidad con la que un individuo es seleccionado para ser enviado a otro deme y en este trabajo es igual

a 0.8, es decir, los individuos con mejor peso o aptitud tienen más probabilidades de migrar, aunque también los individuos malos pueden ser seleccionados. Cada uno de los algoritmos de migración que fueron implementados para los diferentes tipos de red usados se describen en la siguiente sección.

Para la selección de los individuos a migrar se usa el *método de torneos probabilístico* el cual selecciona el 30% del total de la población. Se eligió este método de selección debido a que selecciona un poco de todo, tanto individuos buenos como individuos malos, esto último determinado directamente por la probabilidad de migración. Lo anterior proporciona cierta variedad a la subpoblación de individuos que serán migrados, evitando el envío de sólo cromosomas buenos o malos.

Se usan dos tipos de estructuras poblacionales:

1. En el primer modelo, conocido como modelo de islas, la población se particiona en subpoblaciones por aislamiento geográfico y los individuos pueden migrar a cualquier otra población (esto puede depender de un parámetro).

2. En el modelo conocido como *stepping stone*, la población se particiona en la misma forma, pero la migración de individuos se restringe a vecindarios definidos para las subpoblaciones. En estos dos enfoques de AGPs, los operadores de selección y cruce solamente ocurren dentro de cada subpoblación. El término deme usado en Biología puede usarse para referirse a una subpoblación. En las siguientes secciones se describirá la manera en la que se paralelizó el AG serial para el modelo de islas.

4.7 Topologías y comunicación entre procesadores

Para la implementación de este tipo de AGP's se hace uso de cuatro topologías que son: *red estrella, anillo e hipercubo* que se muestran en la Figura 2.5 del capítulo 2. Se hará una descripción de cada uno de estos modelos.

Topología de red

Existen para este esquema de red dos tipos: el primero llamado *red parcialmente conectada* tiene a cada elemento de proceso conectado a sus cuatro vecinos más cercanos en la red. Y el segundo tipo es llamado *red totalmente conectada* en el cual cada elemento de proceso se conecta con todos los demás vecinos. Esta última se usa en el presente trabajo, la Figura 2.5 del capítulo 2 muestra un ejemplo de esta red y el algoritmo 8 muestra los pasos seguidos para la implementación del operador de migración.

Algunas de las ventajas de esta topología son que cualquier nodo en el sistema está conectado a todos los otros nodos de la red, debido a esto los mensajes entre nodos pueden entregarse rápidamente ya que la comunicación entre cualquier par de nodos es directa. En este tipo de sistema deben fallar muchos nodos para que la red completa falle. La desventaja es que el costo básico es alto, ya que debe existir una conexión directa entre cada dos nodos. El costo de anexar un nuevo nodo crece según la cantidad de nodos que contiene la red.

Algoritmo 8 Proceso de migración para una topología de red.

$numIndividuosMigrar \Leftarrow nSubPoblacion * TASAMIGRACION$
seleccionarIndividuos a migrar con el método de los torneos y almacenarlos en el buffer $poblacionIndividuosAMigrar$
$individuosSeleccionadosParaLaMigracion \Leftarrow$ seleccionTorneoProbabilistico()
for $i = 0$ to $numprocs$ **do**
  if $myid = i$ **then**
    El procesador i envía sus individuos a todos sus vecinos.
    enviarATodos($aptitudSubPoblacionMigrar, i$)
    enviarATodos($poblacionIndividuosMigrar, i$)
  else
    Todos los procesadores excepto el procesador i reciben los individuos que importarán.
    todosRecibenDei($aptitudSubPoblacionImportada, i$)
    todosRecibenDei($poblacionIndividuosImportada, i$)
    Sustituir aleatoriamente los individuos importados en la población actual.
  end if
end for

Topología de estrella

En este tipo de red uno de los nodos se conecta a todos los demás como se muestra en la Figura 2.5. Para el AG paralelo no se mantiene un sólo nodo central durante todo el proceso del AG. Dicho nodo se determina en tiempo de ejecución y la elección estará en función de la aptitud promedio de la población de cada deme en la red lo cual se realizará de acuerdo a la frecuencia de intercambio: es decir, el deme cuya aptitud promedio sea la mejor de todas en la generación *k-ésima* será el deme central. Se usa el método del torneo en un hipercubo para determinar cual deme tiene el mejor promedio en la *k-ésima* generación. El algoritmo 9 nos muestra cómo se realiza este proceso.

Esta forma de elegir al procesador maestro da oportunidad a todos los nodos de la red de ser elegidos, con base en la aptitud de su población de manera que puedan agregar variedad a través del intercambio genético o proceso de migración.
El algoritmo 10 muestra la implementación del proceso de migración usando una topología de estrella.

Primero cada proceso determina la cantidad de individuos a migrar y se forma un paquete que contiene el promedio de su población y su identificación. Se aplica el método de torneos probabilísticos para seleccionar a los individuos que serán migrados y después inicia el torneo entre los demes de cada elemento de procesos para saber qué deme es el mejor de todos y saber quién será el proceso central. Pero al final de dicho torneo, sólo el procesador cero sabrá quién es el mejor deme, es decir, $indiceMejorProcesador$ en el algoritmo 10. Esta información la enviará a todos sus vecinos usando un *broadcast*. Una vez que saben qué proceso es el mejor cada proceso vecino le enviará sus individuos seleccionados para la migración, y el proceso central a su vez seleccionará aleatoriamente N_m/N_v individuos de cada deme, donde N_m es el *número de individuos de intercambio* y N_v es el *número de vecinos*. Con las N_v porciones seleccionadas de cada deme, el proceso central completará su tasa de migración.

Algoritmo 9 Torneo para un hipercubo aplicado para determinar el mejor deme.

for $i = 1, j = 0$ to dim **do**
  if $(numeroProcesadores > i) AND (myid\ MOD\ i = 0)$ **then**
    if $(myid\ MOD\ (i * 2)) <> 0$ **then**
      envia(aptitudPromedio y rank) al proceso $myid - i$
    else
      if $myid < numeroProcesadores - 1$ **then**
        recibe(aptitudPromedio y rank) de proceso $myid + i$
        if $mipromedio < promediorecibido$ **then**
          Forma un mensaje con el mejor promedio y la jerarquía del mejor y colócalo en el siguiente mensaje a enviar
        end if
      end if
    end if
  end if
  $i \Leftarrow i * 2$
  $j \Leftarrow j + 1$
end for
donde dim es la dimensión del cubo.

Algoritmo 10 Proceso de migración para una topología de estrella.

$numIndividuosMigrar \Leftarrow nSubPoblacion * TASAMIGRACION$
$mensajeEnvioTH \rightarrow rank \Leftarrow myid$
$mensajeEnvioTH \rightarrow envio \Leftarrow promedioPoblacion$
seleccionarIndividuos a migrar con el método de los torneos y almacenarlos en el buffer $poblacionIndividuosAMigrar$
$individuosSeleccionadosParaLaMigracion \Leftarrow$ seleccionTorneoProbabilistico()
Iniciar el torneo en un hipercubo para elegir al procesador central
if $myid$=0 **then**
  $indiceMejorProcesador \Leftarrow mensajeEnvioTH \rightarrow rank$
end if
proceso cero envía a todos sus vecinos $indiceMejorProcesador$
broadCast($indiceMejorProcesador$,MASTER)
Todos envían sus individuos a migrar al procesador $indiceMejorProcesador$
gather($aptitudIndividuosMigrar\ y\ cadenasIndividuosMigrar, indiceMejorProcesador$)
Proceso central selecciona aleatoriamente los individuos a importar

Algunas ventajas que se pueden observar en esta topología son que el costo es lineal al aumentar el número de nodos. El costo de comunicación entre dos nodos es bajo, ya que un mensaje entre dos nodos por ejemplo AB (que no son el proceso central C) requiere de dos transferencias (desde A a C y de C a B). Si no hay muchos mensajes la velocidad de transferencia se mantendrá alta. Su desventaja es que el nodo central puede convertirse en un cuello de botella y si éste llegara a fallar la red queda totalmente particionada.

Topología de anillo

En este tipo de red cada nodo está conectado sólo a otros dos nodos, como se muestra en la Figura 2.5. El anillo usado es unidireccional debido a que cada elemento de proceso sólo se comunica con el siguiente y el último nodo de la red se comunica con el primero. El Algoritmo 11 muestra la implementación del proceso de migración usando una topología de anillo.

Algoritmo 11 Proceso de migración para una topología de anillo.

$numIndividuosMigrar \Leftarrow nSubPoblacion * TASAMIGRACION$
seleccionarIndividuos a migrar con el método de los torneos y almacenarlos en el buffer $poblacionIndividuosAMigrar$
$individuosSeleccionadosParaLaMigracion \Leftarrow$ seleccionTorneoProbabilistico()
if $myid$=0 **then**
  Recibe las cadenas y las aptitudes de los individuos a importar del procesador $numeroProcesadores - 1$
  recibe($cadenasyAptitudesIndividuosImportar, numprocs - 1$)
end if
if $myid = numeroProcesadores - 1$ **then**
  Enviar al procesador 0 las aptitudes y cadenas de los individuos seleccionados para la migración.
  envia($cadenasyAptitudesIndividuosMigrar$,0)
end if
if $myid <>0$ **then**
  Recibe las cadenas y las aptitudes de los individuos a importar del procesador $myid - 1$
  recibe($cadenasyAptitudesIndividuosImportar, myid - 1$)
end if
if $myid <> numprocs - 1$ **then**
  Enviar al procesador $myid + 1$ aptitudes y cadenas de los individuos seleccionados para la migración.
  enviar($cadenasyAptitudesIndividuosMigrar, myid + 1$)
end if
Sustituir aleatoriamente los individuos importados en la población actual.

En el algoritmo, cada proceso determina su tasa de migración. Después se aplica el método de torneos probabilísticos para determinar que individuos se exportarán. Cada proceso enviará sus individuos al procesador siguiente, excepto el último procesador, el cual enviará su información al procesador 0. De igual forma, cada proceso recibirá los individuos que le han sido enviados del procesador anterior, excepto el primero quien recibirá individuos del último procesador (*numeroProcesadores-1*). Finalmente cada proceso insertará aleatoriamente en su población a los individuos recibidos.

Algunas ventajas de esta topología son las siguientes: no existe un cuello de botella como en la topología de estrella y cada nodo se conecta a sólo otros dos nodos. Como el anillo es unidireccional un nodo trasmite directamente su información hacia uno de sus vecinos.

El costo básico también es lineal con respecto de la cantidad de nodos. Sus desventajas son que el costo de comunicación puede ser muy alto ya que un mensaje de un nodo a otro debe viajar alrededor del anillo hasta que llegue a su destino, es decir, deberá recorrer a lo sumo $n-1$ nodos. Si un nodo falla, la red se particionará.

Topología de hipercubo

Un cubo tridimensional se ilustra en la Figura 4.6. Si se numeran los nodos del hipercubo usando una numeración binaria acorde con la numeración del mismo, los nodos vecinos difieren en un solo bit (aquél en cuya dimensión la posición de los nodos cambia). Cada nodo se comunicará directamente con otros tres nodos, uno en la línea vertical, otro en la línea diagonal y otro en la línea horizontal. El concepto de cubo puede extenderse a un espacio de dimensión n obteniéndose el $n-cubo$ con n bits para cada vértice. Una red $n-cubo$:

- Se corresponde con N nodos donde $n = log_2\ N$.
- Cada nodo está conectado exactamente con n vecinos. Estos vecinos difieren exactamente en un bit.
- Un cubo $n-dimensional$ que conecta 2^n nodos con $n >= 3$ se denomina hipercubo.

Para un cubo $n-dimensional$ con 2^n nodos, la máxima distancia de comunicación entre dos nodos es $n = log_2\ N$. Cada procesador está directamente conectado con:

$$destino = myid\hat{}(1 << i) \tag{4.1}$$

donde ^ es el operador *or* bit a bit, $<<$ representa el corrimiento a la izquierda e i toma valores en el rango $[0, dim-1]$, donde dim es la dimensión del cubo.

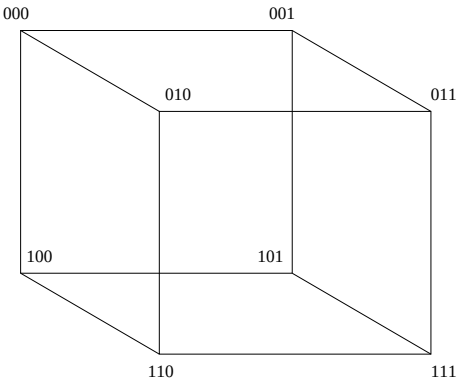

Figura 4.6: N-cubo de grado 3.

El Algoritmo 12 muestra los pasos seguidos para la implementación del operador de migración en una topología de hipercubo.

En dicho algoritmo tenemos que, dim es la dimensión del $n-cubo$ y $destino$ es el procesador al cual se enviarán los individuos a migrar y del cual se obtendrán los individuos

Algoritmo 12 Proceso de migración para una topología de anillo.

$numIndividuosMigrar \Leftarrow nSubPoblacion * TASAMIGRACION$
seleccionarIndividuos a migrar con el método de los torneos y almacenarlos en el buffer $poblacionIndividuosAMigrar$
$individuosSeleccionadosParaLaMigracion \Leftarrow$ seleccionTorneoProbabilistico()
for $i = 0$ to dim **do**
  $destino = myid\hat{}(1 << i)$
  recibir(*aptitud de vecinos de dimensión i,destino*)
  recibir(*poblacion de vecinos de dimensión i,destino*)
  enviar(*aptitud de población a vecinos de dimensión i,destino*)
  enviar(*población a vecinos de dimensión i,destino*)
end for
Sustituir aleatoriamente los individuos importados en la población actual.

a importar por el procesador *myid*. Cada proceso determina su tasa de migración. Posteriormente se aplica el método de torneos probabilísticos para saber cuales individuos se exportarán o migrarán a otro procesador. Cada elemento de proceso entra en un ciclo que va de 0 hasta la dimensión del cubo menos uno. En este ciclo cada procesador enviará su información genética a migrar a *dim* vecinos.

Algunas ventajas de la topología son: en un cubo $n - dimensional$ con 2^n nodos, cada nodo se conecta directamente sólo con n nodos y la máxima distancia de comunicación entre dos nodos es $n = log_2 N$. Debido a las múltiples conexiones entre los nodos, las fallas en una rama no provocan la partición de la red.

4.7.1 Paralelización de los módulos genéticos

En este esquema de paralelización los modulos genéticos no sufren grandes cambios; por ello se describirán a continuación los de mayor relevancia.

La población inicial la obtiene cada elemento de proceso de forma independiente y de la misma forma se calculará su valor de aptitud.

Los operadores genéticos como: *operador de selección, operador de recombinación, operador mutación y la aplicación de la técnica elitista* no sufrieron muchos cambios con respecto al algoritmo genético serial.

4.8 Modelo de paralelización de grano fino

En este esquema, el tamaño de las subpoblaciones es más pequeño aunque el número de comunicación entre los procesadores es mucho mayor que en los otros esquemas (*grano grueso y modelo global*). Cada procesador se relaciona con ciertos procesadores, formando un vecindario el cual delimita la población donde se aplican los operadores de cruce y selección. Los vecindarios pueden traslaparse, por lo que existe intercambio de información genética

constante entre ellos (ver Figura 2.8 del capítulo 2).

4.8.1 Estructura del modelo de grano fino

En un esquema de grano fino la comunicación entre precesadores es intensiva debido a la propia estructura del algoritmo. Idealmente cada elemento de proceso representa a un individuo, por lo que, todos los individuos de la malla estarán enviando y recibiendo mensajes a sus vecinos transportando su material genético. La estructura de un algoritmo genético celular se muestra en la Figura 4.7.

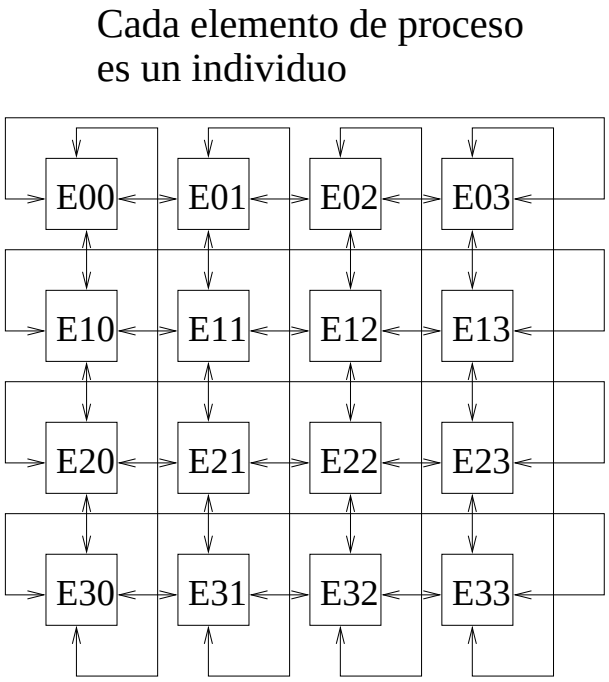

Figura 4.7: Estructura general de un AG de grano fino.

4.8.2 Simulación serial de un AG celular

Debido a que no se cuenta con una arquitectura masivamente paralela se optó primero por implementar una simulación serial de un AG celular en una computadora de un solo procesador y después se hizo la paralelización de dicha simulación. La simulación serial sigue el mismo comportamiento de la implementación paralela de grano fino. Se inicia con una población de N individuos los cuales se organizan en forma de malla bidimensional, quedando la población en una matriz de $m \times m$ donde:

$$N = m \times m$$

Para aplicar los operadores genéticos (*selección, cruza y mutación*) se hace el recorrido de la malla, procesando a cada individuo de manera secuencial y en el siguiente orden: *de*

izquierda a derecha y de arriba hacia abajo, llegando en algún momento a cada cromosoma en la malla. Cada individuo competirá con sus vecinos y con ellos podrá recombinarse para obtener descendencia. Los vecinos de cada cromosoma se determinan por el tipo de vecindario usado, en este trabajo por ejemplo se usan dos tipos de vecindario: **Vecindario lineal y Vecindario compacto.**

Ambos esquemas descritos en el capítulo 2. Otro factor importante es el radio del vecindario. La Figura 2.7 del capítulo 2 muestran ejemplos de estos dos tipos de vecindario. En el presente trabajo se realizaron pruebas usando diferentes radios de vecindario, pero los resultados mostraron que para vecindarios de radio igual a uno, se obtuvieron los mejores resultados por lo que se optó por elegir ese valor.

Módulos genéticos

La población inicial se obtiene de la misma forma que en el algoritmo secuencial pero en este caso los individuos se organizan en una matriz de tamaño $m \times m$, donde

$$m = \sqrt{N}$$

y cada individuo representará en caso ideal un nodo o procesador. La evaluación de los individuos en la población se calcula sólo una vez en cada ciclo genético y cada procesador trabaja con sus propios datos.

Para el proceso de selección se realiza lo siguiente: dentro de cada deme, los procesadores transmiten sus individuos más recientemente formados, así cada procesador obtiene copias de distintos individuos, de estas copias el programa selecciona un compañero. Para la selección del compañero se usa el método de torneos probabilísticos donde participan solamente los individuos del vecidario (ver Figura 4.8).

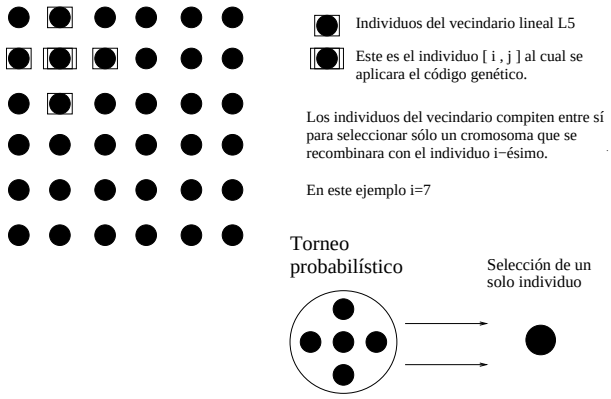

Figura 4.8: Proceso de selección en un vecindario L5.

El operador de recombinación se aplica a cada individuo $[i, j]$ de la malla y la combinación del material genético se efectuá con el cromosoma seleccionado en el proceso de selección descrito arriba. El operador de mutación se aplica con una probabilidad p_m a cada individuo.

Cada nuevo descendiente obtenido en los dos procesos anteriores ocupará una posición cercana, es decir, en el mismo vecindario que sus padres. La elección de la nueva posición para el cromosoma hijo se determina de manera aleatoria y una vez que se sabe qué punto de la malla ocupará, migrará a esa posición sustituyendo al cromosoma que tenga asignado el mismo punto.

El algoritmo 13 muestra la función principal de la simulación del algoritmo genético celular.

Algoritmo 13 Proceso principal de la simulación del AGc.

```
Entrada: nMalla-Tamao de la malla
crearLaPrimeraPoblacion();
while numGTmp do
  numGTmp − − /*Una generación menos*/
  Calcula la aptitud de los cromosomas: evaluacion()
  Aplica esquema elitista: elitismo()
  if numGTmp then
    for i = 0 to individuoActuali = 0; nMalla do
      for j = 0 to individuoActualj = 0; nMalla do
        Obtener los vecinos del individuo [i, j]: obtenVecindario()
        Ejecuta el proceso de selección con probabilidad p_s: seleccion(p_s)
        Ejecuta el proceso de cruza y mutación: cruzaYMutacion()
      end for
    end for
  end if
  Realiza el proceso de reemplazamiento: reemplazamientoGeneracional()
end while
```

En este esquema paralelo es necesario establecer un nuevo parámetro que determine el tipo de vecindario a usar.

4.8.3 Paralelización de la simulación serial del AG celular

Para implementar un algoritmo genético paralelo con esquema celular que realmente explote las caracteríticas del modelo de grano fino es necesario disponer de una arquitectura masivamente paralela, esto aunque posible es poco común. Por tanto, se buscan alternativas que permitan codificar de forma paralela el algoritmo de la sección anterior.
Los procesadores se ordenan de tal forma que los individuos de las supoblaciones quedan dispuestos en una malla bidimensional similar a la del modelo serial. La diferencia está en que los individuos del procesador *i-ésimo* tendrán a sus vecinos del NORTE o SUR en otro procesador. Lo anterior requirió mecanismos de comunicación para migrar a dichos vecinos al procesador que los requiera. Las solicitudes (comunicación entre elementos de proceso) de los vecinos se hace de la manera siguiente:

1. Proceso $myid <> numeroProcesadores-1$ solicita vecinos del SUR al proceso $myid+1$.

2. Proceso $myid <> 0$ solicita vecinos del NORTE al proceso $myid - 1$.

3. Si $myid = numeroDeprocesadores - 1$, solicita vecinos del SUR al procesador 0.

4. Si $myid = 0$ solicita vecinos del norte al procesador $numeroDeProcesadores - 1$.

La figura 4.9 muestra lo anterior de forma más clara.

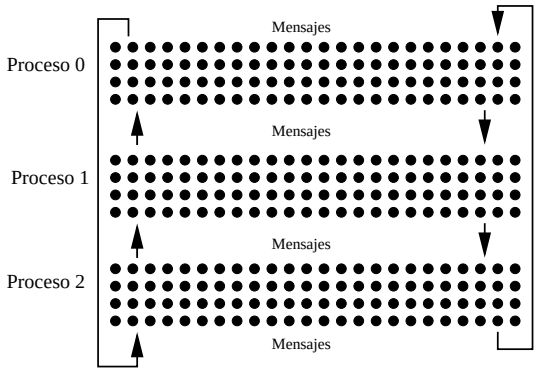

Figura 4.9: Intercambio de cromosomas en el modelo de grano fino paralelo.

Los individuos cuyos vecinos se encuentren en una malla diferente deberán solicitarlos efectuando un proceso de migración de individuos. Esto se hará antes de iniciar un ciclo genético, ver Figura 4.10. El Algoritmo 14 muestra la forma en la que hacen las comunicaciones para el intercambio de información.

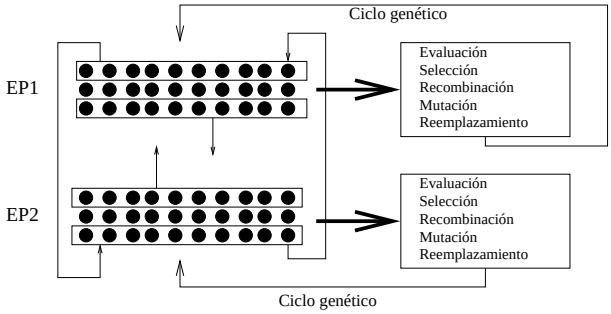

Figura 4.10: Antes de iniciar una generación los EP intercambian información.

Operadores genéticos

Después de que cada elemento de proceso recibe y envía información genética se ejecutan los operadores genéticos sobre los individuos de la población. Al igual que en los esquemas del modelo global y modelo de islas, los operadores sufrieron pequeños cambios en su estructura. De hecho sólo hubo que modificar algunos parámetros, los cuales se describen a continuación.

Algoritmo 14 Proceso de intercabio de vecinos entre los procesadores.

```
/*Vecinos del NORTE*/
if myid = 0 then
  recibe(vecinosDelNORTE del proceso nprocs − 1)
end if
if myid = nprocs − 1 then
  envia(vecinosDelNORTE al proceso 0)
end if
if myid! = 0 then
  recibe(vecinosDelNORTE del proceso myid − 1)
end if
if myid! = nprocs − 1 then
  envia(vecinosDelNORTE al proceso myid + 1)
end if
/*Vecinos del SUR*/
if myid = nprocs − 1 then
  recibe(vecinosDelSUR del proceso 0)
end if
if myid = 0 then
  envia(vecinosDelSUR al proceso nprocs − 1)
end if
if myid! = nprocs − 1 then
  recibe(vecinosDelSUR del proceso myid + 1)
end if
if myid! = 0 then
  envia(vecinosDelSUR al proceso myid − 1)
end if
```

Población inicial y evaluación

La creación de la población inicial la realiza el proceso 0 (en este caso proceso maestro), la organiza en una malla bidimensional y es él quien se encargará también de distribuirla a los demás elementos de proceso. Cada elemento de proceso recibirá la porción de población que le corresponde y calculará para cada elemento existente el valor de aptitud.

Selección

Cada elemento en la malla se procesará de forma individual, es decir, se inicia con el elemento [0,0], después el elemento [0,1], después [0,2],..., hasta terminar con el elemento $[N/N_p,N/N_p]$, donde N es el tamaño de la población total y N_p es el número de procesadores usados.

Para el elemento $[i,j]$ (donde $0 \leq i$ y $j \leq (N/N_p)$) del procesador E_k el proceso de selección es el mismo que en la simulación serial, pero en este caso se debe considerar que los vecinos de los individuos pertenecientes a la primera y última fila de la malla no se encuentran

en la población de dicho proceso E_k. Estos individuos se importaron de un proceso vecino y se almacenan en dos vectores uno de vecinos del NORTE y el otro de vecinos del SUR, por lo que bastará con referenciar a cada uno usando su posición en el vector de vecinos correspondiente.

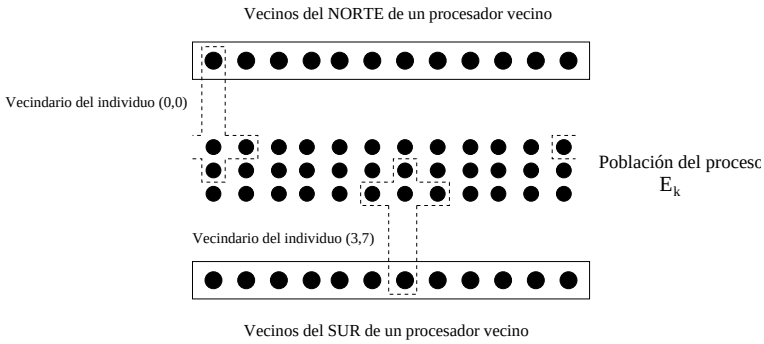

Figura 4.11: Formación de vecindarios para el proceso de selección.

Como se ve en la Figura únicamente los individuos de la primera y última fila necesitan incluir en su vecindario a los individuos que fueron importados de los procesos vecinos. Después de efectuar el torneo probabilístico es seleccionado un individuo del vecindario. El vecindario usado es un *L5*. Para un esquema con *C9* tendríamos algo similar.

Cruza. Después de saber qué individuo del vecindario fue seleccionado se procede a efectuar el módulo de cruza. Los individuos que se combinarán son: el individuo procesado actualmente (individuo $[i, j]$) y el individuo elegido en el proceso de selección. Si un par de individuos no se recombinan, significa que no generarán hijos, sin embargo debe hacerse una elección porque el mejor de ellos sí pasará a la siguiente generación.

Cada nuevo descendiente ocupará la posición del padre (posición $[i, j]$) en la población nueva, por lo que al término de una generación se habrán generado N/N_p individuos hijo.

Mutación. Después de efectuada la cruza sobre el individuo $[i, j]$ se aplica el proceso de mutación al nuevo descendiente con una probabilidad p_m.

Criterio de reemplazamiento y de paro

Durante cada ciclo genético se mantienen dos poblaciones:

1. **Población anterior** (individuos de la generación actual).
2. **Población nueva** (individuos hijos o descendientes).

Dado que en la población nueva se encuentran los descendientes es necesario aplicar un criterio de reemplazamiento sobre los individuos de la población anterior. Así, cada nuevo descendiente obtenido ocupará una posición cercana, es decir, en el mismo vecindario de los

padres. La posición se elige de forma aleatoria y una vez que se sabe en qué punto de la malla deberá ser colocado el individuo, éste migrará a su nueva posición.
Después del proceso de reemplazamiento (fin de una generación en el AG) cada procesador envía los individuos que recibió al iniciar el ciclo genético, ver Figura 4.12.

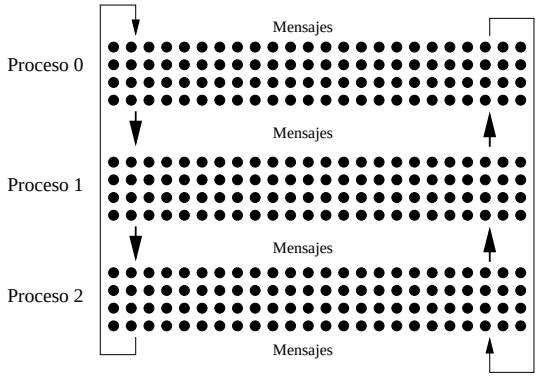

Figura 4.12: Los procesadores envían de regreso a los vecinos recibidos al inicio del ciclo.

El algoritmo es muy similar al Algoritmo 14. Finalmente el algoritmo genético se ejecuta un número fijo k de generaciones o ciclos.

En el siguiente capítulo se muestran y describen algunas de las pruebas realizadas a los AGs implementados.

Capítulo 5

Análisis y evaluación de resultados

En este capítulo se describirán los experimentos más representativos realizados para evaluar el rendimiento de los AGs seriales y paralelos implementados y descritos en los capítulos 3 y 4 del presente reporte. A continuación se describen todas las pruebas realizadas.

5.1 Pruebas realizadas

Para el análisis de rendimiento se describen dos tipos de pruebas: *independientes y generacionales*

Pruebas generacionales

Los resultados que se representan en este tipo de pruebas muestran el mejor valor encontrado por el AG al término de la generación *j-ésima* de una sola ejecución con k generaciones, donde $j \in [1, k]$ y donde k es el número total de generaciones que será ejecutado el AG. Dado que el proceso aplicado es de minimización, entonces, el comportamiento de la gráfica es monótono decreciente, debido al esquema elitista aplicado por los AGs.

Para cada gráfica de prueba generacional que se muestra se observará también una gráfica con el tiempo requerido por el AG para realizar dicha prueba generacional.

Pruebas independientes

Las gráficas que representan este tipo de pruebas muestran el mejor valor encontrado por el AG al término de una ejecución del AG con k generaciones. Para los resultados mostrados más adelante se representarán los mejores tours encontrados en diversas ejecuciones del AG en una sola gráfica, donde cada punto representará una prueba independiente con k generaciones del AG. Los resultados muestran que por ser técnicas heurísticas y estocásticas, los AGs no siempre obtienen un mejor resultado al aumentar el número de generaciones. Para las gráficas presentadas, $0 \leq k \leq 3000$, y se presentan valores de k en intervalos de 10.

Los resultados obtenidos con la realización de las pruebas independientes fueron los esperados. Con el propósito de presentar con mayor claridad los resultados las gráficas presentan

el **valor medio aritmético**, el cual permite observar de forma clara la tendencia general de cada AG mostrado. La segunda es el **mejor y peor tour encontrado en cada prueba**, que muestra el valor del mejor y del peor tour encontrado en la ejecución independiente i, donde $i \leq 300$ y también se indica qué AG obtuvo dicho valor. Finalmente se muestra una tabla con la cantidad de veces (en porcentajes) que un AG obtuvo el mejor tour en las 300 pruebas independientes.

5.2 Elección de los parámetros de entrada

En los capítulos 3 y 4 del presente reporte se describió cómo los 4 modelos genéticos implementados tienen parámetros en común y también parámetros específicos. La elección de los valores para dichos parámetros no es una tarea sencilla debido a que no hay un método que garantice que con determinados valores se obtendrá un buen desempeño del AG. En este trabajo la elección del conjunto de valores para los parámetros se realizó escogiendo aquel que permitió al AG convencional tener un mejor desempeño en cuanto a calidad de resultados. Una vez seleccionado el conjunto de parámetros, éste fue usado para todos los genéticos implementados.

5.3 Parámetros de entrada y pruebas realizadas

Para la elección del conjunto de valores se realizaron una gran cantidad de pruebas. En este reporte se describen únicamente aquellas que tuvieron como fin mostrar el desempeño del AG convencional ante tres tamaños de problema distintos y elegir aquel que represente de manera clara el desempeño de cada algoritmo. Los tres tamaños de instancia son: TSP con 20 ciudades, TSP con 40 ciudades y el TSP con 60 ciudades. Los datos para cada tamaño de problema como son: ciudades (en el plano (x, y)) y costos (de cada recorrido posible) fueron creados de forma aleatoria usando para ello un programa que genera instancias de tamaño n. Las pruebas serán mostradas en cuatro bloques. En el primero de ellos se observan algunas de las gráficas hechas al AG serial y que permitieron determinar uno de los parámetros que se eligieron para las pruebas definitivas. En el segundo se describirán los resultados obtenidos al hacer pruebas generacionales y pruebas independientes a los genéticos de cada modelo paralelo (haciendo una comparación de sus resultados). En el último bloque se hará una comparación final de los mejores resultados obtenidos en cada uno de los modelos paralelos, mostrando también algunas gráficas de aceleración.

5.3.1 Pruebas al algoritmo convencional

Los valores que puede tomar el conjunto de parámetros de un AG depende en gran medida del problema que se intenta resolver. Por esto, se realizaron diversas pruebas estableciendo en cada una un conjunto de valores diferentes para cada uno de los parámetros de entrada. Un ejemplo de ello se muestra a continuación.

Pruebas al algoritmo genético serial variando el tamaño de la población

Se establecen tres tamaños de población diferentes y se hacen para cada una 300 pruebas variando el número de generaciones en intervalos de 10. El tipo de pruebas mostradas son independientes y se observa la gráfica del valor medio aritmético, así como del mejor y peor tour encontrado por cada genético. Para cada ejecución el AG tiene un comportamiento distinto debido por una parte a que es un mecanismo estocástico de búsqueda y por otra, debido al tamaño de la población de individuos. La Tabla 5.1 muestra los tamaños de población usados para estas pruebas.

Población	Tamaño
1	#ciudades/5
2	#ciudades*2
3	#ciudades*(#ciudades/2)

Tabla 5.1: Tres tamaños de población para el AG convencional.

En la Figura 5.1 se muestra la media aritmética de las pruebas independientes.

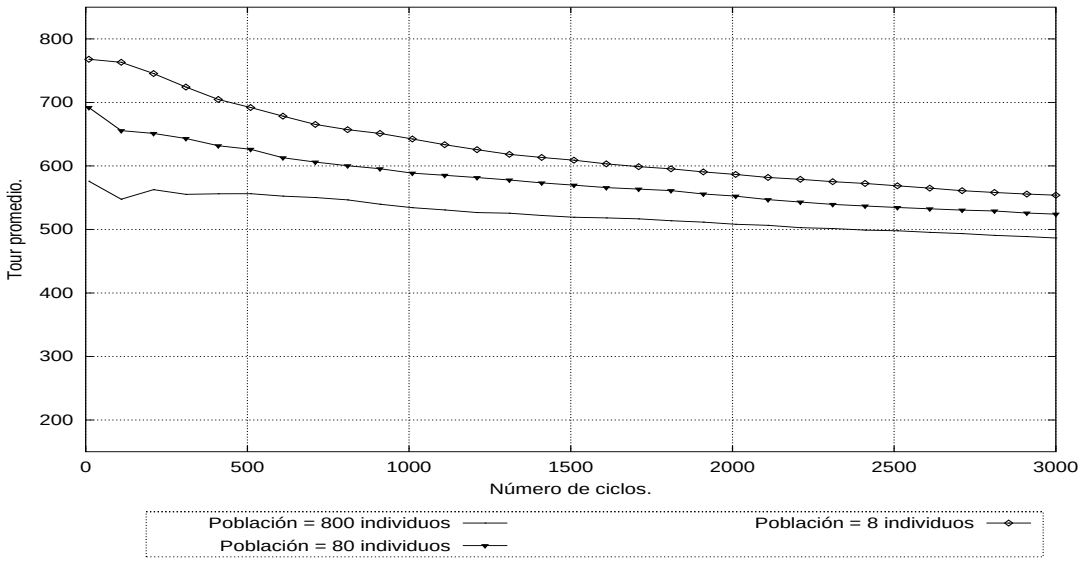

Figura 5.1: Prueba independiente. Comportamiento del AG variando el tamaño de la población.

Es claro que el AG que tiene el mayor tamaño de población muestra una mejor tendencia de los valores promedio. Esto puede deberse a que un tamaño de población mayor da posibilidad de una mayor diversidad de individuos en la población, aumentando con ello la probabilidad de que más de esos individuos posean valores de aptitud bajos.
La Figura 5.2 es una gráfica de los mejores y peores tours encontrados por los AGs y muestra cómo el AG con la mayor población obtuvo en todas las pruebas al mejor tour y el AG con la menor población obtuvo siempre el tour más bajo. La Tabla 5.2 muestra el porcentaje de

ocasiones en que cada genético obtuvo el mejor valor y se ve cómo el genético serial con el mayor tamaño de población obtuvo en el 100% de las pruebas al individuo mejor adaptado.

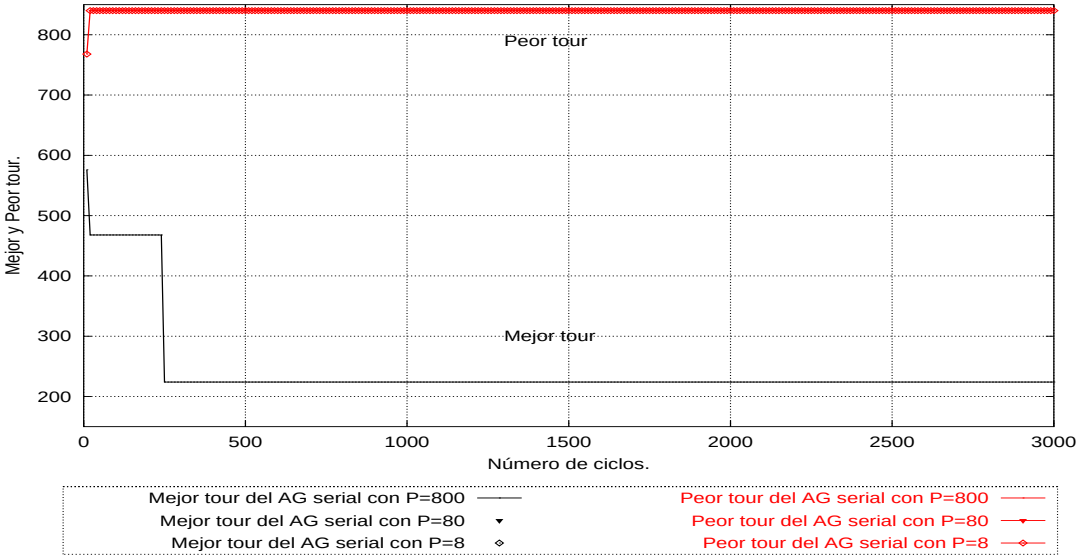

Figura 5.2: Mejores y peores tours encontrados en las pruebas independientes.

Algoritmo genético serial con:	**Porcentaje**
Población = 800 individuos	100%
Población = 80 individuos	0%
Población = 8 individuos	0%

Tabla 5.2: Porcentaje de veces en que cada AG encontró al mejor tour.

Los resultados mostrados son claros pero es necesario considerar que con un tamaño de población mayor el AG requiere mayor cantidad de recursos (memoria y tiempo de procesador) aumentando con esto el tiempo que tarda en cada ciclo genético para procesar una población y aunque también aumenta la probabilidad de encontrar individuos con aptitudes altas es posible que se necesite mucho tiempo antes de lograr encontrar un buen resultado. La Figura 5.3 muestra lo anterior de forma gráfica y se observa claramente el incremento del tiempo requerido al aumentar el tamaño de la población.

Se eligió un tamaño de problema de 40 ciudades que es suficientemente grande para mostrar las ventajas de usar técnicas paralelas en este tipo de heurísticas. La Tabla 5.3 muestra los valores elegidos para los parámetros de entrada del genético serial.

5.3.2 Pruebas a los modelos paralelos

Las pruebas realizadas se describen de forma separada y en el siguiente orden:

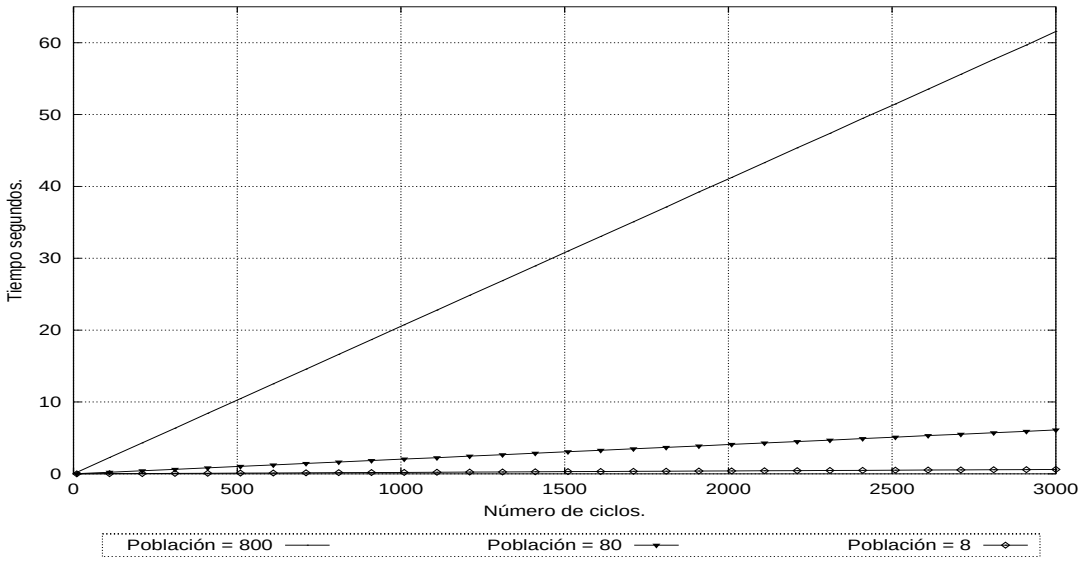

Figura 5.3: Tiempo requerido por el AG al variar el tamaño de la población

Parámetro	**Tamaño**
Número de ciudades	40
Número de generaciones	3000
Tamaño de la población	#ciudades*20
Probabilidad de selección	0.20
Probabilidad de cruza	0.95
Probabilidad de mutación	0.001

Tabla 5.3: Parámetros elegidos para el AG convencional.

1. Mejor tour encontrado en pruebas generacionales.

2. Mejor tour encontrado en pruebas independientes, representadas por la media aritmética.

3. Pruebas del mejor y peor tour encontrado.

4. Tabla de las veces que un AG encontró al mejor o peor individuo, representado en porcentajes

5. Tiempo requerido por los AG de las gráficas anteriores.

Al final se hace una comparación de los mejores resultados obtenidos para cada modelo paralelo implementado. Se inicia con el modelo global, después el de grano grueso y se finaliza con el modelo de grano fino.

Modelo global

Los parámetros propios del AG global y los valores elegidos para ellos se muestran en la Tabla 5.4, donde N es el tamaño de la población y N_p es el número de procesadores a usar.

Parámetro	**Valor**
#Procesadores (N_p)	4 y 8
Tamaño de cada deme	N/N_p

Tabla 5.4: Parámetros del AG con modelo global.

Hay dos implementaciones para este modelo, en el primero sólo 3 operadores genéticos se ejecutan de forma paralela: *evaluación, cruza y la mutación* y en el segundo son 4 operadores que se ejecutan de forma paralela: *evaluación, selección, cruza y mutación*; de aquí en adelante se nombrarán: *modelo global 1 y 2* respectivamente para hacer referencia a ellos. A continuación se describen los resultados de estas dos implementaciones globales, para pruebas generacionales e independientes usando 4 y 8 procesadores.

Pruebas usando 4 procesadores

La Figura 5.4 muestra el comportamiento de cada uno de los AGs de modelo global usando 4 procesadores. Los resultados muestran que al menos para esta prueba generacional el *modelo global 2* obtuvo el mejor resultado. Sin embargo, no se puede garantizar que este modelo sea siempre mejor debido a que sólo se trata de una prueba. Ambas ejecuciones paralelas mejoraron el desempeño del AG convencional.

En las pruebas independientes que se observan en la Figura 5.5 se podrá concluir algo definitivo sobre el comportamiento de los dos modelos globales. Se debe recordar que una gráfica de pruebas independientes está formada por diversas ejecuciones del AG variando el número de generaciones. Es decir, muestran un comportamiento más general.

En el *modelo global 1* el proceso maestro crea la población inicial y selecciona de ella los individuos que se reproducirán, es decir, cada individuo compite con todos los demás elementos de la población, de la misma forma que lo hace el genético convencional. La diferencia está en que los operadores de *cruza, mutación y evaluación* se ejecutan de manera paralela. El hecho de mantener durante todo el proceso evolutivo una sola población de la cual sean seleccionados individuos para recombinarse garantiza un comportamiento muy similar al que tiene el genético convencional, con excepción del tiempo requerido.
En el *modelo global 2* el operador de selección es ejecutado de forma paralela. La población inicial la crea el proceso maestro, sin embargo durante todo el proceso evolutivo cada procesador selecciona y crea sus propios individuos quedando aislados unos de otros. Esto forma islas que evolucionan de forma independiente, originando un comportamiento diferente al genético convencional. Estos dos puntos se observan en la Figura 5.5, el *modelo 1* muestra un comportamiento similar al genético serial y el *modelo 2* muestra una mejor tendencia.

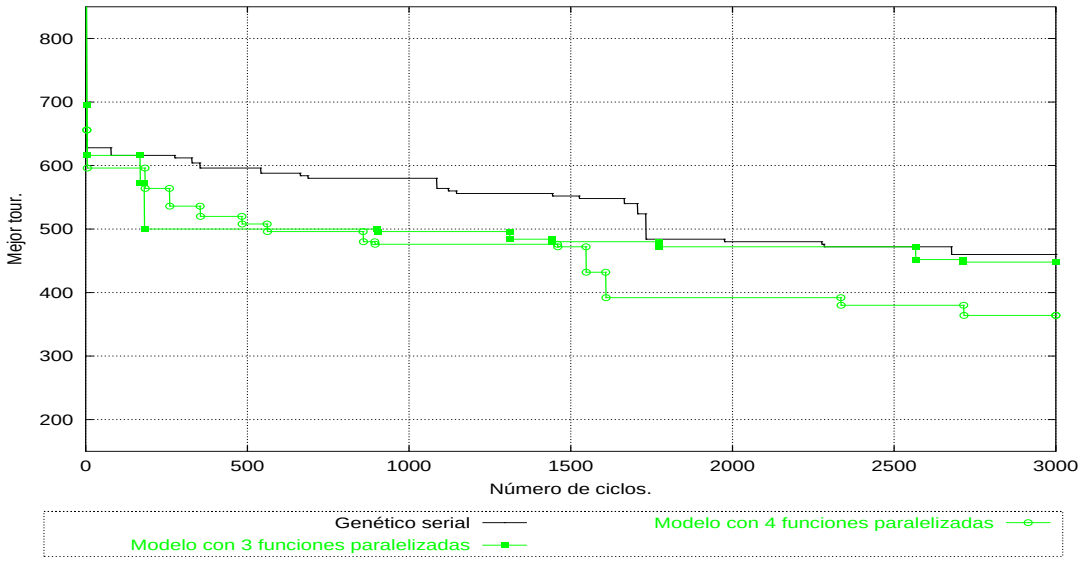

Figura 5.4: Prueba generacional. Resultados de los AGs con modelo global usando 4 procesadores.

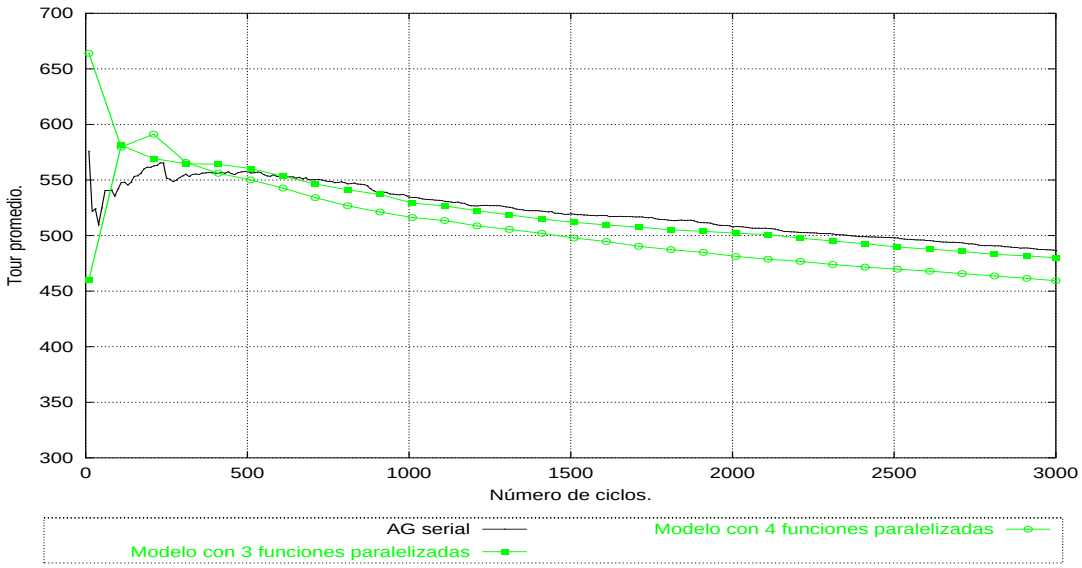

Figura 5.5: Media aritmética, pruebas independientes. Resultados de los AGs con modelo global usando 4 procesadores.

Los resultados en la Figura 5.5 no muestran los mejores y peores valores de aptitud obtenidos por cada AG en la ejecución independiente i, esto debido a que se está representando la media aritmética. Sin embargo, para tener una idea clara del desempeño de cada algoritmo genético, en la Figura 5.6 se presentan los peores y mejores valores obtenidos en el experimento anterior y en la Tabla 5.5 se muestra el porcentaje de ocasiones en que cada genético obtuvo el mejor valor de aptitud.

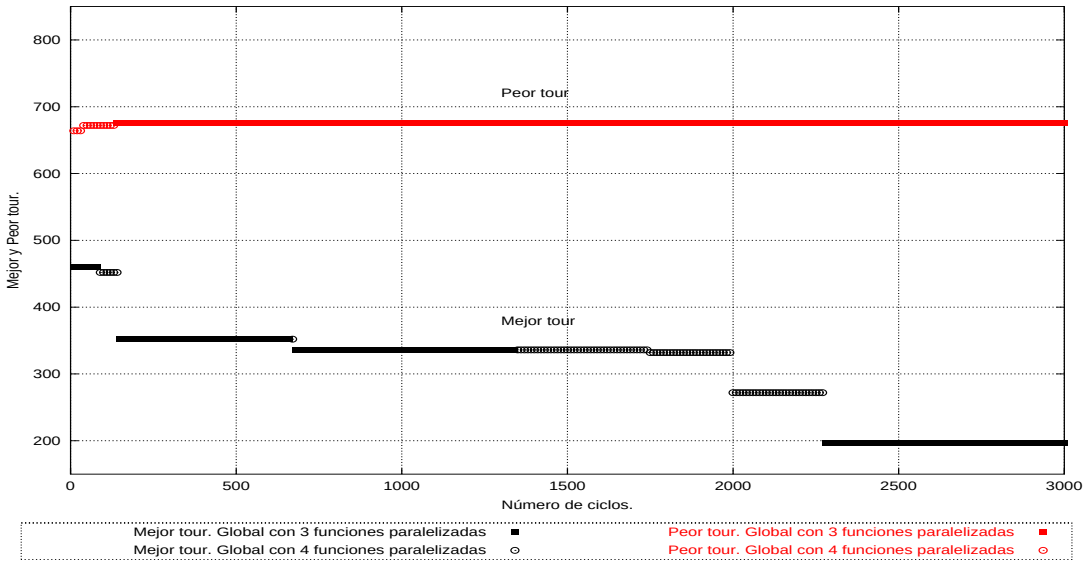

Figura 5.6: Mejor y peor tour obtenidos por los AGs con modelo global usando 4 procesadores.

De la Tabla 5.5 se observa que el *modelo global 1* mantuvo en más pruebas a los mejores valores de aptitud, aunque la mejor tendencia general la tiene el *modelo global 2* (Figura 5.5).

Algoritmo genético global (4 EP) con:	**Porcentaje**
modelo global 1	66.666664%
modelo global 2	33.3333332%

Tabla 5.5: Porcentaje de veces en que cada AG global con 4 procesadores encontró al mejor tour.

Finalmente, la Figura 5.7 muestra el tiempo requerido por cada implementación paralela. Se observa para ambas implementaciones paralelas una ganancia en tiempo con respecto de la implementación serial (Figura 5.3 con población = 800). El modelo global 2 requirió menos tiempo que el modelo global 1, debido a dos puntos:

1. El modelo *global 1* tiene dos puntos de sincronización y el *modelo 2* sólo uno. Lo anterior hace que el primer modelo requiera más tiempo de comunicación.

2. El *modelo global 2* ejecuta 4 operadores de forma paralela y el *modelo 1* sólo 3. Lo anterior hace que el *modelo 2* reduzca su tiempo de procesamiento debido a que existe un operador más (*operador de selección*) que se ejecuta de forma paralela.

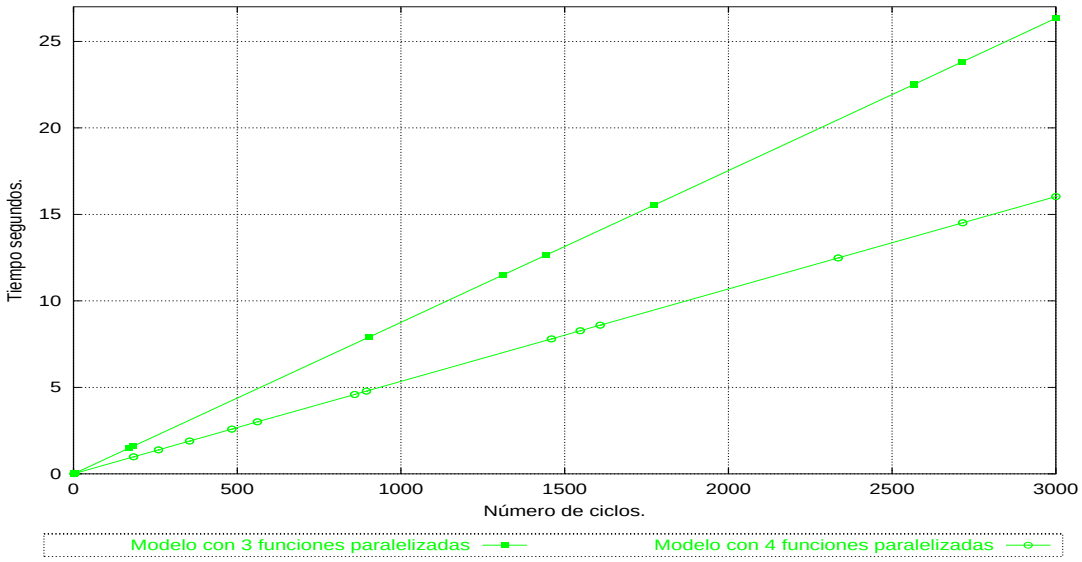

Figura 5.7: Tiempo requerido por los AGs con modelo global usando 4 procesadores.

Pruebas usando 8 procesadores

La Figura 5.8 muestra el comportamiento de cada uno de los AG de modelo global usando 8 procesadores. Nuevamente se observa que el *modelo global 2* obtuvo mejores resultados. El desempeño del *modelo 1* con 8 procesadores es mejor que el de 4 procesadores y el *modelo 2* tuvo un desempeño menor que al usar 4 procesadores. Se trata sólo de una prueba, así que será más conveniente analizar la gráfica de pruebas independientes (Figura 5.9).

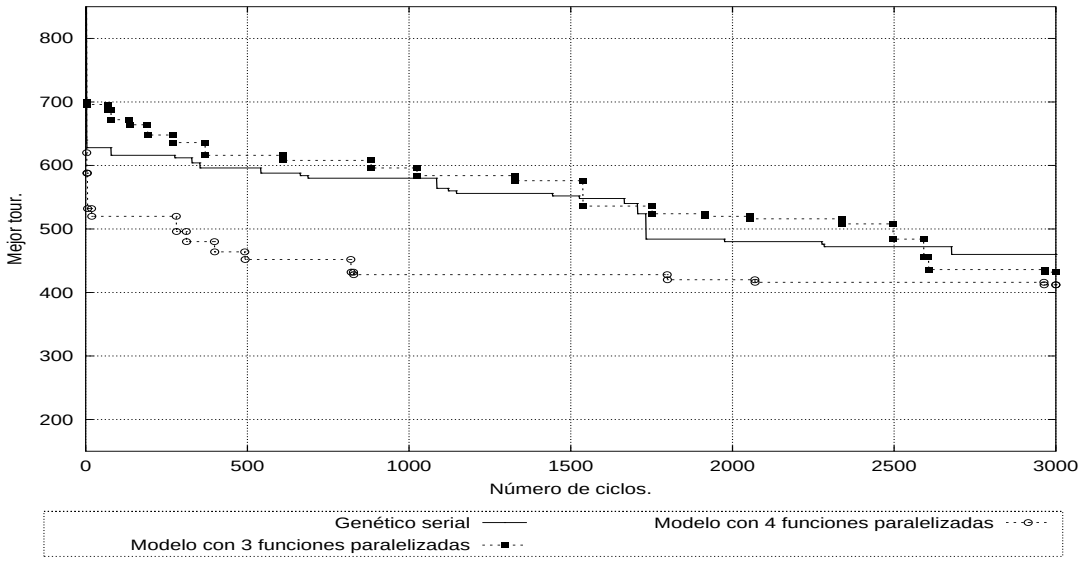

Figura 5.8: Pruebas generacionales. Resultados de los AGs con modelo global usando 8 procesadores.

En las pruebas independientes los resultados son muy similares a las ejecuciones de 4 procesadores. El *modelo global 1* muestra un comportamiento muy similar al AG convencional y el *modelo global 2* observa un mejor desempeño. El *modelo global 2* con 8 procesadores tiene una mejor tendencia que al usar 4 procesadores, debido posiblemente a que al aumentar el número de procesadores se incrementa el número de islas que evolucionan de manera independiente, incrementando así la variedad.

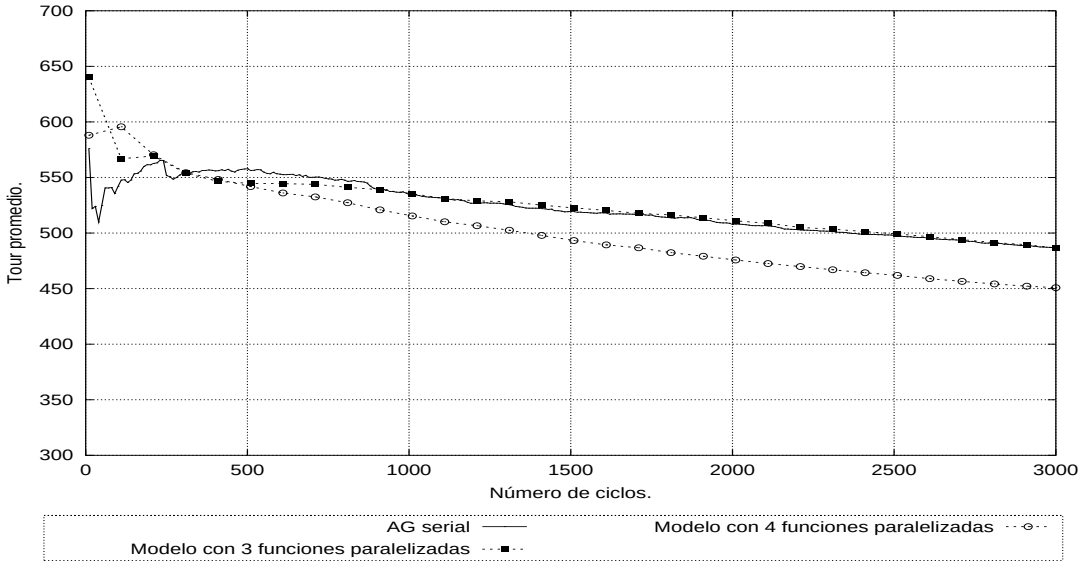

Figura 5.9: Media aritmética, pruebas independientes. Resultados de los AGs con modelo global usando 8 procesadores.

Los resultados en la Figura 5.9 no muestran los mejores y peores valores de aptitud obtenidos por cada AG en la ejecución independiente, lo cual se presenta en la Figura 5.10 y la Tabla 5.6 muestra el porcentaje de ocasiones en que cada genético obtuvo el mejor valor de aptitud.

De la Tabla 5.6 se observa que el *modelo global 2* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud y la mejor tendencia general como se observa en la Figura 5.9.

Algoritmo genético global (8 EP) con:	**Porcentaje**
modelo global 1	4.0%
modelo global 2	96.0%

Tabla 5.6: Porcentaje de veces en que cada AG global con 8 procesadores encontró al mejor tour.

Finalmente la Figura 5.11 muestra el tiempo requerido por cada implementación paralela. Observamos en ambas implementaciones paralelas con 8 procesadores una ganancia en tiempo

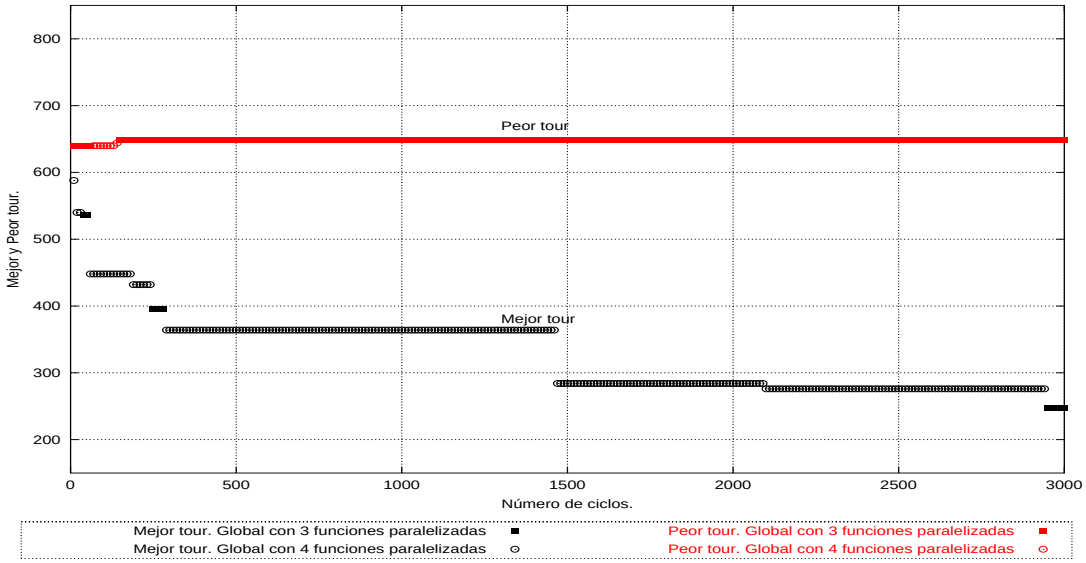

Figura 5.10: Mejor y peor tour obtenidos por los AGs con modelo global usando 8 procesadores.

comparado con la implementación serial y comparado con las implementaciones paralelas de 4 procesadores sólo se mejoró el tiempo para el *modelo global 2*. El *modelo global 1* requirió más tiempo de cómputo que cuando se usaron 4 procesadores, debido a que al aumentar el número de procesadores se incrementaron también las comunicaciones de sincronización llevando a una mayor demanda de tiempo. Por el contrario el *modelo global 2* mostró una ligera mejora, en este caso el tiempo para comunicación (un solo punto de sincronización) no supera la ganancia en procesamiento paralelo.

Modelo de grano grueso

En este modelo se consideran 4 parámetros más que se muestran con sus valores en la Tabla 5.7.

Parámetro	**Valor**
Probabilidad de migración	0.8
Tasa de migración	0.30
Frecuencia de migración	20 generaciones
Radio de migración	Depende de la topología

Tabla 5.7: Valores de los parámetros de un AG de islas.

El conjunto de valores se eligieron siguiendo el mismo procedimiento que en el algoritmo secuencial. En este modelo se implementaron 4 topologías diferentes que son: *topología de anillo, estrella, hipercubo y de red*. En las 4 topologías los parámetros son los mismos excepto el radio de migración que está basado en la posición lógica de los nodos de la red y la forma

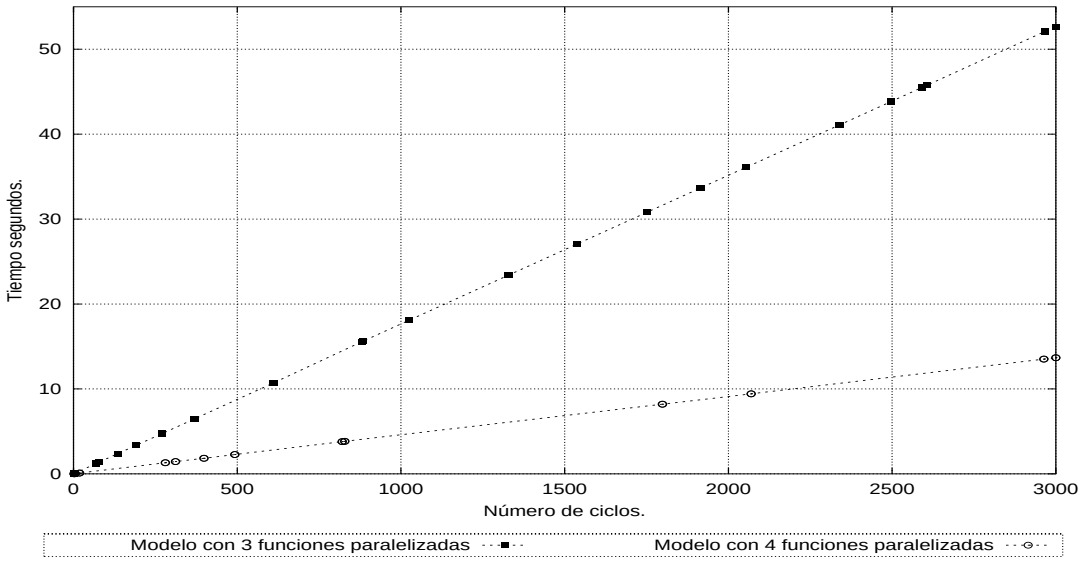

Figura 5.11: Tiempo requerido por los AGs con modelo global usando 8 procesadores.

en la que éstos se comunican entre sí por lo que dependerá de la topología. A continuación se describen las pruebas obtenidas usando 4 y 8 procesadores.

Pruebas usando 4 procesadores

La Figura 5.12 muestra los mejores resultados obtenidos por cada una de las topologías usando 4 procesadores. Se observan también los resultados obtenidos por el AG convencional.

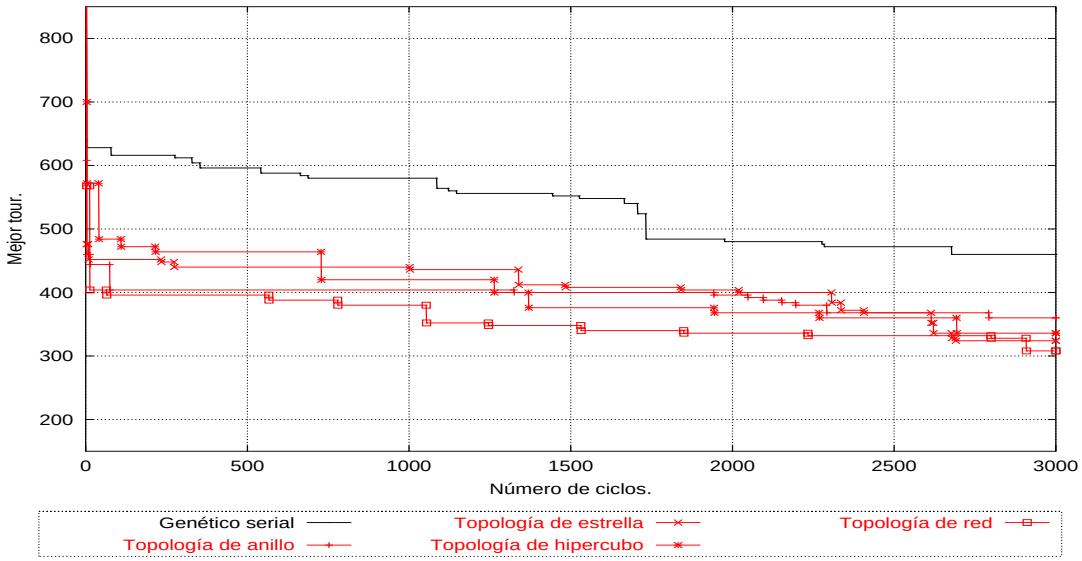

Figura 5.12: Prueba generacional. Resultados de los AGs de grano grueso usando 4 procesadores.

De la Figura 5.12 se aprecia que las 4 topologías obtienen resultados mejores que el AG serial. La topología que obtuvo el peor resultado es la de estrella, después la topología de hipercubo, después la de anillo y el mejor resultado lo obtuvo la topología de red. Estos fueron los resultados obtenidos en una sola ejecución (prueba generacional) de los AGs de islas. Las pruebas independientes (Figura 5.13) muestran los resultados un poco distintos y serán analizados a continuación.
En general un esquema de grano grueso permite a cada procesador manejar su propio conjunto de datos de forma totalmente independiente, aumentando la diversidad. El proceso de migración introducido en este modelo permite a cada isla (procesador) enriquecer su material genético, evitando en cierta medida el estancamiento. Se esperaría por lo tanto una mejor tendencia en este modelo que en los modelos globales, específicamente en el *modelo global 2*, lo cual ocurre en efecto (ver Figura 5.5 de pruebas independientes para el modelo global). También se espera que la tendencia general de cada topología no sea muy diferente. Hay sin embargo ligeras diferencias que se explicarán a continuación:

De la Figura 5.13 se tiene que en la topología de red (que obtuvo el mejor desempeño) hay una comunicación más intensa que en el resto de las topologías, permitiendo a cada procesador recibir material genético de todos sus vecinos (recordar que es una topología de red totalmente conectada) teniendo una mayor variedad al momento de elegir a los nuevos individuos que se quedarán en su isla. Esto enriquese constantemente a cada deme o isla (mayor variedad en cada deme) ampliando sus posibilidades de búsqueda, pero, debido a todo ese intercambio de individuos los demes se parecerán cada vez más a lo largo del proceso evolutivo disminuyendo la diferencia entre los demes. De cualquier forma el desempeño general es bueno.
En el caso contrario, la topología de anillo tiene un desempeño ligeramente menor al de topología de red. En este caso perece ser de beneficio el tener una comunicación escasa, aquí cada procesador recibe material genético de sólo uno de sus vecinos disminuyendo sus opciones de elección. Sin embargo, este hecho puede permitir a cada isla evolucionar de forma casi aislada y esporádicamente recibe individuos nuevos manteniendo mayor tiempo sus diferencias con sus vecinos (mayor diversidad entre demes).

El caso intermedio es la topología de hipercubo; ésta se encuentra entre la topología de red y la de anillo en cuanto a número de comunicaciones se refiere. Cada individuo se comunica con $Log_2 N$, donde N es el total de procesadores. Se puede pensar que hay un equilibrio en la diversidad a nivel de vecindarios y a nivel de individuos, reflejado esto en su desempeño que es ligeramente menor que el de anillo.
Finalmente, la topología de estrella presenta un desempeño pobre debido principalmente a su estructura maestro-esclavo y a su estrategia de migración. El proceso maestro se elige en cada generación, seleccionando al deme con el mejor valor promedio. Esto puede provocar que un deme que tenga un valor promedio regular pero mejor que el de sus vecinos sea seleccionado en repetidas ocasiones, siendo éste el único que reciba individuos de sus vecinos. Mientras esto pasa los vecinos del proceso maestro mantienen poca diversidad debido a que no llegan nuevos individuos. Esto resulta en un desempeño general menor que en las otras topologías.

Los resultados de la Figura 5.13 nos muestran los mejores y peores valores de aptitud obtenidos por cada topología en la ejecución independiente, lo cual se presenta en la Figura

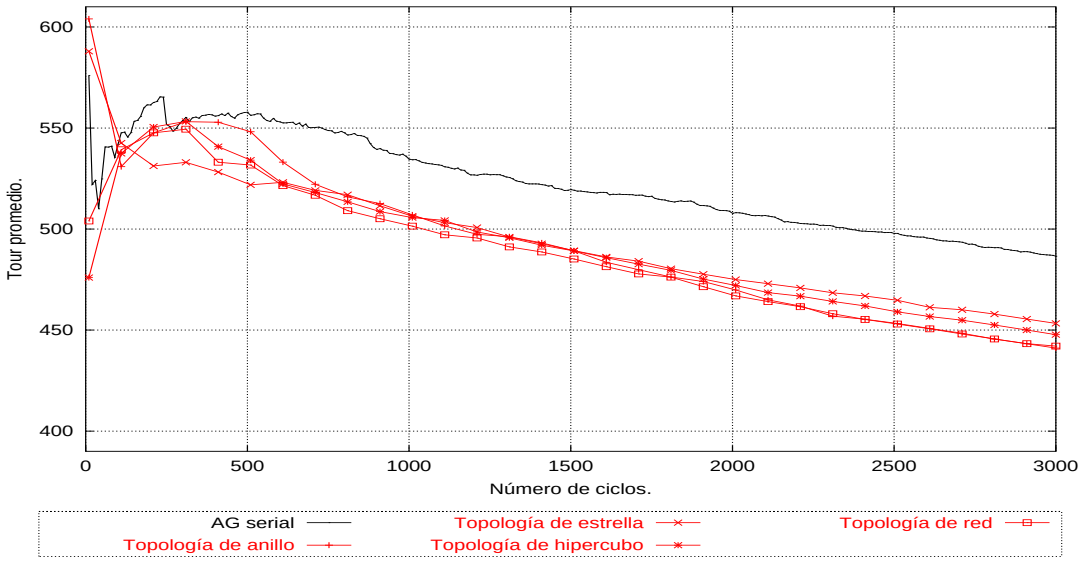

Figura 5.13: Prueba independiente. Resultados de los AGs de grano grueso usando 4 procesadores.

5.14 y la Tabla 5.8 muestra el porcentaje de ocasiones en que cada genético obtuvo el mejor valor de aptitud.

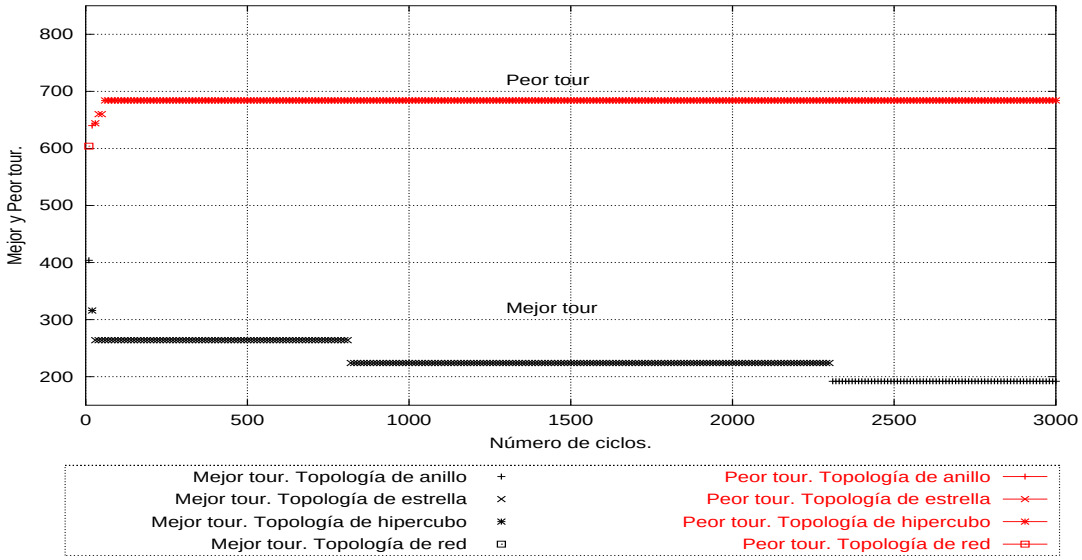

Figura 5.14: Mejor y peor tour obtenidos por los AGs con modelo islas usando 4 procesadores.

De la Tabla 5.8 se observa que la *topología de estrella* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud, después la *topología de anillo* que ganó el 23% de las pruebas independientes, después la de hipercubo con 0.333% y finalmente la topología de red.

Algoritmo genético de islas (4 EP) con:	Porcentaje
Topología de anillo	23.666666%
Topología de estrella	76.000000%
Topología de hipercubo	0.333333%
Topología de red	0.000000%

Tabla 5.8: Porcentaje de veces en que cada AG de islas con 4 procesadores encontró al mejor tour.

En la Figura 5.15 muestra el tiempo requerido por cada una de las topologías. Fue preciso cambiar la escala para observar de manera clara dichos resultados. La diferencia de cada topología es muy ligera y se encuentra en el orden de las décimas de segundo. La topología de estrella requirió el mayor tiempo cómputo, debido a que en su estructura existe un proceso central que recibe y envía mensajes convirtiéndose este en un cuello de botella al aumentar el número de procesadores. La topología de red está ligeramente debajo de la de estrella y se debe a que en una red de N nodos totalmente conectada cada uno de ellos se comunica con sus $N-1$ vecinos, requiriendo mucho tiempo de comunicación. Para el hipercubo hay menos comunicación, cada nodo envía mensajes a $log_2 N$ vecinos, siendo N el número total de nodos. Y finalmente la topología de anillo la cual requirió el menor tiempo de cómputo debido a que cada nodo se comunica con un vecino para enviar un mensaje y un vecino diferente para recibir un mensaje.

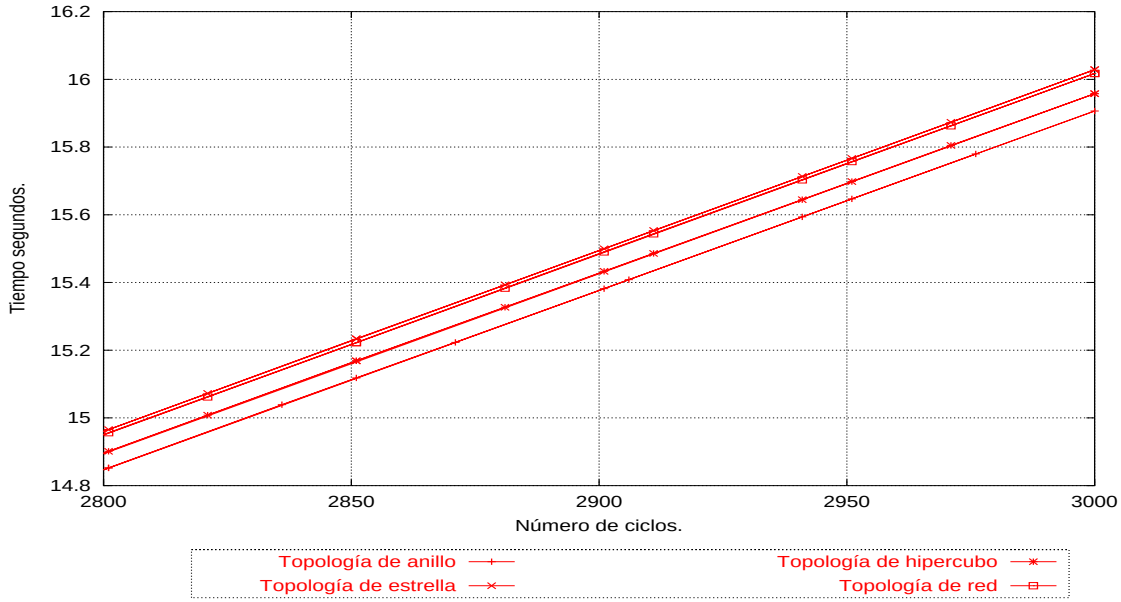

Figura 5.15: Tiempo requerido por los AGs de grano grueso usando 4 procesadores.

Pruebas usando 8 procesadores

La Figura 5.16 muestra los mejores resultados obtenidos por cada una de las topologías usando 8 procesadores. En esta ejecución el mejor resultado lo obtuvo la topología de hipercubo, despues la topología de estrella, la topología de red y el peor resultado lo obtuvo la topología de anillo. La Figura 5.17 muestra las pruebas independientes usando 8 procesadores de las 4 topologías implementadas.

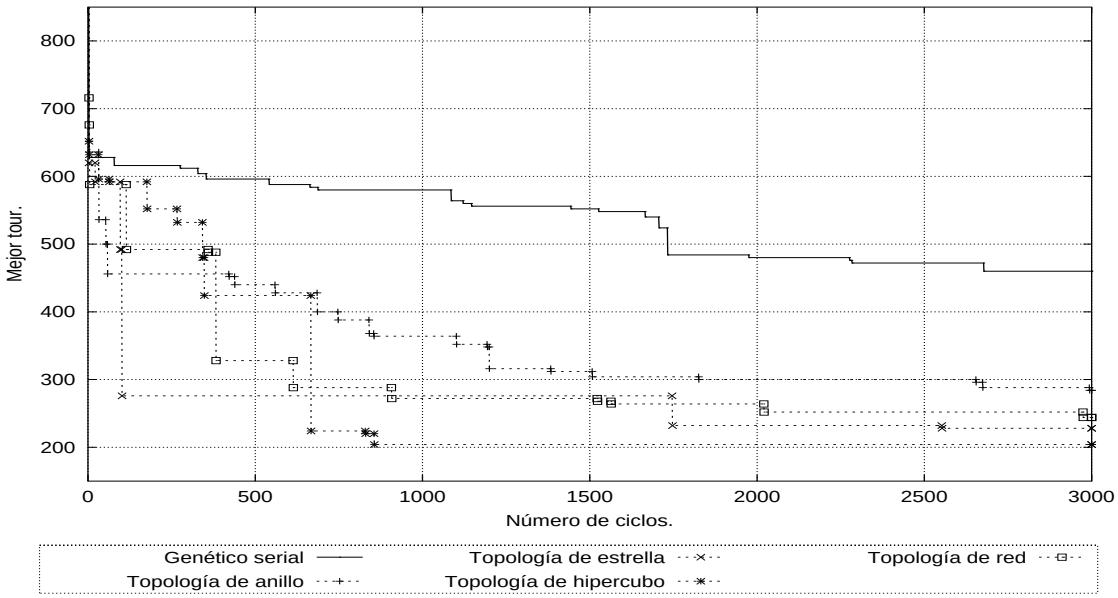

Figura 5.16: Pruebas generacionales. Resultados de los AGs de grano grueso usando 8 procesadores.

En general las 4 topologías obtuvieron mejores resultados para 8 procesadores que para 4. Esto debido a que al aumentar el número de demes se incrementa también la diversidad de individuos debido a que cada deme evoluciona de forma separada. Esporádicamente cambiarán material genético enriqueciéndose de manera mutua. Al comparar las 4 topologías entre sí se observan resultados muy similares con los resultados mostrados en la Figura 5.12 de 4 procesadores. La descripción hecha antes (para 4 procesadores) puede explicar también el comportamiento de los resultados obtenidos para 8 procesadores.

Los resultados en la Figura 5.17 no muestran los mejores y peores valores de aptitud obtenidos por cada topología en la ejecución independiente, lo cual se presenta en la Figura 5.18 y la Tabla 5.9 muestra el porcentaje de ocasiones en que cada topología obtuvo el mejor valor de aptitud.

De la Tabla 5.9 se observa que la *topología de hipercubo* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud, después la *topología de estrella* que ganó el 26% de las pruebas independientes, después la de red con 24.333% y finalmente la topología de anillo con 18.333%. En este caso la disputa entre las topologías fue más pareja.

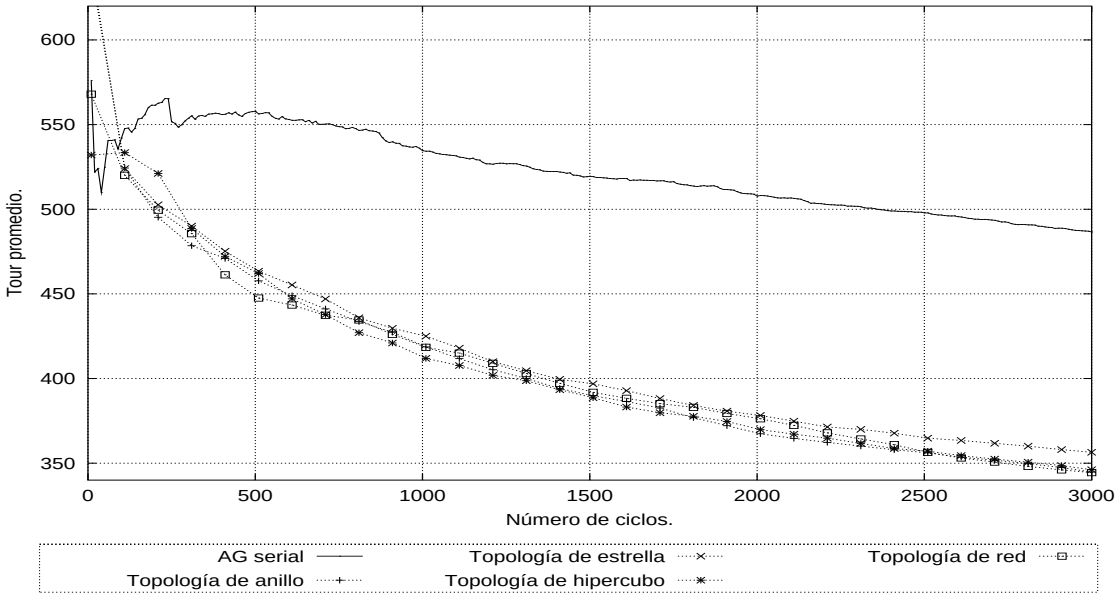

Figura 5.17: Media aritmética, pruebas independientes. Resultados de los AGs de grano grueso usando 8 procesadores.

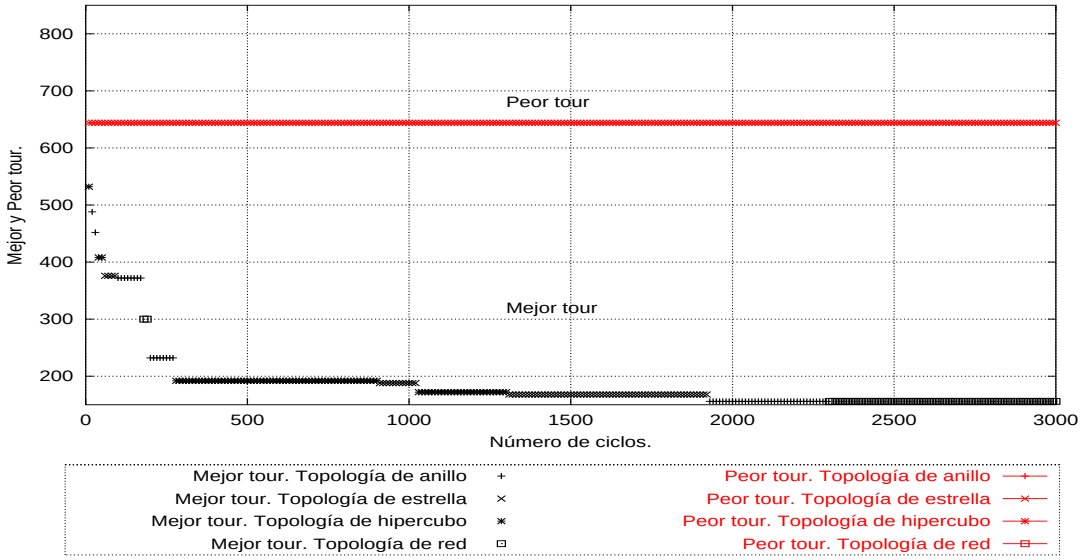

Figura 5.18: Mejor y peor tour obtenidos por los AGs con modelo islas usando 8 procesadores.

La Figura 5.19 muestra el tiempo para cada una de las topologías. Nuevamente se cambió la escala para observar de manera clara dichos resultados.

Los resultados de la Figura 5.19 muestran tiempos más pequeños que al usar 4 procesadores, además las diferencias que hay entre las topologías tiene mucha similitud con las ejecuciones para 4 procesadores confirmando lo que se describió para la Figura 5.15 del

Algoritmo genético de islas (8 EP) con:	Porcentaje
Topología de anillo	18.333334%
Topología de estrella	26.000000%
Topología de hipercubo	31.333334%
Topología de red	24.333334%

Tabla 5.9: Porcentaje de veces en que cada AG de islas con 8 procesadores encontró al mejor tour.

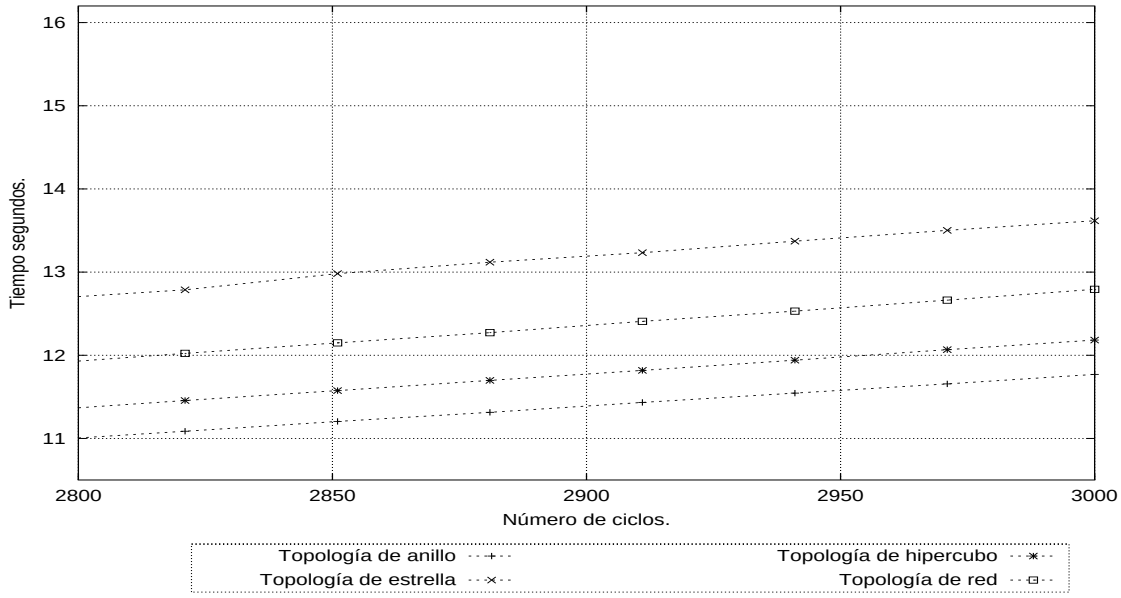

Figura 5.19: Tiempo requerido por los AGs de grano grueso usando 8 procesadores.

modelo de islas de 4 procesadores. Los tiempos en este caso están por debajo de los 14 segundos.

Modelo de grano fino

Además de los parámetros del código serial, del modelo paralelo global y del modelo paralelo de islas se consideran dos parámetros más para este modelo, la Tabla 5.10 muestra cuáles son junto con sus valores elegidos.

Parámetro	Valor
Tipo de vecindario	Lineal, Compacto
Radio del vecindario	1

Tabla 5.10: Valores de los parámetros de un AG celular.

Debido a la falta de una arquitectura masivamente paralela propia para este modelo, se

diseñó e implementó: una simulación serial de un genético celular y una versión paralela de dicha simulación serial.

Para la simulación serial se realizaron pruebas variando el radio del vecindario, se observó que no había una ganancia considerable al establecer tamaños de radio mayores que 1 y en algunas ejecuciones se obtuvieron resultados peores. Esto se observó para ambos tipos de vecindario, *lineal y compacto*. Por ello se estableció el radio del vecindario a 1, tanto para la simulación serial como paralela del genético celular.

Pruebas Usando 4 procesadores

La Figura 5.20 muestra una comparación de los resultados obtenidos por ambas implementaciones serial convencional, celular serial y paralela usando 4 procesadores. De la Figura 5.20 se observa que las dos implementaciones con *vecindario lineal* (serial y paralela) obtienen los valores de aptitud más bajos. La implementación con *vecindario compacto* serial es la que obtuvo el valor de aptitud más alto y el *vecindario lineal* paralelo obtuvo el mejor resultado. La Figura 5.21 muestra los resultados de las pruebas independientes.

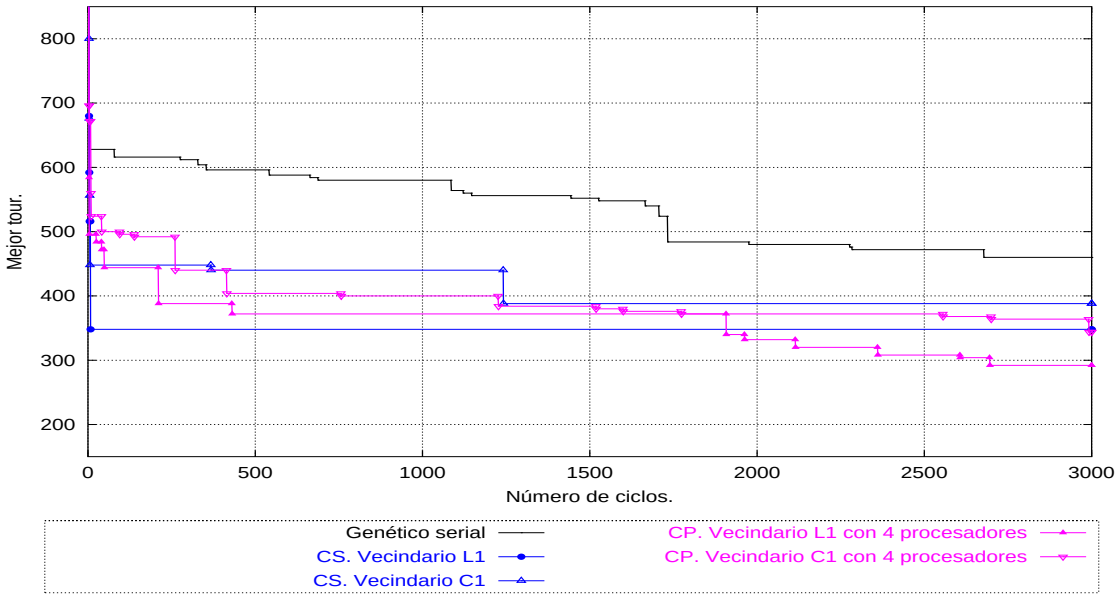

Figura 5.20: Resultados de la simulación del AG de grano fino serial y paralelo con 4 procesadores.

De la Figura 5.21 se pueden concluir algunos puntos: una de las principales ventajas de un modelo celular es la formación de vecindarios que evolucionen aisladamente y se comuniquen a través del solapamiento[1], es decir el traslape de vecindarios. Esto no es así en la simulación serial debido a que el procesamiento de cada individuo en la malla se hace de manera secuencial. Por tanto, se esperaría que el comportamiento de las dos simulaciones seriales sea muy similar. Esto se observa mejor en las pruebas independientes (Figura 5.21).

[1]Dicho solapamiento se refiere a que cada individuo de un vecindario pertenece a su vez al vecindario contiguo. Lo anterior permite la transferencia de información genética (ver Figura 2.8 del capítulo 2).

En las implementaciones celulares paralelas, el solapamiento de vecindarios contribuye a la propagación de individuos buenos y malos, pero es preciso considerar que al incrementar el radio de un vecindario se incrementará también el número de individuos que se encuentran en más de un deme (debido al solapamiento), es decir, los vecindarios son más parecidos a medida que se incrementa su radio; esto puede afectar a la diversidad. Por el contrario los vecindarios con radio pequeño quedan más aislados permitiendo una evolución más independiente (similar al modelo de grano grueso), aunque eventualmente el solapamiento propagará a los individuos seleccionados. En las pruebas mostradas, el radio de los vecindarios es 1 pero el *vecindario compacto* ocupa un área mallor en la maya respecto al *vecindario lineal*. La diferencia en número de individuos entre los vecindarios lineal y compacto no es muy grande, pero como se observa en la Figura 5.21 en las pruebas independientes fue suficiente para lograr un mejor rendimiento del *vecindario lineal* paralelo que el *vecindario compacto* paralelo.

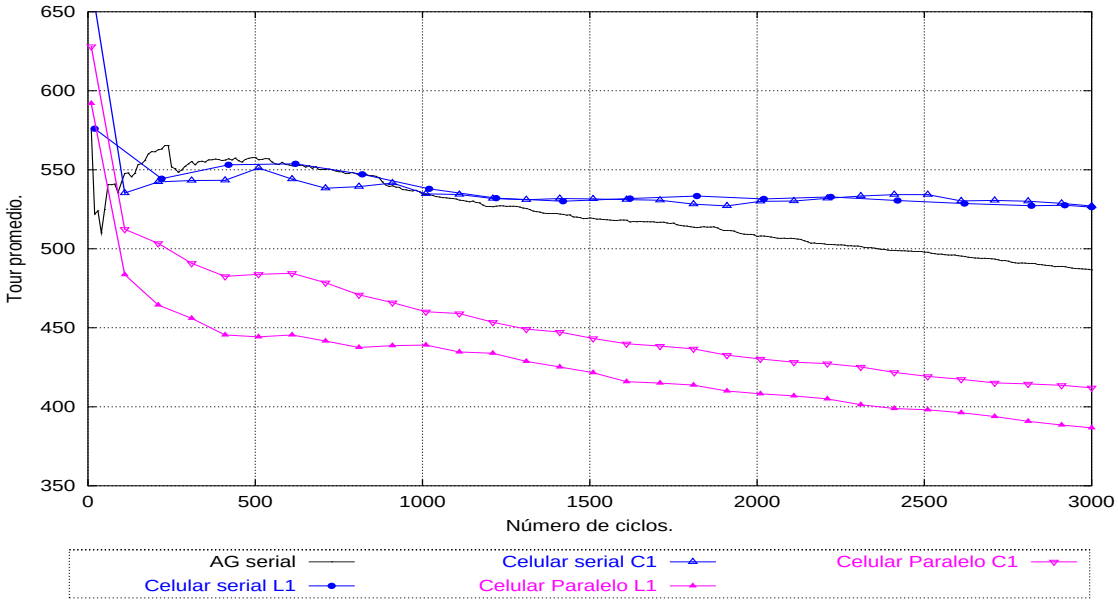

Figura 5.21: Resultados de la simulación del AG de grano fino serial y paralelo con 4 procesadores.

Los resultados de la Figura 5.21 no muestran los mejores y peores valores de aptitud obtenidos por cada vecindario tanto serial como paralelo en la ejecución independiente, lo cual se presenta en las Figuras 5.22 para las pruebas seriales y 5.23 para las pruebas paralelas, y las Tablas 5.11 y 5.12 muestran el porcentaje de ocasiones en que cada vecindario obtuvo el mejor valor de aptitud.

De la Tabla 5.11 del celular serial se observa que el *vecindario compacto* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud con un 81.33%, y el *vecindario lineal* queda con un 18.66%. De la Tabla 5.12 del AG celular paralelo se observa que nuevamente el *vecindario lineal* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud con un 96.66%, mientras que el *vecindario compacto* tiene 3.333%.

Finalmente la Figura 5.24 muestran los tiempos requeridos por cada vecindario lineal y

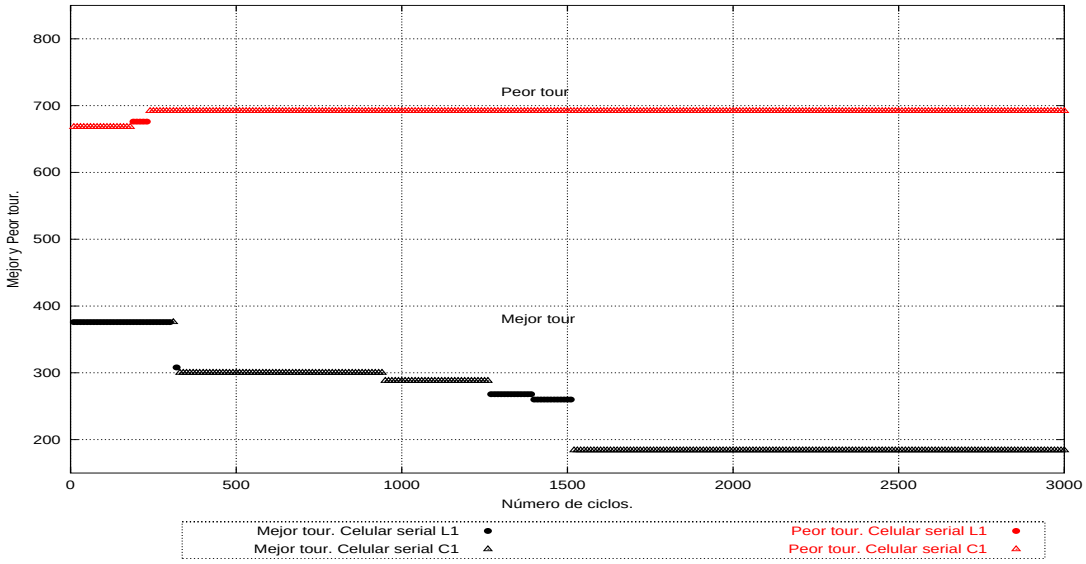

Figura 5.22: Mejor y peor tour obtenidos por los AGs de **modelo celular serial**

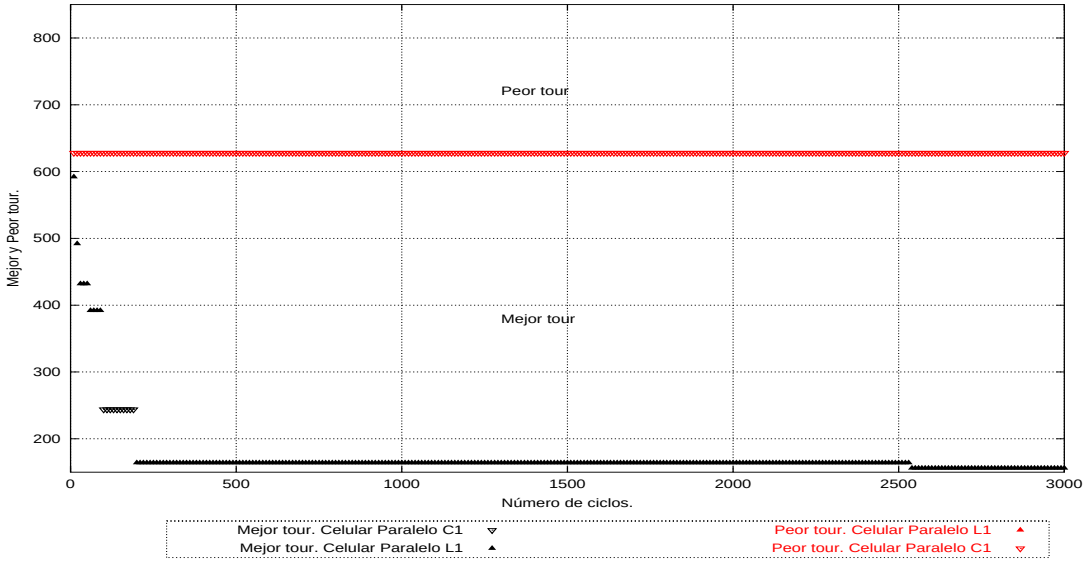

Figura 5.23: Mejor y peor tour obtenidos por los AGs de **modelo celular paralelo** usando 4 procesadores.

Algoritmo genético celular serial con:	**Porcentaje**
Vecinadrio Lineal	18.666666%
Vecindario compacto	81.333336%

Tabla 5.11: Porcentaje de veces en que cada AG celular serial encontró al mejor tour.

Algoritmo genético celular paralelo (4 EP) con:	Porcentaje
Vecinadrio Lineal	96.666664%
Vecindario compacto	3.333333%

Tabla 5.12: Porcentaje de veces en que cada AG celular paralelo con 4 procesadores encontró al mejor tour.

compacto. Las dos implementaciones seriales del AG celular requieren tiempos muy similares a la implementación del AG convencional, esto debido a su ejecución secuencial. Las dos versiones paralelas encuentran soluciones a un menor costo de tiempo, sin embargo, esa diferencia en tiempo no es tan grande como en los modelos de grano grueso porque hay una mayor comunicación entre procesadores (una migración de individuos en cada generación). Al igual que en la versión serial el vecindario lineal requirió menos tiempo que el compacto, esto debido a que el vecindario lineal contiene menos individuos.

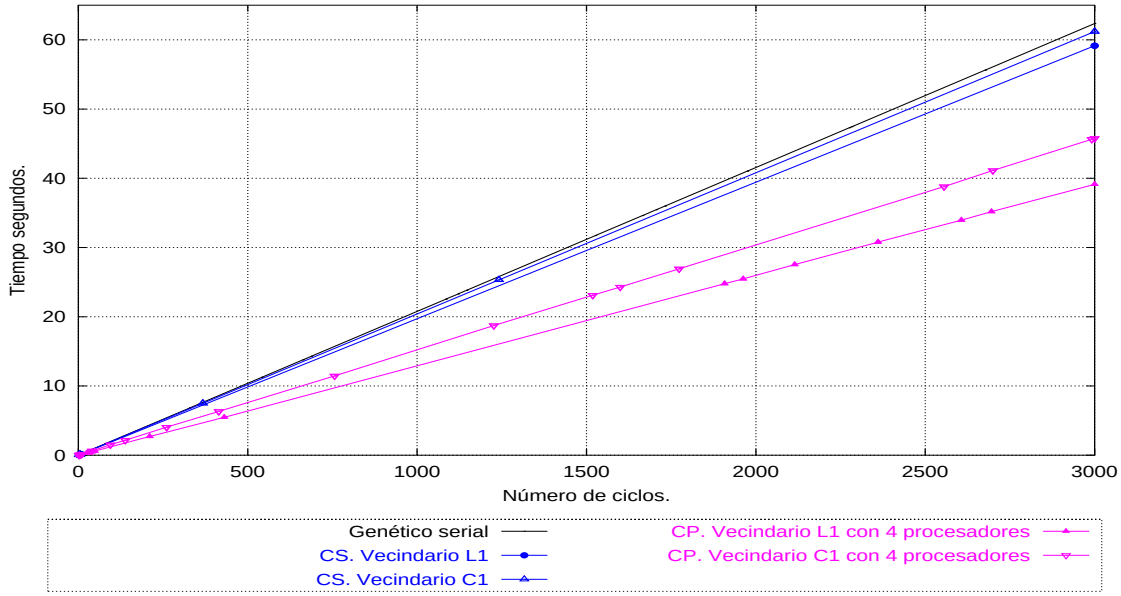

Figura 5.24: Tiempos requeridos por la simulación del AG de grano fino serial y paralelo (4 procesadores)

Pruebas para 8 procesadores

La Figura 5.25 muestra una comparación de los resultados obtenidos por ambas implementaciones serial convencional y celular paralela usando 8 procesadores.

De la Figura 5.25 se observa que las implementaciones paralelas usado 8 procesadores obtuvieron al menos para esta ejecución resultados de aptitud mejores que al usar 4 procesadores. La Figura 5.26 muestra las ejecuciones independientes. Al usar 8 elementos de proceso en este modelo se obtiene un comportamiento similar a las pruebas efectuadas en un AG de grano grueso, es decir, al existir más procesadores aumenta también el número de demes que evolucionan de forma separada (esto es más notorio en el vecindario lineal

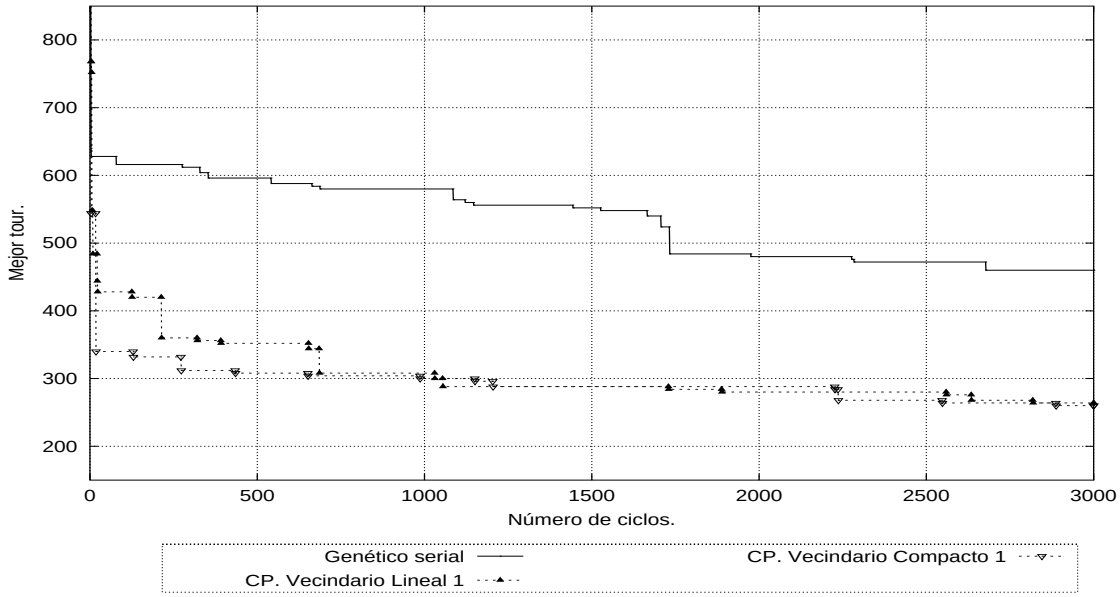

Figura 5.25: Resultados de la simulación del AG de grano fino paralelo con 8 procesadores.

debido a su tamaño) aumentando con esto la variedad, compartiendo individuos a lo largo del proceso genético. Se observa claramente que lo anterior permitió mejorar el desempeño del paralelo celular con 8 procesadores ya que su tendencia está por debajo (Figura 5.26) de las ejecuciones de 4 procesadores. El vecindario lineal obtuvo mejores promedio que el vecindario compacto, la razón de ello se describió para 4 procesadores.

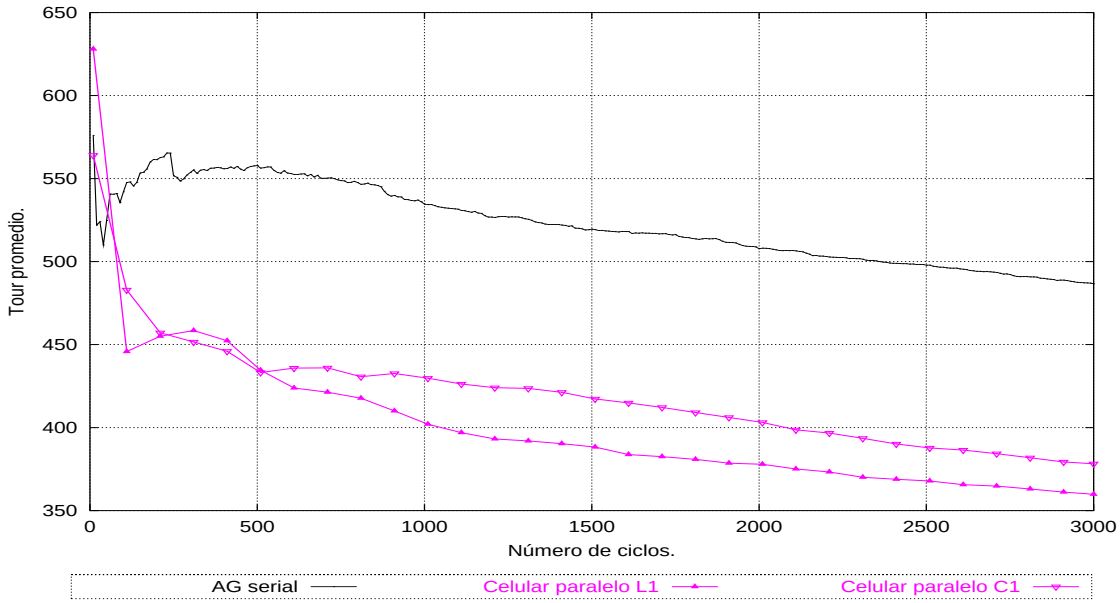

Figura 5.26: Resultados de la simulación del AG de grano fino paralelo con 8 procesadores.

Los resultados de la Figura 5.26 no muestran los mejores y peores valores de aptitud obtenidos por cada vecindario paralelo en la ejecución independiente, lo cual se muestra en la Figura 5.27 y la Tabla 5.13 muestra el porcentaje de ocasiones en que cada vecindario obtuvo el mejor valor de aptitud.

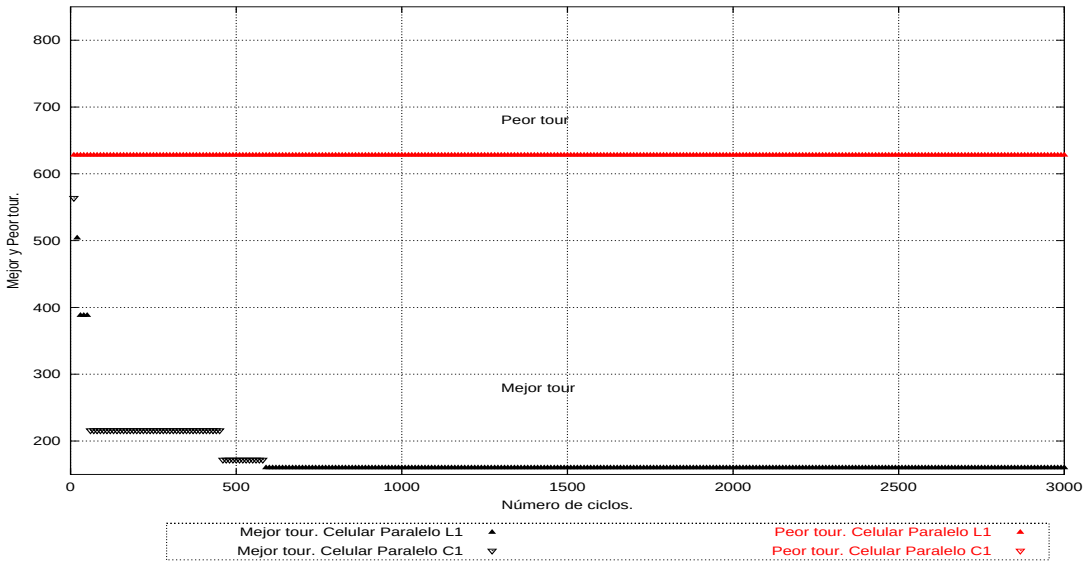

Figura 5.27: Mejor y peor tour obtenidos por los AGs de modelo celular paralelo usando 8 procesadores.

De la Tabla 5.13 se observa que el *vecindario lineal* mantuvo en la mayoría de las pruebas a los mejores valores de aptitud con un 82%, dejando al *vecindario compacto* con el 18%.

Algoritmo genético celular paralelo (8 EP) con:	**Porcentaje**
Vecindario Lineal	82.0000%
Vecindario compacto	18.0000%

Tabla 5.13: Porcentaje de veces en que cada AG celular paralelo con 8 procesadores encontró al mejor tour.

Finalmente la Figura 5.28 muestra los tiempos requeridos por cada vecindario (lineal y compacto) paralelo con 8 procesadores. Como se observa las dos versiones paralelas encuentran soluciones requiriendo menos tiempo que al usar 4 procesadores. La ganancia no es muy grande debido principalmente a las comunicaciones, éstas se incrementan al aumentar el número total de procesadores; sin embargo no son excesivas permitiendo al AG reducir su tiempo de ejecución. Nuevamente la versión paralela con vecindario lineal requirió menos tiempo que el compacto, confirmando lo expuesto en las pruebas de 4 procesadores.

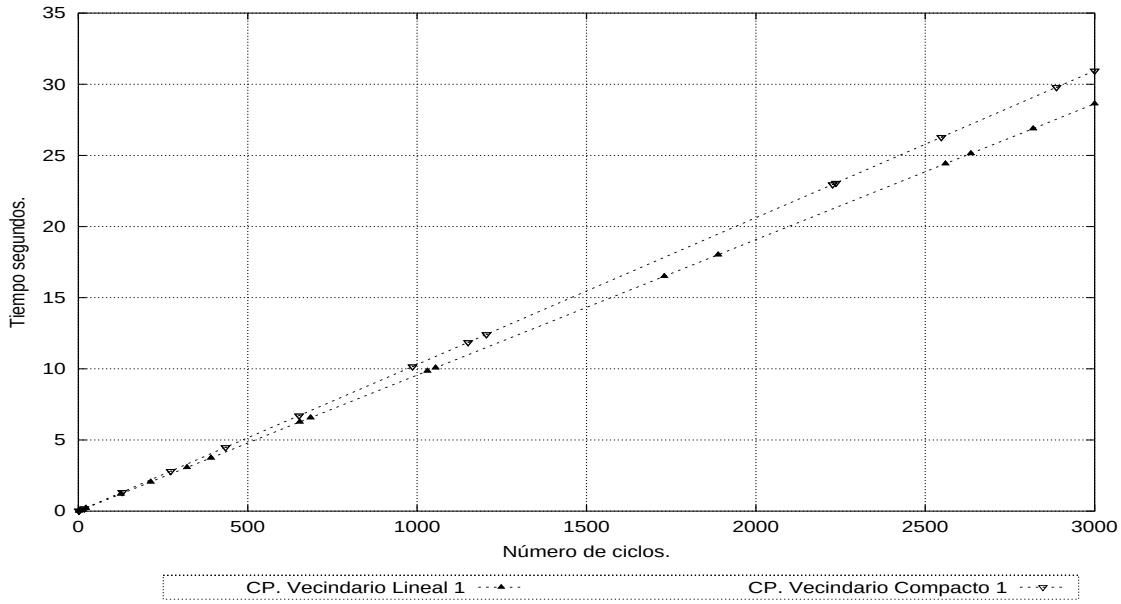

Figura 5.28: Tiempos requeridos por la simulación del AG de grano fino paralelo (8 procesadores)

5.3.3 Comparación de los mejores resultados

Con el fin de ver una comparación del rendimiento de cada algoritmo implementado se mostrarán únicamente los mejores resultados obtenidos en las pruebas independientes para cada uno de los modelos paralelos usando 4 y 8 procesadores. Se inicia con las pruebas generacionales, independientes, de tiempos y de aceleración con 4 procesadores y se continua con las pruebas para 8 procesadores. En cada gráfica se mostrará también los resultados obtenidos por el AG convencional.

Para el modelo global los mejores resultados fueron obtenidos por la implementación que paraleliza 4 procesos. En el modelo de islas los mejores resultados fueron obtenidos por la topología de hipercubo y la topología de anillo, para el modelo celular, la simulación paralela con vecindario compacto es la que encontró los resultados más bajos.

Pruebas independientes con 4 procesadores

Pruebas para 4 procesadores

Los mejores algoritmos de cada modelo para pruebas independientes con 4 procesadores se listan en la Tabla 5.14.

La Figura 5.29 muestra los resultados obtenidos por los mejores algoritmos para cada modelo paralelo usando 4 procesadores. Se observa que el *modelo paralelo celular L1* obtuvo el mejor desempeño de todos los genéticos. Parte de la razón de estos resultados ya se explicó en las figuras mostradas anteriormente. Algunas otras razones se escriben a continuación.
La estrategia usada por el algoritmo paralelo celular es muy diferente a los otros dos modelos paralelos, ya que además de dividir la población en demes (como lo hacen las otras

Modelo	Algoritmo
Global	*Modelo global 2*
Islas	*Topología de red*
Celular	*Paralelo L1*

Tabla 5.14: Mejores algoritmos en las pruebas independientes para 4 procesadores.

estrategias) cada procesador ordena sus individuos en forma de malla bidimensional. Esa distribución de malla la hacen tomando en cuenta la posición de las mallas vecinas formando así una sola malla lógica. Debido a esto, algunos individuos de cualquier deme tendrán a sus vecinos del NORTE o del SUR en demes contiguos, debiendo ser necesario aplicar mecanismos de migración en cada generación. La diferencia fundamental con la simulación celular serial radica en que aún y cuando se procesa de forma lógica una sola malla, cada submalla evoluciona de forma independiente intercambiando individuos al finalizar el procesamiento de cada submalla, aprovechando más las características de un genético celular.
El segundo mejor desempeño es el *modelo de islas con topología de red*. El *modelo global 2* y la *topología de red* son muy parecidas, a excepción de algunos módulos genéticos (que son ejecutados de manera serial en el genético global) y principalmente del operador de migración el cual marca la diferencia de rendimiento observada en la Figura 5.29.

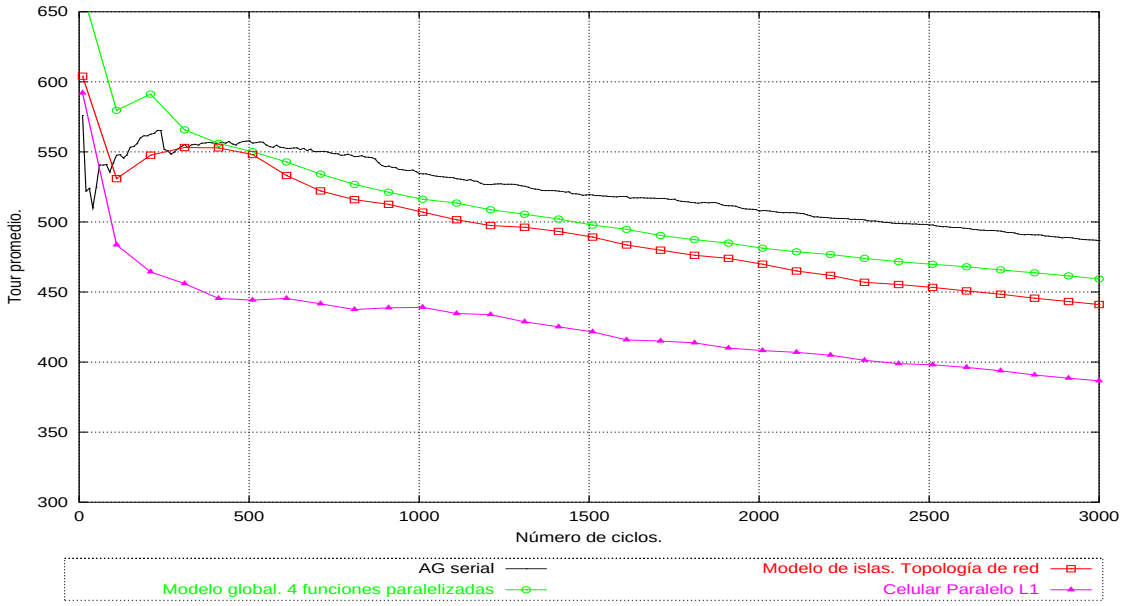

Figura 5.29: Pruebas independientes. Comparación de los mejores resultados de cada modelo paralelo (4 procesadores)

La Figura 5.30 muestra el tiempo que tarda cada modelo para la búsqueda genética. El modelo que requiere más tiempo es el genético celular L1, debido al gran número de comunicaciones. El requerimiento de tiempo necesario para los modelos global y de islas es muy parecido, pero sus diferencias se observan mejor en la Figura 5.31. La razón de que

ambos tiempos sean similares es porque: Por una parte el modelo global ejecuta algunos módulos de forma secuencial como son: *Creación de la población inicial, aplicación de la técnica elitista y algunas otras funciones.* Y por su parte el modelo de grano grueso ejecuta todo el código de manera paralela pero incluye mecanismos de migración entre procesadores. De ésta ejecución secuencial (*modelo global 2*) y el envío de mensajes (*modelo de islas-red*) evitan que uno de los dos genéticos establezca un dominio claro del tiempo.

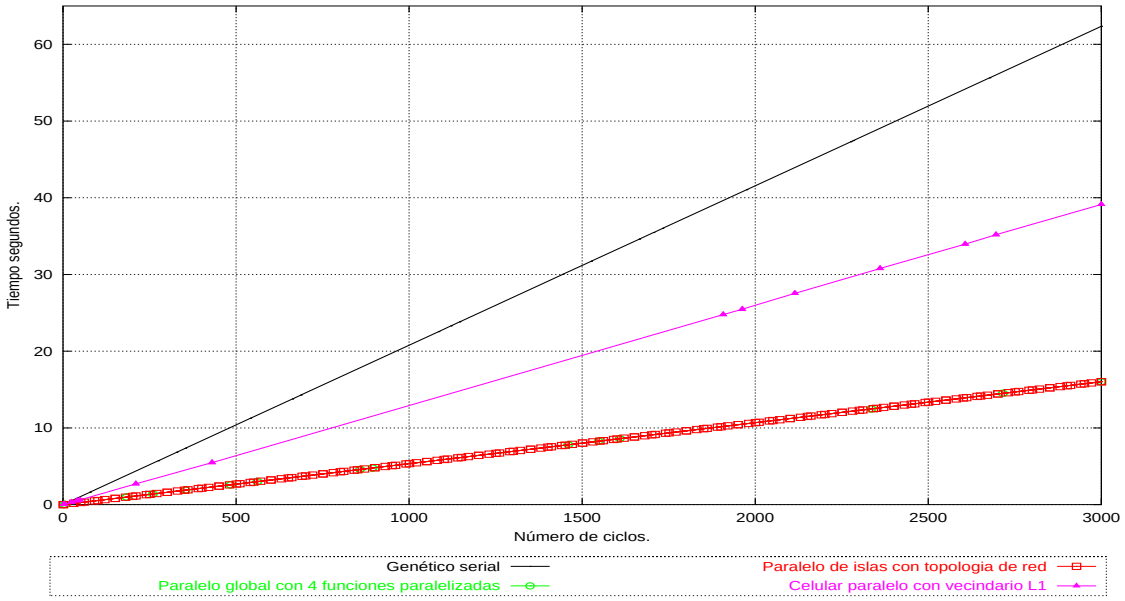

Figura 5.30: Tiempos requeridos por los modelos paralelos implementados (4 procesadores).

Modelo	Algoritmo
Global	*Modelo global 2*
Islas	*Topología de red*
Celular	*Paralelo L1*

Tabla 5.15: Mejores algoritmos en las pruebas independientes para 8 procesadores.

Pruebas para 8 procesadores

Los mejores algoritmos de cada modelo para pruebas independientes con 8 procesadores se listan en la Tabla 5.15. La Figura 5.32 muestra los resultados obtenidos por los mejores algoritmos para cada modelo paralelo usando 8 procesadores. Se observa que para estas pruebas el *modelo de islas con topología de red* obtuvo el mejor desempeño de todos los genéticos. El hecho de aumentar el número de procesadores permite al genético de islas incrementar también su desempeño, parte de esto ya se explicó en las Figuras de los modelos de islas para 8 procesadores. El genético celular también aumentó su rendimiento (pero en menor medida) debido al aumento de procesadores y finalmente el genético global que no tuvo una gran diferencia respecto de las pruebas con 4 procesadores. En los 3 casos el hecho de aumentar el número de procesadores mejora también su desempeño.

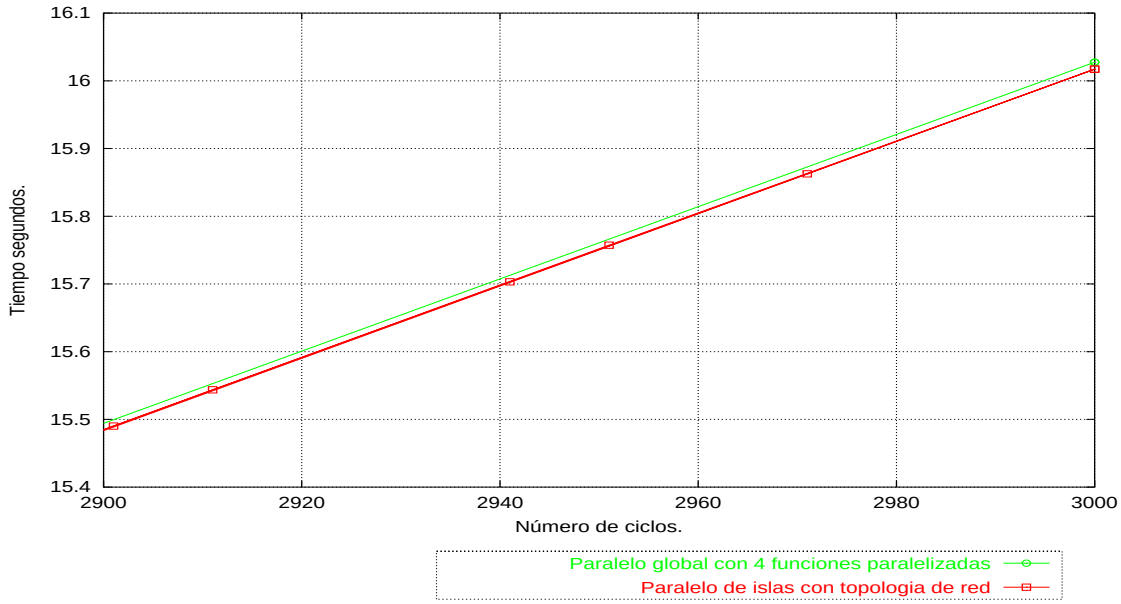

Figura 5.31: Acercamiento. Tiempos requeridos por los modelos paralelos global y de islas (4 procesadores).

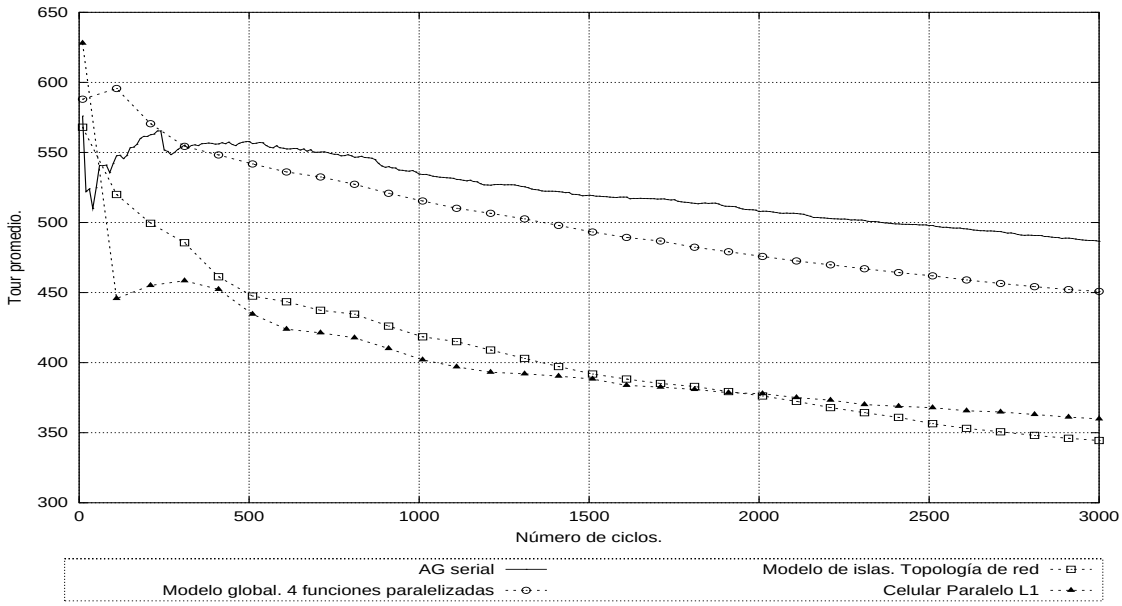

Figura 5.32: Pruebas independientes. Comparación de los mejores resultados de cada modelo paralelo (8 procesadores)

La Figura 5.33 muestra el tiempo que tarda cada modelo para la búsqueda genética. Cada modelo disminuyó el tiempo requerido usando 8 procesadores que al usar 4. La diferencia de los tiempos entre las 3 topologías es muy similar a la Figura 5.30 donde se usan 4

procesadores.

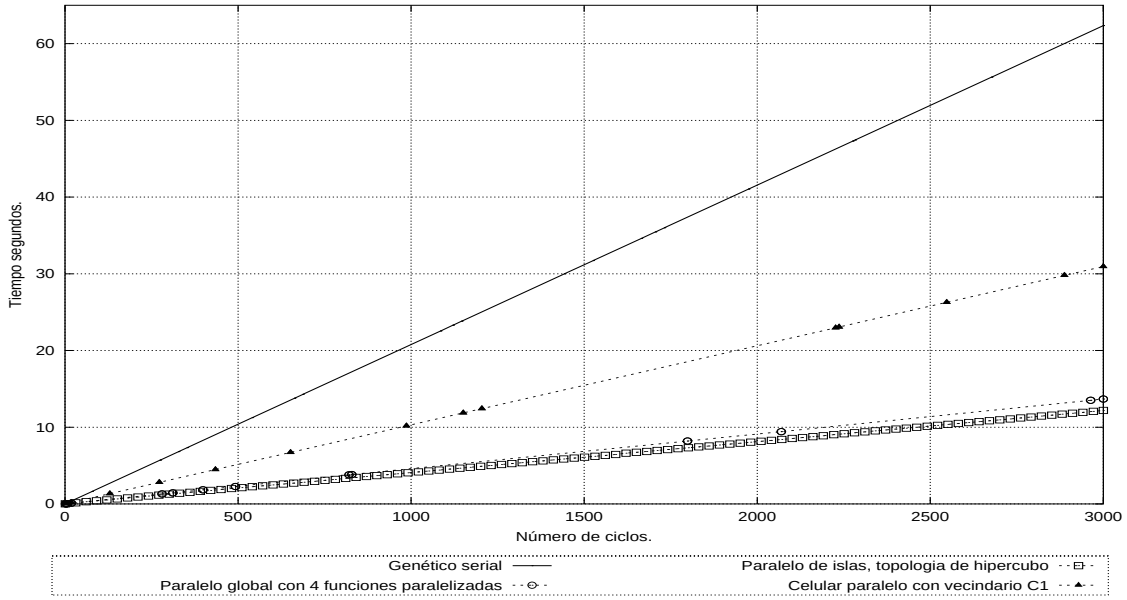

Figura 5.33: Tiempos requeridos por los modelos paralelos implementados (8 procesadores).

Gráficas de aceleración

Finalmente se describirá la aceleración que obtuvo cada AG paralelo. La Tabla 5.16 muestra la aceleración de cada uno de los mejores AG paralelos de cada modelo para 2, 4 y 8 procesadores.

Procesadores	Tiempo de ejecución	Aceleración
Modelo global 2		
2	31.1837	1.9995
4	16.0275	3.8904
8	13.6720	4.560729
Modelo de islas-red		
2	31.227	1.9970
4	16.0172	3.8929
8	12.7929	4.8741
Modelo paralelo celular L1		
2	46.9198	1.3289
4	39.1263	1.5936
8	28.6343	2.1776

Tabla 5.16: Tiempos y aceleración de los mejores AGs paralelos para 2,4 y 8 procesadores.

La Figura 5.34 muestra la gráfica de aceleración de la Tabla 5.16. En ella se observa cómo los modelos global y de grano grueso tienen una aceleración muy semejante a la ideal cuando usan 2 y 4 procesadores, pero ésta disimuye considerablemente al usar 8 procesadores. Por su parte la aceleración del genético de grano fino cuando usa 2 y 4 procesadores es considerablemente menor a las de las otras dos topologías y, como se observa la aceleración es aún menor al usar ocho procesadores. Lo anterior se debe principalmente al costo en comunicaciones que experimenta cada algoritmo. El algoritmo que tiene el mayor número de comunicaciones es el genético de grano fino, después el genético de grano grueso y finalmente el modelo global. En el modelo global además de los mecanismos de sincronización se da un aspecto adicional para la disminución de su aceleración y es que muchas de las funciones se ejecutan de manera secuencial. Lo anterior nos dice que al menos para las pruebas mostradas es conveniente usar 2, 4 y 8 procesadores para aumentar el rendimiento de los AG paralelos de grano grueso y modelo global; pero para el modelo de grano fino resulta peor usar 8 procesadores que usar 4 debido a que su aceleración disminuye de forma considerable.

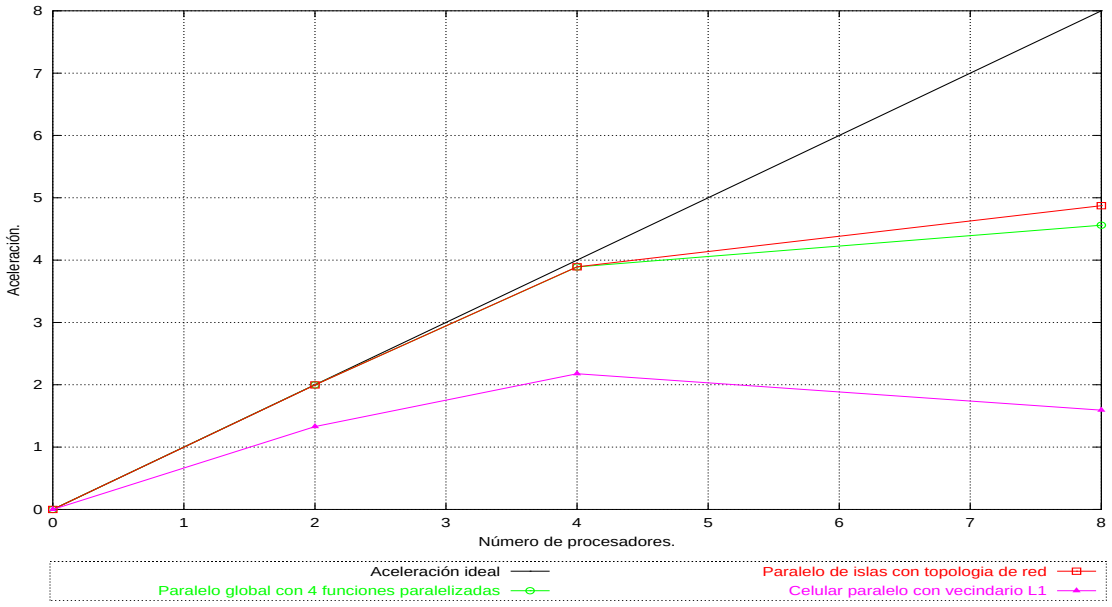

Figura 5.34: Gráfica de aceleración de los mejores AGs paralelos para 2,4 y 8 procesadores.

5.3.4 Conclusiones de los resultados obtenidos

En algunas de las pruebas hechas al AG convencional se observó que son técnicas relativamente lentas y poco efectivas para buscar óptimos en espacios de soluciones muy grandes como el caso del TSP. Por ello se buscaron mecanismos que mejoren su desempeño (tiempo-efectividad). Este mecanismo es el paralelismo del AG convencional, para ello se estudiaron tres modelos: *AG paralelo con modelo global, AG paralelo de grano grueso y el AG paralelo de grano fino.*

Para el modelo global se realizaron dos implementaciones, en la primera se paralelizan 3 operadores genéticos (*modelo global 1*) y en la segunda se paralelizan 4 operadores (*modelo global 2*). El *modelo global 1* mejoró el tiempo de búsqueda del algoritmo convencional, pero su efectividad fue casi la misma, debido a que sus estructuras y funcionamiento son muy similares. El *modelo global 2* tuvo un mejor desempeño (tiempo-efectividad), en este caso su funcionamiento respecto del genético serial es diferente ya que incluye al operador de selección en el procesamiento paralelo, es decir, la selección se efectúa de forma separada en cada deme y no en una sola población.

En el modelo de grano grueso se hicieron 4 implementaciones basadas en las topologías de: *anillo, estrella, hipercubo y red.* Las 4 topologías mejoraron notablemente el desempeño (tiempo-efectividad) de los AG convencionales e incluso del modelo global. El operador de migración incluido en este modelo permite a cada isla aumentar su diversidad genética, aumentando con ello sus posibilidades de búsqueda. Las diferencias principales entre cada topología están relacionadas directamente con la organización lógica de los nodos de su red ya que en base a ella se determinará la forma de enviar y recibir mensajes y la cantidad de éstos. Se observó que una *topología de estrella* es la que consume más tiempo debido a que el proceso maestro puede llegar a convertirse en un cuello de botella. En la *topología de red* se tiene también un gran número de comunicaciones ya que todos los nodos envían mensajes a todos sus vecinos. La *topología de anillo* es la que requirió el menor tiempo de cómputo. En ella cada nodo envía mensajes a un nodo y recibe de otro. Finalmente el punto intermedio entre la *topología de red* y la de *anillo* es la *topología de hipercubo*; en ella cada nodo se comunica con $log_2 N$ vecinos, no habiendo por lo tanto un exceso de comunicación (*topología de red*), muy poca comunicación (*topología de anillo*) ni cuellos de botella (*topología de estrella*).

Por último, está el modelo de grano fino para el cual se creó una versión serial y una versión paralela. En las gráficas se observó claramente que los mejores resultados se obtuvieron por las versiones paralelas. Se estableció el radio del vecindario igual a 1 ya que radios más grandes no ofrecen más ventajas. De las ejecuciones mostradas el *AG celular L1* siempre tuvo el mejor desempeño (tiempo-efectividad) tanto en la versión serial como paralela. Lo anterior se debe a que vecindarios más grandes (como es el caso del *celular C1*) tienden a ser similares con mayor rapidez en el transcurso del proceso evolutivo que los vecindarios más pequeños como el *vecindario L1.*

En general los resultados obtenidos en los tres modelos paralelos implementados en éste trabajo mejoraron el desempeño de los AG convencionales que fue el objetivo principal de éste trabajo.

Conclusiones

La utilización de algoritmos genéticos para resolver problemas de optimización combinatoria ha experimentado un gran auge en los últimos tiempos debido a que el espacio de soluciones de este tipo de problemas es excesivamente grande y un método convencional sería poco viable para su resolución exacta o aproximada. Los AGs encuentran generalmente soluciones aproximadas al problema atacado y aunque son mucho más rápidos que una estrategia de búsqueda exhaustiva, requieren de mucho tiempo de ejecución cuando los problemas son relativamente grandes; además las soluciones encontradas pueden ser mínimos locales que estén muy lejos de la solución exacta. Esto puede hacer a los AGs estrategias poco adecuadas para problemas combinatorios, por ello se han analizado también mecanismos y técnicas que mejoren su desempeño, tanto en tiempo de ejecución como en efectividad de búsqueda de la mejor solución. En este contexto el estudio de métodos paralelos para aumentar la eficiencia de un algoritmo genético convencional ha dado lugar a numerosos modelos, algunos de ellos fueron estudiados en este trabajo y son: *modelo global, modelo de grano grueso y el modelo de grano fino.*

Entonces, el propósito del presente trabajo es mostrar cómo al usar un AG secuencial para resolver un problema de optimización combinatoria, como el *problema del agente viajero (TSP)* puede aumentar su eficiencia al incrementar: el tamaño de la población o el número de generaciones, pero debido a esto se vuelve un algoritmo más lento. También se observa cómo al usar las ténicas paralelas mencionadas en el primer párrafo es posible disminuir la velocidad de ejecución de un AG sin disminuir su efectividad de búsqueda e incluso para algunos modelos paralelos aumenta dicha efectividad, llegando a encontrar soluciones mejores que la versión convencional. Durante el desarrollo del presente trabajo se implementaron diversos algoritmos genéticos para la resolución de un problema clásico en optimización combinatoria conocido como el *problemadelagenteviajero*. Primero se diseñó e implementó una versión convencional que resolviera dicho problema y después, a partir de ese genético serial, se usaron diversas estrategias para la implementación de las técnicas paralelas. La primera de ellas fue la técnica paralela conocida como *modelo global* del cual después de realizar un análisis de rendimiento al genético serial, se desprendieron dos versiones:

1. *modelo global 1.* Se ejecutan 3 operadores en forma paralela: *evaluación, cruza y mutación.*

2. *modelo global 2.* Se ejecutan 4 operadores en forma paralela: *evaluación, selección, cruza y mutación.*

De las cuales el *modelo global 1* mejoró sólo en la velocidad de ejecución pero no en la efectividad y el *modelo global 2* mejoró en ambos aspectos tiempo-efectividad. La segunda técnica fue el *modelo de grano grueso* del cual se implementaron 4 topologías diferentes: *anillo, estrella, hipercubo y red*. Las 4 topologías observaron una gran mejora de tiempo-efectividad respecto del AG convencional e incluso de los *modelos globales*. Y finalmente la tercer técnica implementada fue el *modelo de grano fino* que mostró también buen desempeño en tiempo (aunque menor que las otras técnicas, excepto con el *modelo global 1*). En cuanto a efectividad su desempeño fue mejor que en los *modelos globales* pero menor que el *modelo de grano grueso*. Tras el diseño, implementación y evaluación de cada algoritmo genético se obtuvieron diversos resultados que se resumirán en los siguientes párrafos.

En las pruebas del AG convencional para resolver el TSP con 40 ciudades se observó que al incrementar el tamaño de la población de individuos y del número de generaciones el AG se hace muy lento, sin embargo aumenta su efectividad en la búsqueda de la solución.
En lo que respecta a las pruebas realizadas al AG con modelo global se observa que cuando no se incluye el operador de selección en el paralelismo se mejora el tiempo de búsqueda respecto del algoritmo convencional, pero no su efectividad, debido a que sus estructuras y funcionamiento son muy similares. Por el contrario cuando se incluye en los procedimientos paralelos dicho operador se observa una clara mejoría en el desempeño (tiempo-efectividad) porque en este caso su funcionamiento respecto del genético serial es diferente ya que al incluir al operador de selección en el procesamiento paralelo la elección de cromosomas se efectúa de forma separada en cada deme y no de manera global.
En el modelo de grano grueso las gráficas mostraron que las 4 topologías mejoraron notablemente el desempeño (tiempo-efectividad) respecto de los AGs convencional y global. El operador de migración incluido en este modelo permite a cada isla aumentar su diversidad genética debido al intercambio de individuos, aumentando también sus posibilidades de búsqueda. Hubo diferencias ligeras en el desempeño de cada topología las cuales se relacionan directamente con la organización lógica de los nodos de su red ya que en base a ella se determina la forma de enviar y recibir mensajes y la cantidad de éstos.
Finalmente en las pruebas efectuadas a las simulaciones celulares se vío que radios de vecindarios mayores que 1 no ofrecían mayores ventajas e incluso se observaron desempeños más bajos. También los resultados mostraron que los vecindarios tipo L1 obtienen resultados (tiempo-efectividad) mejores. Lo anterior se debe a que vecindarios grandes (como es el caso del *celular C1* o radios mayores que 1) tienden a ser similares con mayor rapidez en el transcurso del proceso evolutivo que los vecindarios más pequeños como el *vecindario L1*.

En general los resultados obtenidos en los tres modelos paralelos implementados en este trabajo mejoraron el desempeño del AG convencional.

Los resultados mostrados en este trabajo son sólo una parte de todas las pruebas realizadas a los modelos paralelos implementados, sin embargo existen algunas modificaciones adicionales que podrían mejorar el desempeño de los algoritmos realizados. Sería importante que en el modelo paralelo de grano grueso sea posible establecer parámetros de entrada distintos para cada deme y ver si esto ofrece mayores ventajas o desventajas en el desempeño. También es conveniente evaluar los AG paralelos en diferentes plataformas o supercomputa-

doras, usando para ello diferentes modelos de programación y observar su comportamiento. Finalmente se podrían realizar pruebas a los AGs usando operadores de cruza y mutación diferentes y observar el desempeño obtenido comparado con lo mostrado en este reporte.

El trabajo realizado constituye una pequeña revisión de algunos de los modelos paralelos para algoritmos genéticos más importantes y muestra los beneficios de las diferentes implementaciones. No obstante, esta línea de investigación ofrece más perspectivas que no pueden ser abordadas en una sola tesis, y por ello existen aspectos que se pueden explorar en trabajos futuros. El primero de ellos es realizar un análisis teórico que justifique los resultados y conclusiones obtenidos de las pruebas realizadas a los algoritmos de esta tesis. Sería interesante también hacer las modificaciones necesarias a los AGs de este trabajo para que resuelvan un problema de diferente tipo al del TSP y hacer un análisis de pruebas y resultados con el fin de entender su comportamiento en otro tipo de problemas. Finalmente, implementar algoritmos genéticos híbridos y realizar comparaciones de rendimiento con los efectuados en este trabajo.

Bibliografía

[1] Dr. Carlos A. Coello Coello. Introducción a la computación evolutiva. CINVESTAV-IPN, departamento de Ingeniería Eléctrica, sección computación. Mayo - 2003. URL: http://delta.cs.cinvestav.mx/ ccoello/genetic.html

[2] Jose Luis González Velarde y Roger Z. Ríos Mercado. Investigación de operaciones en acción: Aplicaciones del TSP en problemas de manufactura y logística. UANL, Ingenierias. Vol. II. No. 4. 1999.
URL: www.uanl.mx/publicaciones/ingenierias/4/pdf/

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Cliff Stein. Introduction to algorithms. McGraw-Hill 1990.

[4] D. Goldberg. Genetic Algorithms in search optimization and Machine Learning. Addison Wesley 1989.

[5] John H. Holland. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975.

[6] Goldberg, David E. Genetic Algorithms in search, Optimization, and Machine Learning. Addison-wesley Publishing Company, 1989.

[7] B.D. Bunday. Basic Optimisation methods. Edward Arnold, 1984.

[8] D.H. Ackley. An empirical study of bit vector function optimization. In L. Davis, editor, Genetic Algorithms and Simulated Annealing. Pitman, 1987.

[9] Fred Glover and Manuel Laguna. Tabus Search. Kluwer Academic Publishers, Norwell Massachusetts, 1998.

[10] A. Wetzel. Evaluation of the effectiveness of genetic algorithms in combinational optimization. University of Pittsburgh, Pittsbirgh (unpublished), 1983.

[11] Nichal Lynn Cramer. A Representation for the Adaptive Generation of Simple Secuencial Programs. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algoritnms and Their Aplications*, pages 183-187. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1985.

[12] David E. Goldberg. Optimal initial population size for binary-coded genetic algorithms Tecnical Report 85001, Tuscaloosa: University of Alabama, The Clearinghouse for Genetics Algorithms, 1985.

[13] Gilbert Syswerda. Schedule Optimization using Genetic Algorithms. In Lawrence Davis, editor, Handbook of genetic Algorithms. Van Nostrand Reinold, New York, 1991.

[14] R. Tanese. Distributed genetic algorithms. *In Proceding of the Third International Conference on Genetic Algorithms*, Schaffer, J.D. (Ed.), Morgan Kaufmann Publishers, San Mateo. *In Schaffer*, J.D. (Ed), San Mateo, CA: Morgan Kaufmann, 1989. Third International Conference on Genetic Algorithms.

[15] John R. Koza. Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press. 1992.

[16] Lashon B. Booker. Intelligent Behavior as an Adaptation to the Task Environment. PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan, 1982.

[17] A. Brindle. Genetic Algorithms for Function Optimization. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta 1981.

[18] James Edward Baker. Adaptive Selection Methods for Genetic Algorithms. In John J. Grefenstette, editor, Proceedings of the First International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, New Jersey, July 1985.

[19] J. L. Gonález Velarde y R. Z. Ríos Mercado. Investigacion de operaciones en acción: Aplicación del TSP en problemas de manufactura y logística. Ingenierías, 2(4):18-23, 1999

[20] Voigt B.F. Der Handlungsreisende, wie er sein soll und was erzu thun hat, um Auftrage zu erhalten und eins gluchlichen Erfolgs in seinen Geschaften gewiss zu sein, Von eimen alten Commis Voyaguer, llmenau, 1831. Y Muller-Merbach. Zweimal travelling Salesman. DGOR-Bulletin 25, 12-13. 1983.

[21] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. y Shmoys, D.B. (ads.) (1985) The Traveling Saleman Problem: A Guided Tour of Combinatorial Optimization, Wiley, Chichester.

[22] Brady, R.M. (1985) optimization strategies glaned frombiological evolution, Nature.

[23] Grefenstette, J., Gopal, R., Rosmaita, B. y Van Gucht, D. (1985) Genetic algorithms for the traveling saleman problem, en Proceedings of the First International Conference on Genetic Algorithms and Their Applications.

[24] Michael R. Garey / Davis S. Johnson: Computers and Intractability A Guide to the theory of NP-Completeness; W.H. Freeman and Co., New York, 1979.

[25] Moscato y F. Tinetti, "'Blending Heuristics with a Population-Based Approach: A Memetic Algorithm for the Traveling Salesman Problem, Diciembre 1992.

[26] Erick Cantú-Paz. A Survey of Parallel Genetic Algorithms. Technical Report 97003, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois, May 1997.

[27] Foster, Ian T. Designing and Building Parallel Programs. Addisson-Wesley, 1995. ISBN 0-201-57594-9

[28] A.K. De Jong. An Analysis of the Behavior of a Class of Genetic Adaptive System. PhD thesis, University of Michigan, 1975.

[29] Frederic Gruau. Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. PhD thesis, Ecole Normale Supérieure de Lyon, Lyon France 1994

[30] Roger Z. Ríos Mercado, Jose Luis González Velarde. Investigación de operaciones en acción, heurísticas para la solución del TSP.

[31] Marco Aurélio Cavalcanti Pacheco. ICA Núcleo de Pesquisa Inteligencia Computacional Aplicada. Departamento de Ingeniería Eléctrica. Pontifícia Universidad de Católica de Río de Janeiro.

[32] David Beasley, Ralph R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. Department of Computing Mathematics, University of Cardiff, cardiff, CF2 4YN, UK. David R. Bull, Department of Electrical and Electronic Engineering, University of Bristol, Bristol, BS8 1TR, UK.

[33] David Beasley, Ralph R. Martin. An Overview of Genetic Algorithms: Part 2, Topics. Department of Computing Mathematics, University of Cardiff, cardiff, CF2 4YN, UK. David R. Bull, Department of Electrical and Electronic Engineering, University of Bristol, Bristol, BS8 1TR, UK.

[34] R. A. Rutenbar. Simulated annealing algorithms: An overview. Dpto. of Electr. & Comput. Eng., Carnegie-Mellon Univ., Pittsburgh, PA; IEEE Circuits and Devices Magazine, Vol. 5, Issue: 1 January 1989.

[35] UEA CALMA Group. Parallelism in combinatorial optimization. Technical Report 2.4, School of Information System. University of Esat Anglia, Norwich, UK, September 1995.

[36] E. Minty, R. Davey, A. Simpson, and D. Hanty. Descomposing the potentially parallel. Technical report, Edinburgh Parallel Computing Centre. The University of Edinburgh, 1998.

[37] Pacheco, Peter S. Parallel programming with MPI / Peter S. Pacheco. cop. 1997.

[38] Al Geist, Vaidy Sunderam. Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing. 1995.

[39] David Levine. Users Guide to the PGAPack Parallel Genetic Algorithm Library. Mathematics and Computer Science Division at Argonne National Laboratory. January 31, 1996.

[40] http://delta.cs.cinvestav.mx/ adiaz/ParProg2001/ParProg.html