UNIVERSIDAD
DE MÁLAGA

LENGUAJES Y
CIENCIAS DE LA
COMPUTACIÓN

TESIS DOCTORAL

Resolución de Problemas Combinatorios con Aplicación Real en Sistemas Distribuidos

Autor
Gabriel Jesús Luque Polo

Director
Enrique Alba Torres

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

21 de Marzo de 2006

D. **Enrique Alba Torres**, Titular de Universidad del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga,

Certifica

que D. **Gabriel Jesús Luque Polo**, Ingeniero en Informática por la Universidad de Málaga, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo su dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulada

Resolución de Problemas Combinatorios con Aplicación Real en Sistemas Distribuidos

Revisado el presente trabajo, estimo que puede ser presentado al tribunal que ha de juzgarlo, y autorizo la presentación de esta Tesis Doctoral en la Universidad de Málaga.

En Málaga, 21 de Marzo de 2006

Firmado: Enrique Alba Torres
Titular de Universidad
Dpto. de Lenguajes y Ciencias de la Computación
Universidad de Málaga

Agradecimientos

Durante estos años son muchas las personas e instituciones que han participado en este trabajo y a quienes quiero expresar mi gratitud por el apoyo y la confianza que me han prestado de forma desinteresada.

En primer lugar un sincero agradecimiento a mi director Enrique Alba, por todo el tiempo que me ha dado, por sus sugerencias e ideas de las que tanto provecho he sacado, por su paciencia ante los problemas que surgieron en esta tesis, por su respaldo y su amistad.

No puedo olvidar a mis compañeros y amigos con los cuales he compartido laboratorio e incontables horas de trabajo. Gracias por los buenos momentos, por aguantarme y por escucharme.

Todo esto nunca hubiera sido posible sin el amparo incondicional de mi familia, mis padres y mis hermanos. Su apoyo, ayuda y ánimo ha sido un elemento clave para lograr realizar este trabajo.

Finalmente debo un especial reconocimiento a la Junta de Andalucía por la confianza que mostraron en mí al concederme una beca FPDI con la cual fue posible aventurarme en esta travesía.

Índice general

Capítulo 1

Introducción

Planeamiento

Uno de los principales frentes de trabajo en el ámbito de la Informática ha sido tradicionalmente el diseño de algoritmos cada vez más eficientes para la solución de problemas tanto de optimización como de búsqueda [95, 219]. La investigación en algoritmos exactos, heurísticos y metaheurísticas para resolver problemas de optimización combinatoria tiene una vigencia inusualmente importante en estos días, ya que nos enfrentamos a nuevos problemas de ingeniería constantemente, al mismo tiempo que disponemos de nuevos recursos computaciones tales como nuevos tipos de máquinas, redes y entornos como Internet. La principal ventaja de la utilización de algoritmos exactos es que garantizan encontrar el óptimo global de cualquier problema, pero tienen el grave inconveniente de que en problemas reales (NP-Completos) su tiempo de ejecución crece de forma exponencial con el tamaño de la instancia. En cambio los heurísticos suelen ser bastante rápidos, pero la calidad de las soluciones encontradas suele ser bastante mediocre, y además son muy complejos de definir para muchos problemas. Las metaheurísticas ofrecen un equilibrio entre ambos extremos; son métodos genéricos que ofrecen una buena solución (incluso en muchos casos el óptimo global) en un tiempo de cómputo moderado.

De esta forma, actualmente es acuciante la necesidad de algoritmos muy eficientes que puedan ofrecer respuesta en tiempo real a problemas del usuario o del sistema, y que permitan trabajar con los nuevos recursos mencionados antes. Dos de las ramas con más éxito para diseñar algoritmos eficientes en la actualidad son la hibridación y el paralelismo [9]. La hibridación permite incorporar información del problema en el algoritmo de resolución para trabajar en contacto con sus características diferenciadoras; esto tiene también relación con la posibilidad de involucrar a varios algoritmos distintos en el proceso de búsqueda de una solución de manera más eficiente [75]. Por otro lado, el uso de múltiples procesadores para resolver en paralelo un problema permite acelerar la búsqueda en tiempo real y alcanzar nuevos dominios de aplicación antes imposibles [12]. El estudio de algoritmos paralelos tanto en red local (LAN) como extensa (WAN) es un campo que necesita de aportaciones metodológicas importantes especialmente en metaheurísticas. Igual de importante resulta el diseño de herramientas teóricas para modelar y estudiar las propuestas más recientes de técnicas.

El uso de Internet para proporcionar servicios, así como la ejecución de algoritmos eficientes utilizando Internet como grupo de redes de computación (*grid computing*) [100] es un campo inexplorado que requiere la definición de algoritmos avanzados, protocolos de comunicación propios, estructuras software adecuadas y estudios de resultados fundamentados. Para aumentar el interés de tal estudio todo ello podría realizarse sobre problemas típicos de la academia y, además, sobre

problemas de corte real que tengan impacto en la industria.

Los estudios en hibridación y paralelismo pueden realizarse en múltiples entornos y familias algorítmicas [95]. Así, tomaremos como hipótesis de trabajo el campo de investigación en algoritmos evolutivos [40, 182], metaheurísticas en general [48] y su hibridación con otras técnicas [75]. Los algoritmos evolutivos representan una plantilla genérica de optimización que da lugar a multitud de distintas técnicas concretas que hoy en día están proporcionando grandes éxitos en optimización combinatoria, ingeniería, economía, telecomunicaciones, bioinformática y otras aplicaciones muy variadas [32].

Nuestro planteamiento se basa en explorar propuestas de algoritmos puros, híbridos y paralelos que mejoren los resultados existentes en dominios reales cuyo sustrato sea un problema combinatorio [109, 182]. Así, diseñar nuevos modelos paralelos, estudiar y caracterizar el comportamiento LAN/WAN, y resolver problemas reales, representa el resumen de objetivos de esta tesis.

Objetivos y Fases

En esta tesis proponemos resolver problemas de optimización usando un sistema de esqueletos (un patrón software [105]) secuenciales y paralelos distribuidos. El objetivo fundamental es comprender el funcionamiento básico de los algoritmos paralelos en estas condiciones y su capacidad diferenciadora para la resolución de problemas, tanto tradicionales como de corte práctico. Este objetivo genérico se puede descomponer en una serie de subobjetivos más concretos:

- Descripción unificada de las metaheurísticas propuestas y sus modelos paralelos.
- Descripción y construcción de modelos paralelos para varias metaheurísticas, entre las que destacan varios algoritmos evolutivos (estrategias evolutivas, algoritmos genéticos y CHC) [40], búsqueda dispersa [115], enfriamiento simulado [145] y varios algoritmo híbridos.
- Formalización de los principales parámetros que gobiernan los modelos paralelos distribuidos y análisis teórico de su influencia en la evolución/convergencia de la metaheurística.
- Implementación de los diferentes modelos siguiendo técnicas orientadas a objetos y análisis de su comportamiento en diferentes plataformas paralelas.
- Aplicación de estos modelos para la resolución de problemas complejos y reales, como el diseño de circuitos, el etiquetado léxico del lenguaje natural, el problema de ensamblado de fragmentos de ADN, problemas de planificación y asignación de trabajadores (WPP) y problemas relacionados con las telecomunicaciones.

Para llevar a cabo esos objetivos hemos seguidos las fases resumidas en la Figura 1.1. Inicialmente se procederá a realizar un estudio sobre el estado del arte actual en el campo de las metaheurísticas, centrándonos principalmente en el dominio de los algoritmos evolutivos paralelos. Una vez comprendidas las ventajas y limitaciones de los modelos propuestos en el pasado, y basadas en ese conocimiento, se realizarán propuestas algorítmicas paralelas como extensiones de los modelos más eficientes en la literatura. Estos modelos serán estudiados en profundidad tanto en un aspecto teórico (estudiando las influencia de los principales parámetros en su convergencia) como en un aspecto experimental, analizando su comportamiento sobre diferentes plataformas paralelas. Basado en el conocimiento adquirido en estas dos últimas fases, se abordará la resolución de diferentes problemas combinatorios con diferentes características pero que comparten como característica diferenciadora su gran dificultad y su aplicación real.

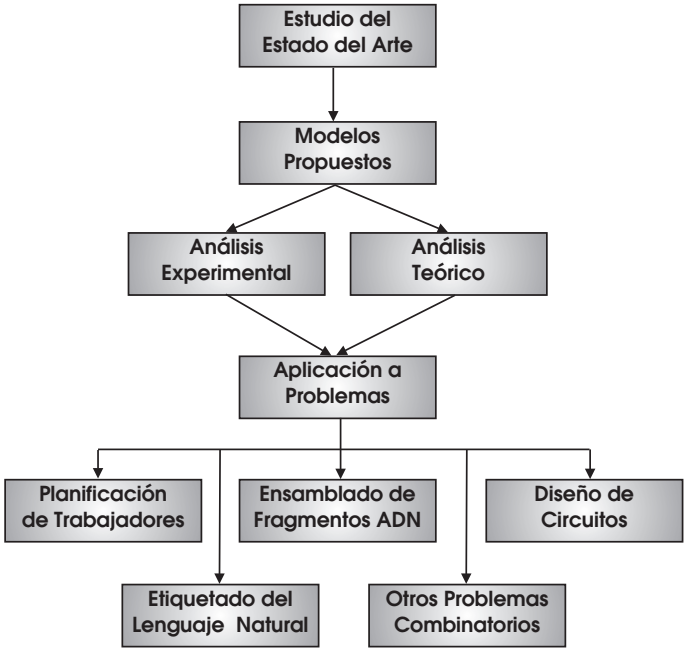


Figura 1.1: Fases seguidas durante la elaboración de esta tesis.

Aportaciones

En este apartado listamos las principales aportaciones de la presenta tesis. De forma esquemática se pueden resumir como sigue:

- Caracterización de las metaheurísticas secuenciales, ofreciendo un modelo matemático unificado para todas ellas y que nos permite establecer las características diferenciadoras de cada una de ellas.
- Estudio y clasificación de los modelos paralelos para las metaheurísticas, centrándonos en especial en el campo de la computación evolutiva [40], una de las familias que más éxito ha dado al aplicarse a resolver problemas de gran complejidad. El modelo unificado que ofrecimos para metaheurísticas secuenciales es extendido para abarcar los diferentes esquemas paralelos existentes y nos sirve para caracterizar sus diferencias y proponer nuevos modelos.
- Establecer una metodología para el diseño experimental, analizando las diferentes métricas aplicables a las metaheurísticas paralelas y los diferentes fases que se deben tener en cuenta y esquematizando los análisis estadísticos que se deben realizar para asegurar la corrección de las conclusiones.
- Incorporación del paradigma orientado a objetos y patrones software (*esqueletos software* [105]) en el diseño e implementación de los modelos propuestos.
- Estudio del comportamiento de varios modelos paralelos en diferentes arquitecturas paralelas, que incluyen varios sistemas de área local, plataformas en área extensa y sistemas grid.
- Estudio de la influencia de la política de migración sobre los algoritmos evolutivos que siguen el modelo distribuido.

- Aplicación de todo el conocimiento adquirido para abordar aplicaciones muy complejas con interés real. Además de probar el correcto funcionamiento de los modelos propuestos y estudiar su comportamiento cuando se enfrentan a aplicaciones de corte real.

Esos puntos esquemáticos que acabamos indicar se concentran en las siguientes acciones:

Se ha realizado un estudio de eficacia/eficiencia de las características de las distintas variantes de estos sistemas distribuidos y una elaboración de conclusiones sobre dicha experiencia. De esta forma, se consideró primordial conocer los algoritmos mencionados, estudiando su comportamiento secuencial, para después extender su ejecución a un entorno distribuido y comprobar las prestaciones relativas para un gran banco de problemas.

Adicionalmente, todo este trabajo tiene un punto más de interés que consiste en hacer este tipo de algoritmos de acuerdo a patrones software que automaticen la incorporación de nuevos algoritmos junto con otros existentes para mejorar la fiabilidad, reutilización, un rápido prototipado de nuevas técnicas y calidad del algoritmo resultante.

También, se han desarrollado estudios teóricos que nos permitan mejorar nuestro conocimiento sobre algoritmos paralelos distribuidos. Entre los aspectos que se desean formalizar se encuentran los efectos sobre el algoritmo de la frecuencia de migración, el ratio de migración y la topología de migración. Además también se estudian modelos matemáticos que nos permitan caracterizar los entornos distribuidos LAN y WAN. Esos conocimientos teóricos se utilizarán para desarrollar versiones paralelas eficientes de varios algoritmos entre los que se encuentran CHC, los algoritmos genéticos, el enfriamiento simulado, la búsqueda dispersa y varios algoritmos híbridos. También se han desarrollado nuevos modelos que recojan las ventajas intrínsecas que se hayan detectado durante el estudio de los anteriores algoritmos y que permita aprovechar las características de los sistemas WAN y grid.

Para probar la eficiencia de los algoritmos paralelos distribuidos se han resuelto varios problemas de gran interés práctico como el diseño de circuitos, el etiquetado léxico del lenguaje natural, el problema de ensamblado de fragmentos de ADN, problemas de planificación y asignación de trabajadores (WPP) y problemas relacionados con las telecomunicaciones. Otra aportación adicional, es ofrecer una guía concreta que se debe seguir a la hora de realizar la fase experimental que produzca una investigación de calidad y correcta.

Organización de la Tesis

Esta tesis está organizada en cuatro bloques principales. En el primero se ofrece una presentación del campo de las metaheurísticas: los métodos que involucran, cómo construirlas, evaluarlas y finalmente presentamos los modelos paralelos existente y los nuevos propuesto. En la segunda parte realizamos un análisis de las metaheurísticas paralelas tratadas, tanto en el aspecto experimental sobre diferentes plataformas, como un análisis teórico sobre el impacto de diferentes parámetros sobre el comportamiento del algoritmo paralelo. En la tercera parte describimos los problemas que hemos abordado y terminamos en la cuarta parte indicando las principales conclusiones que se extraen de esta tesis. A continuación mostramos de forma más detallada el contenido de los capítulos:

- **Parte I: Modelos**

  En el Capítulo 2 ofrecemos una introducción genérica sobre el campo de las metaheurísticas secuenciales. Posteriormente pasamos a describir los modelos paralelos más populares en los diferentes tipos de metaheurísticas, mostrando el estado del arte sobre el aspecto paralelo de cada una de ellas.

El Capítulo 3 tiene dos partes bien diferenciadas pero relacionadas. En la primera describimos la forma de diseñar la implementación de metaheurísticas y las herramientas software existentes para su paralelización. Terminamos esta primera parte presentando MALLBA, que es la biblioteca software en la que se han implementado todos los algoritmos usados en esta tesis. En la segunda parte desarrollamos una metodología de cómo diseñar y analizar los experimentos que utilicen este tipo de métodos.

En el Capítulo 4 presentamos un modelo formal unificado para los diferentes modelos paralelos y proponemos varios tipos de algoritmos y que van a ser analizados y utilizados para resolver los problemas de los siguientes capítulos.

- **Parte II: Análisis y Formalización**

  En el Capítulo 5 se analiza el comportamiento de los algoritmos usados de forma experimental, examinando su comportamiento sobre diferente tipo de plataformas paralelas, desde plataformas de área local hasta sistemas grid, pasando por plataformas de área extensa.

  Como complemento al capítulo anterior en el Capítulo 6 se analizan los algoritmos desde un punto de vista teórico. En este capítulo analizamos la influencia de los principales parámetros de los métodos paralelos, observando como su variación influye en el comportamiento de la técnica paralela. También caracterizamos matemáticamente las observaciones y comprobamos su capacidad de generalización a situaciones no estudiadas.

- **Parte III: Aplicaciones**

  En los capítulos 7-11 mostramos los problemas tratados en la tesis. En el Capítulo 7 se ofrece una introducción sobre los problemas combinatorios y mostramos brevemente los problemas elegidos y las motivaciones por las que han sido elegidos. También se analizará sucintamente los resultados para algunos de ellos, aunque el análisis detallado se realizará en los siguientes capítulos. Después pasaremos a un análisis más completo de los problemas que hemos abordado en detalle. En todos ellos seguimos una misma metodología científica; primero, describimos el problema, posteriormente mostramos cómo se ha abordado el problema, seguidamente analizaremos los resultados ofrecidos y finalizaremos mostrando las principales conclusiones que se pueden extraer de todo este proceso. Los problemas tratados pertenecen a diferentes dominios, pero todos ellos comparten dos características: tienen un gran interés en el ambiente académico o en la industria y tienen una gran dificultad contrastada. Los problemas abarcados son:

  - El ensamblado de fragmentos de ADN (Capítulo 8),
  - el etiquetado léxico del lenguaje natural (Capítulo 9),
  - el diseño de circuitos (Capítulo 10) y
  - la planificación y asignación de trabajadores (Capítulo 11).

  También en las fases previas se han utilizado otros problemas también de interés como son la planificación de las rutas de un conjunto de vehículos, planificación de tareas, diseño de códigos correctores de errores, etc. (Capítulo 5).

- **Parte IV: Conclusiones**

  Terminamos esta tesis dando unas conclusiones sobre todo lo expuesto en el resto del documento, dando especial prioridad a las principales contribuciones que se han conseguido desarrollar. Este apartado está redactado tanto en español como en inglés para cumplir los requisitos exigidos para la obtención de la mención de *Doctorado Europeo* concedido por la Universidad de Málaga.

Completando los capítulos anteriores, se ofrece un apéndice donde se recogen las aportaciones en cuestión de artículos y capítulos de libros que se han realizado durante el desarrollo de esta tesis y una detalla bibliografía de las referencias consultadas para la realización de esta investigación.

Parte I

Modelos

Capítulo 2

Metaheurísticas Secuenciales y Paralelas

La optimización en el sentido de encontrar la mejor solución, o al menos una solución lo suficientemente buena, para un problema, es un campo de vital importancia en la vida real. Constantemente estamos resolviendo pequeños problemas de optimización, como el camino más corto de ir un lugar a otro, la organización de una agenda, etc. En general estos problemas son lo suficientemente pequeños y pueden ser resueltos sin recurrir a elementos externo a nuestro cerebro. Pero conforme estos problemas se hacen más grandes y complejos, el uso de los ordenadores para su resolución es inevitable.

Debido a la gran importancia de estos problemas, a lo largo de la historia de la Informática se han desarrollado múltiples métodos para tratar con este tipo de problemas. Una clasificación muy simple de estos métodos se muestra en la Figura 2.1. Inicialmente, las técnicas las podemos clasificar en exactas (o enumerativas, o exhaustivas, etc.) y técnicas de aproximación. Las técnicas exactas garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado. El inconveniente de estos métodos es que aunque su tiempo es acotado, debido a que la mayoría de los problemas interesantes son NP-Completos, este tiempo es exponencial en el peor caso. Esto provoca en muchos casos que el tiempo necesario para la resolución del problema sea inabordable (cientos de años). Por lo tanto, los algoritmos aproximativos para resolver estos problemas han ido recibiendo un incremento del interés por la comunidad internacional a lo largo de los últimos 30 años. Estos métodos sacrifican la garantía de encontrar el óptimo a cambio de encontrar una "buena" solución en una cantidad razonable de tiempo.

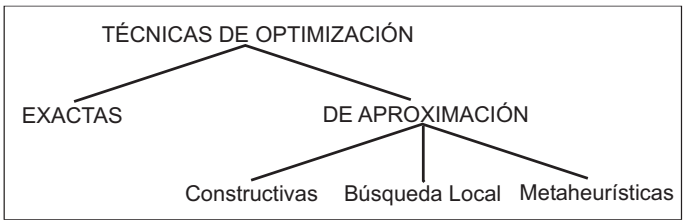


Figura 2.1: Clasificación de las técnicas de optimización.

Dentro de los algoritmo no exactos se pueden encontrar tres tipo: los heurístico constructivos (también llamados voraces), los métodos de búsqueda local (o métodos de seguimiento del gradi-

ente) y las metaheurísticas (en las que nos centramos en este capítulo).

Los heurísticos constructivos suelen ser los métodos más rápidos. Generan una solución partiendo de una vacía a la que se les va añadiendo componentes hasta tener una solución completa, que es el resultado del algoritmo. Aunque en muchos casos encontrar un heurístico constructivo es relativamente fácil, las soluciones ofrecidas son de muy baja calidad, y encontrar métodos de esta clase que produzca buenas soluciones es muy difícil ya que dependen mucho del problema, y para su planteamiento se debe tener un conocimiento muy extenso del mismo. Además en muchos problemas es casi imposible, ya que por ejemplo en aquellos con muchas restricciones puede que la mayoría de las soluciones parciales sólo conduzcan a soluciones no factibles.

Los métodos de búsqueda local o seguimiento del gradiente, parten de una solución ya completa y usando el concepto de vecindario, recorren parte del espacio de búsqueda hasta encontrar un óptimo local. En esa definición han surgido diferentes conceptos, como el de vecindario u óptimo local que ahora pasamos a definir. El vecindario de una solución s, que notamos como $N(s)$, es el conjunto de soluciones que se pueden construir a partir de s aplicando un operador específico de modificación (generalmente denominado *movimiento*). Un óptimo local es una solución tal que su valor de adecuación al problema (*fitness*) es mejor o igual a cualquier solución de su vecindario. Estos métodos partiendo de una solución inicial, examinan su vecindario y se quedan con el mejor vecino y continúan el proceso hasta que encuentran un óptimo local. En muchos casos, la exploración completa del vecindario es inabordable y se siguen diversas estrategias, dando lugar a diferentes variaciones del esquema genérico. Según el operador de *movimiento* elegido, el vecindario cambia y el modo de explorar el espacio de búsqueda también, pudiendo simplificarse o complicarse el proceso de búsqueda.

Finalmente, en los años setenta surgió una nueva clase de algoritmos no exactos, cuya idea básica era combinar diferentes métodos heurísticos a un nivel más alto para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. Estas técnicas se han denominado *metaheurísticas*. Este término fue introducido por primera vez en [114] por Glover. Antes de que este término fuese aceptado por completamente por la comunidad científica, estas técnicas eran denominadas como heurísticos modernos [218]. Esta clase de técnicas metaheurísticas incluyen –pero no están restringida– a colonias de hormigas (ACO), algoritmos evolutivos (EA), búsqueda local iterada (ILS), enfriamiento simulado (SA), y búsqueda tabú (TS). Se pueden encontrar revisiones de metaheurísticas en [12, 48, 116].

De las diferentes descripciones de metaheurísticas que se encuentran en la literatura se pueden encontrar ciertas propiedades fundamentales que caracterizan a este tipo de métodos:

- Las metaheurísticas son estrategias o plantillas generales que "guían" el proceso de búsqueda.
- El objetivo es una exploración del espacio de búsqueda eficiente para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son algoritmos no exactos y generalmente son no deterministas.
- Pueden incorporar mecanismos para evitar las áreas del espacio de búsqueda no óptimas.
- El esquema básico de cualquier metaheurística es general y no depende del problema a resolver.
- Las metaheurísticas hacen uso de conocimiento del problema que se trata resolver en forma de heurísticos específicos que son controlados por una estrategia de más alto nivel.

Resumiendo esos puntos, se puede acordar que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda. En otras palabras, una

metaheurística es una plantilla general no determinista que debe ser rellenada con datos específicos del problema (representación de las soluciones, operadores para manipularlas, etc.) y que permiten abordar problemas con espacios de búsqueda de gran tamaño (por ejemplo, 2^{1000}). Por lo tanto es de especial interés el correcto equilibrio (generalmente dinámico) que haya entre diversificación (o exploración) y intensificación (o explotación). El termino diversificación se refiere a la exploración del espacio de búsqueda, mientras que intensificación se refiere a la explotación de algún área concreta de ese espacio. El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente la regiones prometedoras del espacio de búsqueda global y por el otro lado no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Dentro de las metaheurísticas podemos distinguir dos tipos de estrategia de búsqueda. Por un lado tenemos, las extensiones "inteligentes" de los métodos de búsqueda local. La meta de estas estrategias es evitar de alguna forma los mínimos locales y moverse a otras regiones prometedoras del espacio de búsqueda. Este tipo de estrategia es el seguido por la búsqueda tabú, la búsqueda local iterada, la búsqueda con vecindario variable y el enfriamiento simulado. Estas metaheurísticas (llamadas basadas en trayectoria) trabajan sobre una o varias estructuras de vecindario impuestas por el espacio de búsqueda. Otro tipo de estrategia es la seguida por las colonias de hormigas o los algoritmos evolutivos. Éstos incorporan un componente de aprendizaje en el sentido de que, de forma implícita o explícita, intentan aprender la correlación entre las variables del problema para identificar las áreas correctas en el espacio de búsqueda. Estos métodos realizan, en este sentido, un muestreo sesgado del espacio de búsqueda.

A continuación ofrecemos una definición formal de los elementos incluidos en una metaheurísticas y que dependiendo de como la instanciemos produce un mecanismo u otro.

DEFINICIÓN **1** *Una metaheurística $\mathcal{M}$ esta caracterizada por una tupla formada por los siguientes componentes:*

$$\mathcal{M} = \langle \Theta, \Phi, \sigma, \mu, \lambda, \Xi, \tau \rangle$$

donde:

- $\Theta = \{\theta_1, \ldots, \theta_\mu\}$ *es el conjunto de estructuras con las que trabaja $\mathcal{M}$. Cada $\theta_i \in \mathcal{T}$, y $\mathcal{T}$ es el conjunto de todas las posibles soluciones al problema.*
- $\Phi : \mathcal{T}^\mu \times \Xi \rightarrow \mathcal{T}^\lambda$ *es el operador que modifica las estructuras de $\mathcal{S}$.*
- $\sigma : \mathcal{T}^{\mu+\lambda} \times \Xi \rightarrow \mathcal{T}^\mu$ *es una función que permite seleccionar las nuevas estructuras a ser tratadas en la siguiente iteración de $\mathcal{M}$.*
- μ *es el número de estructuras con las que trabaja $\mathcal{M}$.*
- λ *es el número de nuevas estructuras generadas de la aplicación de Φ a Θ en cada iteración de $\mathcal{M}$.*
- $\Xi = \{\xi_1, \xi_2, \ldots\}$ *es un conjunto de variables de estado que contienen información relativa a la evolución de $\mathcal{M}$. Cualquier $\mathcal{M}$ al menos contendrá un elemento $\xi_1 = \theta^*$ que representa la mejor estructura encontrada.*
- $\tau : \Theta \times \Xi \rightarrow \{true, false\}$ *es una función que decide la terminación del algoritmo.*

DEFINICIÓN **2** *Según la Definición 1 la dinámica de funcionamiento de cualquier metaheurística es la siguiente:*

$$\Theta_{i+1} = \begin{cases} \{\theta^*\}^\mu & si\ \tau(\Theta_i \times \Xi) \\ \sigma(\Phi(\Theta_i \times \Xi)) & en\ otro\ caso. \end{cases}$$

En las próximas secciones estudiaremos cada una de las técnicas concretas incluidas en este campo y los modelos paralelos existentes, terminando con un breve estado del arte sobre el paralelismo de cada una de ellas.

2.1. Clasificación de las Metaheurísticas

Hay diferentes formas de clasificar y describir las técnicas metaheurísticas [56, 72]. Dependiendo de las características que se seleccionen se pueden obtener diferentes taxonomías: basadas en la naturaleza y no basadas en la naturaleza, basadas en memoria o sin memoria, con función objetivo estática o dinámica, etc. En esta tesis hemos elegido clasificarlas de acuerdo a si en cada paso manipulan un único punto del espacio de búsqueda o trabajan sobre un conjunto (población) de ellos, es decir, esta clasificación divide a las metaheurísticas en basadas en trayectoria y basadas en población. Esta taxonomía se muestra de forma gráfica en la Figura 2.2. Elegimos esta clasificación porque es ampliamente utilizada en la comunidad científica, además de ser muy coherente.

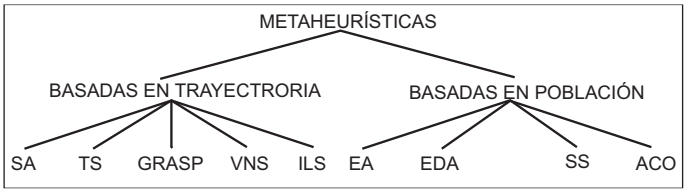


Figura 2.2: Clasificación de las metaheurísticas.

2.1.1. Basadas en Trayectoria

En esta sección resumiremos brevemente las metaheurísticas clasificados como basadas en trayectoria. La principal característica de estos métodos es que parten de un punto y mediante exploración del vecindario van actualizando la punto actual, formando una trayectoria. Según la notación de la Definición 1, la principal característica es que $\mu = 1$. La mayoría de estos algoritmos surgen como extensiones de los métodos de búsqueda local simples a los que se les añade alguna característica para escapar de los mínimos locales. Esto implica la necesidad de una condición de parada más compleja que la de encontrar un mínimo local. Normalmente se termina la búsqueda cuando se alcanza un número máximo predefinido de iteraciones, una solución con una calidad aceptable, o cuando se detecta un estancamiento del proceso.

Enfriamiento Simulado (SA)

El Enfriamiento Simulado o *Simulated Annealing* (SA) es uno de los más antiguos entre las metaheurísticas y seguramente es el primer algoritmo con una estrategia explícita para escapar de los mínimos locales. Los orígenes del algoritmo se encuentran en un mecanismo estadístico, denominado Metropolis [181]. La idea del SA es simular el proceso de recocido del metal y del cristal. El SA fue inicialmente presentado en [145]. Para evitar quedar atrapado en un mínimo local, el algoritmo permite elegir una solución cuyo valor de fitness sea peor que el de la solución

actual. En cada iteración mediante algún tipo de método elige una solución vecina de la solución actual $s' \in N(s)$. Si s' es mejor que s (es decir, tiene un mejor valor en la función de fitness), se sustituye s por s' como solución actual. Si la solución s' es peor, entonces es aceptada con una determinada probabilidad que depende de la temperatura actual T y de la variación en la función de fitness, $f(s') - f(s)$ (caso de minimización). Esta probabilidad generalmente se calcula siguiendo la distribución de Boltzmann:

$$p(s'|T, s) = e^{-\frac{f(s')-f(s)}{T}}. \tag{2.1}$$

El pseudo-código de este método se muestra en el Algoritmo 1. A continuación describimos brevemente los principales componentes de este algoritmo.

Algoritmo 1 Enfriamiento Simulado.

```
s ← GenerarSolucionInicial()
k ← 0
T ← TemperaturaInicial()
while no se alcance la condición de parada do
  s' ← ElegirVecino(N(s))
  if f(s') < f(s) then
    s ← s' {s' remplaza s}
  else
    Aceptar s' con probabilidad p(s'|T, s) {Ec. 2.1}
  end if
  if k mod MCL == 0 then
    ActualizarTemperatura(T)
  end if
  k ← k + 1
end while
Salida: la mejor solución encontrada
```

GenerarSolucionInicial(): El algoritmo empieza generando una solución que puede ser aleatoria o construida heurísticamente.

TemperaturaInicial(): Se elige una temperatura inicial, generalmente alta para que inicialmente se permita una mayor diversificación y al disminuirla a lo largo del algoritmo, el balance vaya cambiando y haciendo mayor intensificación.

MCL: Con este parámetro (*Markov Chain Length*) se hace referencia a cada cuantas iteraciones, el algoritmo actualiza la temperatura.

ActualizarTemperatura(T): Este proceso procede a actualizar la temperatura de acuerdo a un esquema de enfriamiento. El esquema de enfriamiento es un elemento crucial de este algoritmo. Uno de los esquemas más utilizados sigue la ley geométrica: $T \leftarrow \alpha \cdot T$, donde $\alpha \in (0, 1)$, lo que corresponde con una caída exponencial de la temperatura. Como se ha comentado este proceso de enfriamiento sirve como balance entre la diversificación e intensificación y es muy importante para el correcto funcionamiento del método. Otros esquemas son [267]:

$$CSA : T_i = \frac{T_0}{\ln(1+i)}. \tag{2.2}$$

$$FSA : T_i = \frac{T_0}{1+i}. \tag{2.3}$$

$$VFSA : T_i = \frac{T_0}{e^i}. \tag{2.4}$$

Siguiendo la notación que definimos al inicio del capítulo, esta metaheurística se puede caracterizar como (los parámetros no especificados no están fijados por el método y dependen de la implementación y del problema):

$$\mathcal{SA} = \langle \Theta_{SA}, \Phi_{SA}, \sigma_{SA}, \mu_{SA}, \lambda_{SA}, \Xi_{SA}, \tau_{SA} \rangle$$

donde:

- $\mu_{SA} = 1$.
- $\lambda_{SA} = 1$.
- $\Phi_{SA}(\theta) = ElegirVecino(N(\theta))$, donde:
  - $ElegirVecino : 2^{\mathcal{T}} \rightarrow \mathcal{T}$ es una función que elige una estructura de entre un conjunto de ellas.
  - $N : \mathcal{T} \rightarrow 2^{\mathcal{T}}$ es la función de vecindario, que relaciona una estructura con un subconjunto de las estructuras vecinas.
- $\Xi_{SA} = \{\theta^*, T\}$.
- $\sigma_{SA}(\theta, \theta', T) = \begin{cases} \{\theta'\} & si\ f(\theta') < f(\theta)\ o\ random() < p(\theta'|T, \theta) \\ \{\theta\} & en\ otro\ caso. \end{cases}$

Búsqueda Tabú (TS)

La Búsqueda Tabú o *Tabu Search* (TS) es una de las metaheurísticas que se se han aplicado con más éxito a la hora de resolver problemas de optimización combinatorias. Los fundamentos de este método fueron introducidos en [114], y están basados en las ideas formuladas en [113]. Un buen resumen de esta técnica y sus componentes se puede encontrar en [118]. La idea básica de la búsqueda tabú es el uso explícito de un historial de la búsqueda (una memoria de corto plazo), tanto para escapar de los mínimos locales como para implementar su estrategia de exploración y evitar buscar varias veces en la misma región. Esta memoria de corto plazo es implementada en esta técnica como una lista tabú (TL) donde se mantienen las soluciones visitadas más recientemente, para excluirlas de los próximos movimientos. Cada iteración, se elige la mejor solución entre las permitidas y la solución es añadida a la lista tabú.

Desde el punto de vista de la implementación, en general mantener una lista de las soluciones completas no es práctico debido a su ineficiencia. Por lo tanto, en general se suelen almacenar los movimientos que nos ha llevado a generar esa solución o los componentes principales que define esa solución. En cualquier caso, los elementos que esta lista nos permite filtrar el vecindario, generando un vecindario reducido de soluciones elegibles $N_a(s)$. El almacenamiento de los movimientos en vez de las soluciones completas, es bastante más eficiente, pero introduce una perdida de información. Para evitar este problema, se define un criterio de aspiración que permite incluir una solución en vecindario de elegibles incluso si está prohibida debido a la lista tabú. El criterio de aspiración más ampliamente usado es permitir soluciones cuyo fitness sea mejor que el mejor encontrado hasta el momento. El pseudo-código de este método se muestra en el Algoritmo 2.

Algoritmo 2 Búsqueda Tabú.

$s \leftarrow GenerarSolucionInicial()$
Inicializar la lista tabú: TL
while no se alcance la condición de parada **do**
  $N_a(s) \leftarrow \{s' \in N(s) | s'$ no viola la condición tabú, o satisface al menos una condición de aspiración $\}$
  $s' \leftarrow argmin\{f(s'') | s'' \in N_a(s)\}$ {Caso de minimización}
  Actualizar las listas tabú TL con s y s'
  $s \leftarrow s'$ {s' reemplaza a s}
end while
Salida: la mejor solución encontrada

El uso de la lista tabú previene volver a visitar soluciones que han sido examinadas recientemente; además, evita los ciclos infinitos dentro de la búsqueda y permite que durante el proceso se acepten soluciones que empeoren el fitness, para evitar los mínimos locales. La longitud de la la lista tabú es un parámetro muy importante en este algoritmo. Con tamaños pequeños, la búsqueda se concentrará en pequeñas áreas del espacio de búsqueda. Por el contrario, una lista grande fuerza al proceso a explorar grandes regiones, debido a que impide revisitar un alto número de soluciones. Esta longitud puede variar a lo largo de la búsqueda, lo que suele llevar a un algoritmo más robusto aunque más complejo.

Según la notación descrita en la Definición 1 una búsqueda tabú se puede caracterizar como (los parámetros no especificados no están fijados por el método y dependen de la implementación y del problema):

$$\mathcal{TS} = \langle \Theta_{TS}, \Phi_{TS}, \sigma_{TS}, \mu_{TS}, \lambda_{TS}, \Xi_{TS}, \tau_{TS} \rangle$$

donde:

- $\mu_{TS} = 1$.
- $\Xi_{TS} = \{\phi^*, TL\}$.
- $\Phi_{TS}(\theta) = Aspiracion(CondicionTabu(N(\theta)) \cup N(\theta))$, donde:
  - $Aspiracion : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}}$ es una función que elige las estructuras que cumplen los criterios de aspiración de entre un conjunto de ellas.
  - $CondicionTabu : 2^{\mathcal{T}} \times TL \rightarrow 2^{\mathcal{T}}$ es una función que elige las estructuras que no viola la lista tabu de entre un conjunto de ellas.
  - $N : \mathcal{T} \rightarrow 2^{\mathcal{T}}$ es la función de vecindario, que relaciona una estructura con un subconjunto de las estructuras vecinas.
- $\sigma_{TS}(\{\theta_1, \ldots, \theta_\lambda\}) = argmin\{f(\theta') | \theta' \in \{\theta_1, \ldots, \theta_\lambda\}\}$.

GRASP

El Procedimiento de Búsqueda Miope Aleatorizado y Adaptativo o *The Greedy Randomized Adaptive Search Procedure* (GRASP) [90] es una metaheurística simple que combina heurísticos constructivos con búsqueda local. La estructura está esquematizada en el Algoritmo 3. GRASP es un procedimiento iterativo, compuesto de dos fases: primero una construcción de una solución y posterior proceso de mejora. La solución mejorada es el resultado del proceso de búsqueda.

Algoritmo 3 GRASP.

while no se alcance la condición de parada **do**
  $s \leftarrow GenerarSolucion()$ {véase Algoritmo 4}
  $AplicarBusquedaLocal(s)$
end while
Salida: la mejor solución encontrada

El mecanismo de construcción de soluciones (Algoritmo 4) es un heurístico constructivo aleatorizado. Va añadiendo paso a paso diferentes componentes c a la solución parcial s^p, que inicialmente está vacía. Los componentes que se añaden en cada paso son elegidos aleatoriamente de una lista restringida de candidatos (RCL). Esta lista es un conjunto de $N(s^p)$ (el conjunto de componentes permitidos). Para generar esta lista, los componentes de la solución en $N(s^p)$ son ordenados de acuerdo alguna función dependiente del problema (η). La lista RCL está compuesta por los α mejores componentes de ese conjunto. En el caso extremo de $\alpha = 1$, siempre se añade el mejor componente encontrado de manera determinista, con lo que el método de construcción es equivalente a un algoritmo voraz. En el otro extremo con $\alpha = |N(s^p)|$, tenemos que el componente a añadir se elige totalmente aleatoriamente entre todos los disponibles. Por lo tanto, α es un parámetro clave que influye en cómo se va a muestrear el espacio de búsqueda. En [207] se presentan los esquemas más importantes para definir α. El esquema más simple es mantener α constante a lo largo de la búsqueda. Otros más elaborados, modifican este parámetro a lo largo de búsqueda.

Algoritmo 4 GRASP: Método de Generación de Soluciones.

$s^p = \emptyset$
$\alpha \leftarrow DeterminarLongitudRCL()$
while $N(s^p) \neq \emptyset$ **do**
  $RCL \leftarrow GenerarRCL(\eta, N(s^p), \alpha)$
  $c \leftarrow ElegirAleatoriamente(RCL)$
  $s^p \leftarrow$ extender s^p con el componente c
end while

La segunda fase del algoritmo consiste en aplicar un algoritmo de búsqueda local para mejorar la solución generada en el paso anterior. Este mecanismo de mejora, ya puede ser una técnica de mejora simple o algoritmo más complejos como SA o TS.

Según la notación de la Definición 1, GRASP se puede caracterizar como (los parámetros no especificados no están fijados por el método y dependen de la implementación y del problema):

$$\mathcal{GRASP} = \langle \Theta_{GRASP}, \Phi_{GRASP}, \sigma_{GRASP}, \mu_{GRASP}, \lambda_{GRASP}, \Xi_{GRASP}, \tau_{GRASP} \rangle$$

donde:

- $\mu_{GRASP} = 1$.
- $\lambda_{GRASP} = 1$.
- $\Xi_{GRASP} = \{\phi^*\}$.
- $\Phi_{GRASP}(\theta) = BusquedaLocal(GenerarSolucion())$, donde:
  - $BusquedaLocal : \emptyset\mathcal{T} \rightarrow \mathcal{T}$ es una función que a partir de una estructura genera otra.

- $GenerarSolucion :\rightarrow \mathcal{T}$ es una función que genera una estructura.

- $\sigma_{TS}(\theta, \theta') = \theta'$.

Búsqueda con Vecindario Variable (VNS)

La Búsqueda con Vecindario Variable o *Variable Neighborhood Search* (VNS) es una metaheurística propuesta en [190], que aplica explícitamente una estrategia para cambiar entre diferentes estructuras de vecindario de entre un conjunto de ellas definidas al inicio del algoritmo. Este algoritmo es muy general y con muchos grados de libertad a la hora de diseñar variaciones e instanciaciones particulares.

Algoritmo 5 Búsqueda con Vecindario Variable.

```
Seleccionar un conjunto de estructuras de vecindario: N_k, k = 1, ..., k_max
s ← GenerarSolucionInicial()
while no se alcance la condición de parada do
  k ← 1
  while k < k_max do
    s' ← ElegirAleatoriamente(N_k(s))
    s'' ← BusquedaLocal(s')
    if f(s'') < f(s) then
      s ← s''
      k ← 1
    else
      k ← k + 1
    end if
  end while
end while
Salida: la mejor solución encontrada
```

En el Algoritmo 5 se muestra el esquema básico del método. El primer paso a realizar es definir un conjunto de vecindarios. Esta elección puede hacerse de muchas formas: desde ser elegidos aleatoriamente hasta utilizando complejas ecuaciones deducidas del problema. Cada iteración consiste en tres fases: elección del candidato, una fase de mejora y finalmente el movimiento. En la primera fase, se elige aleatoriamente un vecino s' de s usando el k-ésimo vecindario. Esta solución s' es utilizada como punto de partida de la búsqueda local de la segunda fase. Cuando termina el proceso de mejora, se compara la nueva solución s'' con la original s. Si es mejor, pues s'' se convierte en la solución actual y se inicializa el contador de vecindarios ($k \leftarrow 1$). Si no es mejor, se repite el proceso pero utilizando el siguiente vecindario ($k \leftarrow k + 1$). Mientras que la búsqueda local es el paso de intensificación del método, el cambio de vecindario puede considerarse como el paso de diversificación.

Siguiendo la notación que definimos al inicio del capítulo, esta metaheurística se puede caracterizar como (los parámetros no especificados no están fijados por el método y dependen de la implementación y del problema):

$$\mathcal{VNS} = \langle \Theta_{VNS}, \Phi_{VNS}, \sigma_{VNS}, \mu_{VNS}, \lambda_{VNS}, \Xi_{VNS}, \tau_{VNS} \rangle$$

donde:

- $\mu_{VNS} = 1$.
- $\lambda_{VNS} = 1$.
- $\Xi_{VNS} = \{\theta^*, k\}$.
- $\Phi_{VNS}(\theta) = BusquedaLocal(ElegirVecino(N(\theta, k)))$, donde:
  - $BusquedaLocal : \mathcal{T} \to \mathcal{T}$ es una función que a partir de una estructura genera otra.
  - $ElegirVecino : 2^{\mathcal{T}} \to \mathcal{T}$ es una función que elige una estructura de entre un conjunto de ellas.
  - $N : \mathcal{T} \times int \to 2^{\mathcal{T}}$ es la función de vecindario, que relaciona una estructura con un subconjunto de todas las estructuras y dependiendo del entero se eligen vecindarios diferentes.
- $\sigma_{VNS}(\theta, \theta', T) = \begin{cases} \{\theta'\} & si\ f(\theta') < f(\theta) \\ \{\theta\} & en\ otro\ caso. \end{cases}$

Búsqueda Local Iterada (ILS)

La Búsqueda Local Iterada o *Iterated Local Search* (ILS) [130, 246] es una metaheurística basada en un concepto simple pero muy efectivo. En cada iteración, la solución actual es perturbada y a esta nueva solución se le aplica un método de búsqueda local para mejorarla. Este nuevo mínimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa el test de aceptación. El esquema básico del método es mostrado en el Algoritmo 6.

Algoritmo 6 Búsqueda Local Iterada.

$s \leftarrow GenerarSolucionInicial()$
$s' \leftarrow BusquedaLocal(s)$
while no se alcance la condición de parada **do**
  $s'' \leftarrow Perturbacion(s', historia)$
  $s'' \leftarrow BusquedaLocal(s')$
  $s' \leftarrow AplicarCriterioAceptacion(s', s'', historia)$
end while
Salida: la mejor solución encontrada

La importancia del proceso de perturbación es obvia: si es demasiada pequeña puede que el sistema no sea capaz de escapar del mínimo local, por otro lado, si es demasiado grande, la permutación puede hacer que el algoritmo sea como un método de búsqueda local con un reinicio aleatorio. Por lo tanto, el método de perturbación debe generar un nuevo punto que sirva como inicio a la búsqueda local, pero que no debe estar muy lejos del actual para que no sea una solución aleatoria.

El criterio de aceptación actúa como contra-balance, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de la búsqueda y las características del nuevo mínimo local.

Como se dijo anteriormente, el diseño de un algoritmo que siga el esquema del ILS tiene varios grados de libertad: la generación de la solución inicial, el método de perturbación, y el criterio de aceptación. Además la historia de las soluciones visitadas por el algoritmo pueden ser explotadas a corto y largo plazo.

Siguiendo la notación que definimos al inicio del capítulo, esta metaheurística se puede caracterizar como:

$$\mathcal{ILS} = \langle \Theta_{ILS}, \Phi_{ILS}, \sigma_{ILS}, \mu_{ILS}, \lambda_{ILS}, \Xi_{ILS}, \tau_{ILS} \rangle$$

donde:

- $\mu_{ILS} = 1$.
- $\lambda_{ILS} = 1$.
- $\Xi_{ILS} = \{\theta^*, H\}$.
- $\Phi_{ILS}(\theta) = BusquedaLocal(Perturbar(\theta, H))$, donde:
  - $BusquedaLocal : \mathcal{T} \to \mathcal{T}$ es una función que a partir de una estructura genera otra.
  - $Perturbar : \mathcal{T} \times H \to \mathcal{T}$ es una función que elige una estructura de entre un conjunto de ellas.
- $\sigma_{ILS}(\theta, \theta', T) = AplicarCriterioAceptacion(\theta, \theta', H)$, donde:
  - $AplicarCriterioAceptacion : \mathcal{T} \times \mathcal{T} \times H \to \mathcal{T}$ es una función que a partir de dos estructuras y otra información histórica selecciona una de las estructuras.

Esta metaheurística es muy genérica y la mayoría de los métodos que utilizan son abiertos, dependiendo del problema, por eso esta descripción formal apenas define un par de características.

2.1.2. Basadas en Población

Los métodos basados en población se caracterizan por trabajar con un conjunto de soluciones (población) en cada iteración, a diferencia con los métodos que vimos antes que únicamente utilizan un punto del espacio de búsqueda por iteración (es decir, generalmente $\mu \neq 1$ y/o $\lambda \neq 1$). El resultado final proporcionado por este tipo de algoritmos depende fuertemente de la forma en que manipulan la población. Los algoritmos basados en población más conocidos son los algoritmos evolutivos (EA) y las colonias de hormigas (ACO). En los métodos que siguen el esquema de los algoritmos evolutivos, la modificación de a población se lleva a cabo mediante tres operadores: selección, recombinación y mutación, y en los sistemas de colonias de hormigas para construir las nuevas soluciones se usa los rastros de feromonas y alguna información heurística.

Algoritmos Evolutivos (EA)

Los Algoritmos Evolutivos (EA) están inspirados en la capacidad de la naturaleza para evolucionar seres para adaptarlos a los cambios de su entorno. Esta familia de técnicas siguen un proceso iterativo y estocástico que opera sobre un conjunto de individuos (población). Cada individuo representa una solución potencial al problema que se está resolviendo. Inicialmente, la población es generada aleatoriamente (quizás con ayuda de un heurístico de construcción). Cada individuo en la población tiene asignado, por medio de una función de aptitud (fitness), una medida de su bondad con respecto al problema bajo consideración. Este valor es la información cuantitativa que el algoritmo usa para guiar su búsqueda. El proceso completo está esbozado en el Algoritmo 7.

Algoritmo 7 Algoritmos Evolutivos.

$P \leftarrow GenerarPoblacionInicial()$
$Evaluar(P)$
while no se alcance la condición de parada **do**
  $P' \leftarrow SeleccionarPadres(P)$
  $P' \leftarrow Recombinar(P')$
  $P' \leftarrow Mutar(P')$
  $Evaluar(P')$
  $P \leftarrow SelecionarNuevaPoblacion(P, P')$
end while
Salida: la mejor solución encontrada

Puede verse que el algoritmo comprende las tres fases principales: selección, reproducción y reemplazo. El proceso completo es repetido hasta que se cumpla un cierto criterio de terminación (normalmente después de un número dado de iteraciones).

- **Selección:** Partiendo de la población inicial P de μ individuos, durante esta fase, se crea una nueva población temporal (P') de λ individuos. Generalmente los individuos más aptos (aquellos correspondientes a las mejores soluciones contenidas en la población) tienen un mayor número de instancias que aquellos que tienen menos aptitud (selección natural). De acuerdo con los valores de μ y λ podemos definir distintos esquemas de selección (Figura 2.3):

  1. **Selección por Estado Estacionario.** Cuando $\lambda = 1$ tenemos una selección por estado estacionario (*steady-state*) en la que únicamente se genera un hijo en cada paso de la evolución.
  2. **Selección Generacional.** Cuando $\lambda = \mu$ tenemos una selección por generaciones en la que genera una nueva población completa de individuos en cada paso.
  3. **Selección Ajustable.** Cuando $1 \leq \lambda \leq \mu$ tenemos una selección intermedia en la que se calcula un número ajustable (*generation gap*) de individuos en cada paso de la evolución. Las anteriores son casos particulares de ésta.
  4. **Selección por Exceso.** Cuando $\lambda > \mu$ tenemos una selección por exceso típica de los procesos naturales reales.

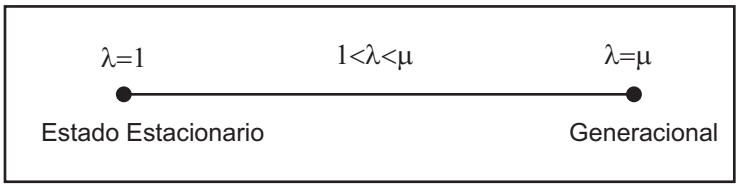


Figura 2.3: Diferencia entre los diversos esquemas de selección.

- **Reproducción:** En esta fase se aplican los operadores reproductivos a los individuos de la población P' para producir una nueva población. Típicamente, esos operadores se corresponden con la recombinación de parejas y con la mutación de los nuevos individuos generados. Estos operadores de variación son, en general, no deterministas, es decir, no siempre se tienen

que aplicar a todos los individuos y en todas las generaciones del algoritmo, sino que su comportamiento viene determinado por su probabilidad asociada.

- **Reemplazo:** Finalmente, los individuos de la población original son sustituidos por los individuos recién creados. Este reemplazo usualmente intenta mantener los mejores individuos borrando los peores. Dependiendo si para realizar el reemplazo se tienen en cuenta la antigua población P podemos obtener 2 tipos de estrategia de reemplazo:

  1. (μ, λ) si es reemplazo se realiza utilizando únicamente los individuos de la nueva población P'. Se debe cumplir que $\mu \leq \lambda$
  2. $(\mu + \lambda)$ si es reemplazo se realiza seleccionando μ individuos de la unión de P y P'.

Estos algoritmos establecen una equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basados sobre el hecho que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente la existentes.

Según la Definición 1, la principal diferencia con otros algoritmos es Φ_{EA} que esta formada por la composición de tres funciones:

$$\Phi_{EA}(\Theta) = \{\omega_m(\theta')|\theta' \in \omega_r(\omega_s(\Theta))\}$$

donde:

- $\omega_s : \mathcal{T}^\mu \rightarrow \mathcal{T}^\lambda$, se corresponde con la función de selección de los padres.
- $\omega_r : \mathcal{T}^\lambda \rightarrow \mathcal{T}^\lambda$, se corresponde con el operador de recombinación.
- $\omega_m : \mathcal{T} \rightarrow \mathcal{T}$, se corresponde con el operador de mutación.

Finalmente para completar la caracterización se define: $\sigma_{EA} : \mathcal{T}^{\mu+\lambda} \rightarrow \mathcal{T}^\mu$ que se corresponde a la función que se encarga de crear la nueva población a partir de la antigua.

En la literatura se han propuesto diferentes algoritmos basados en este esquema general. Básicamente, estas propuestas se pueden clasificar en tres categorías que fueron desarrolladas de forma independiente. Estas categorías son la Programación Evolutiva o *Evolutionary Programming* (EP) desarrollada por Fogel [96], las Estrategias Evolutivas o *Evolution Strategies* (ES) propuestas por Rechenberg en [217], y los Algoritmos Genéticos o *Genetic Algorithms* (GA) introducidos por Holland en [133]. La programación evolutiva surgió originalmente para operar con representaciones discretas de máquinas de estados finitos, aunque en la actualidad hay muchas variantes que son usadas para problemas de optimización continua. Las estrategias evolutivas están desarrolladas para la optimización continua, normalmente usan algún tipo de selección elitista (donde los mejores individuos son conservados a lo largo del procesos completo) y una mutación específica (la recombinación no suele usarse en este tipo de métodos). En ES, los individuos están formados por las variables propias de la solución y también por otros parámetros usados para guiar la búsqueda, permitiendo así, una auto-adaptación de estos parámetros. Finalmente los algoritmos genéticos, son los EAs más populares. Tradicionalmente usaban una representación binaria, aunque en la actualidad es fácil encontrar GAs que usan otro tipo de representación. En la literatura se han presentado bastantes resúmenes y clasificaciones donde se explican con un mayor detalle estos algoritmos, por ejemplo, un repaso bastante interesante de estos algoritmos fue realizada por Bäck [244].

Como ejemplo de particularización de este esquema general vamos a describir en más detalle el CHC [89]. Hemos elegido este método debido a que va a ser ampliamente utilizado en el resto de esta tesis, además de ser bastante más desconocido ya que es más reciente. El método está descrito en el Algoritmo 8.

Algoritmo 8 CHC.

$P \leftarrow GenerarPoblacionInicial()$
$Evaluar(P)$
while no se alcance la condición de parada **do**
  $P' \leftarrow SeleccionarPadres(P)$
  $P'' \leftarrow \emptyset$
  for all $p_1, p_2 \in P'$ **do**
    if p_1 y p_2 superan la condición de incesto **then**
      $P'' \leftarrow P'' \cup \{HUX(p_1, p_2)\}$
    end if
  end for
  $Evaluar(P'')$
  $P \leftarrow SelecionarMejores(P, P'')$
  if $Convergencia(P)$ **then**
    $Reiniciar(P)$
  end if
end while
Salida: la mejor solución encontrada

Ahora describiremos las características más importantes de este algoritmo y en que aspecto se diferencia del esquema básico de los EAs. El CHC fija la selección a utilizar tanto en la propia fase de selección (donde todos los elementos de la población son elegidos) como en la fase de reemplazo (que es elitista). A diferencia de los EAs tradicionales que tienen una selección orientada principalmente al proceso de reproducción, en los CHCs está orientada a la supervivencia de los mejores. Esto se plasma en la utilización de una selección elitista en la fase de reemplazo, donde se seleccionan los mejores individuos de entre la población actual y lo población auxiliar generada en el proceso reproductivo. Debido al tipo de selección introducida, hay el riesgo de que se produzca una convergencia rápida hacia una solución no óptima, con lo que hay que introducir elementos que mantengan la diversidad genética. Uno de esos elementos es que se impide que se crucen individuos similares entre sí (*incest prevention*). El CHC utiliza como operador de recombinación una variante del cruce uniforme (el HUX), que es una forma altamente destructiva de cruce. Otra característica en la que se diferencia de los EAs tradicionales, es la ausencia de operador de mutación en la fase de reproducción. Pese a no usarse la mutación en esa fase, el operador en sí si existe y es aplicado para reiniciar la población parcialmente cuando se ha detecte una convergencia prematura de la población.

Algoritmos de Estimación de la Distribución (EDA)

Los Algoritmos de Estimación de la Distribución o *Estimation Distribution Algorithm* (EDA) [192] muestra un comportamiento similar a los algoritmos evolutivos presentados en la sección anterior y de hecho muchos autores consideran los EDAs como otro tipo de EA. Su esquema está presentada en Algoritmo 9. Los EDAs operan sobre una población de soluciones tentativas como los algoritmos evolutivos pero a diferencia de estos últimos que utilizan operadores de recombinación y mutación para mejorar las soluciones los EDAs aprenden la distribución de probabilidad del conjunto seleccionado y a partir de esta simulan nuevos puntos que formarán parte de la población.

El paso más costoso y delicado en este algoritmo es el de estimar $p^s(x,t)$ y generar nuevos puntos de acuerdo a esta distribución. Debido a que el resultado de los algoritmos EDAs depende

Algoritmo 9 Algoritmo de Estimación de la Distribución.

```
Poner t ← 1;
Generar N >> 0 puntos aleatoriamente;
while no se alcance el criterio de parada do
  Seleccionar M ≤ N puntos de acuerdo a un método de selección;
  Estimar la distribución p^s(x, t) del conjunto seleccionado;
  Generar N nuevos puntos de acuerdo a la distribución p^s(x, t);
  Poner t ← t + 1;
end while
```

de como la distribución será estimada, los modelos gráficos son herramientas comunes capaces de eficientemente representar la distribución de probabilidad. Algunos autores [237, 203, 153] han propuesto las redes bayesianas para representar la distribución de probabilidad en dominio discreto, mientras que las redes Gausianas se emplean usualmente en dominio continuo [264].

Búsqueda Dispersa (SS)

La Búsqueda Dispersa o *Scatter Search* (SS) [115] es una metaheurística cuyos principios fueron presentados en [113] y que actualmente está recibiendo una gran atención por parte de la comunidad científica [150]. El Algoritmo 10 esquematiza los pasos básicos seguidos por este método y que a continuación describimos en más detalle.

Algoritmo 10 Búsqueda Dispersa.

```
P ← GenerarPoblacion()
RefSet ← GenerarConjuntoReferencia(P)
while no se alcance la condición de parada do
  S ← GenerarSubConjuntos(RefSet)
  for all s ∈ S do
    x_s ← Recombinar(s)
    x_s ← MetodoMejora(x_s)
    RefSet ← Actualizar(RefSet, x_s)
  end for
  if Convergencia(RefSet) then
    P ← GenerarPoblacion()
    RefSet ← GenerarConjuntoReferencia(P, RefSet)
  end if
end while
Salida: la mejor solución encontrada
```

- *Creación de la Población Inicial*: Este método se encarga de crear una población inicial de forma aleatoria pero que contenga buenas soluciones e intentando mantener una alta diversidad de soluciones.
- *Generación del Conjunto de Referencia*: Se crea el conjunto de referencia a partir de los elementos más representativos de la población inicial del paso anterior.
- *Generación de Subconjuntos*: Este método genera diferentes subconjuntos de soluciones a partir del conjunto de referencia, a los cuales se les aplicará la recombinación.

- *Método de Combinación de Soluciones*: Este método combina las soluciones de los subconjuntos generados en el paso anterior para producir nuevas soluciones.
- *Método de Mejora*: Las soluciones anteriormente generadas son refinadas mediante algún tipo de búsqueda local.
- *Actualización del Conjunto de Referencia*: Por último, las nuevas soluciones generadas reemplazan a algunas de las soluciones del conjunto de referencia si las mejoran en algún aspecto (calidad y/o diversidad).

Según la Definición 1, la principal diferencia con otros algoritmos es Φ_{SS} que es una función definida por partes de la siguiente forma:

$$\Phi_{SS}(\Theta) = \begin{cases} \omega_{crf}(\Theta) & si\ \omega_c(\Theta) \\ \{\omega_i(\theta') | \theta' \in \omega_r(\omega_s(\Theta))\} & en\ otro\ caso \end{cases}$$

donde:

- $\omega_{crf} : \mathcal{T}^k \rightarrow \mathcal{T}^\mu$, se corresponde que crea un nuevo conjunto de referencia.
- $\omega_c : \mathcal{T}^\mu \rightarrow \{true, false\}$, se corresponde con la función que comprueba la convergencia del conjunto de referencia.
- $\omega_s : \mathcal{T}^\mu \rightarrow (2^{\mathcal{T}})^{(\sum_{i=2}^{\mu}\binom{\mu}{i})}$, se corresponde con el operador que genera los subconjuntos.
- $\omega_r : 2^{\mathcal{T}} \rightarrow \mathcal{T}$, se corresponde con el operador de recombinación.
- $\omega_i : \mathcal{T} \rightarrow \mathcal{T}$, se corresponde con el operador de mejora.

Optimización basada en Colonias de Hormigas (ACO)

Los sistemas basados en colonias de hormigas o *Ant colony optimization* (ACO) [80, 81] son una metaheurísticas que están inspirados en el comportamiento en la búsqueda de comida de las hormigas reales. Este comportamiento es el siguiente: inicialmente, las hormigas explorar el área cercana a su nido de forma aleatoria. Tan pronto como una hormiga encuentra la comida, la lleva al nido. Mientras que realiza este camino, la hormiga va depositando un componente químico denominado feromona. Este componente ayudará al resto de las hormigas a encontrar la comida. Esta comunicación indirecta entre las hormigas mediante el rastro de feromona, las capacita para encontrar el camino más corto entre el nido y la comida. Esta funcionalidad es la que intenta simular este método para resolver problemas de optimización. En esta técnica, el rastro de feromona es simulado mediante un modelo probabilístico.

Algoritmo 11 Colonias de Hormigas.

while no se alcance la condición de parada **do**
  $ConstruccionSolucionPorHormiga()$
  $ActualizacionFeromona()$
  $OtrasActivades()$ {Acciones opcionales}
end while
Salida: la mejor solución encontrada

En el Algoritmo 11 se muestra el esquema general seguido por este algoritmo. Como se ve en ese código, esta técnica se basa en tres pasos: las construcción de una solución basado en el comportamiento de una hormiga, la actualización de parte de la información aportada por cada hormiga y otras posibles acciones que deban ser realizadas. Aunque en el Algoritmo 11 estas tres fases están indicadas una detrás de otra, realmente el algoritmo no fija ninguna planificación o sincronización a priori entre las fases, incluso pudiendo ser ejecutadas simultáneamente todas a la vez.

La fase $ConstruccionSolucionPorHormiga()$, como su nombre indica, lo que hace es construir una solución tentativa al problema a resolver. Para ello utiliza dos fuentes de información: por un lado utiliza un heurístico de construcción específico del problema y por otro lado utiliza el rastro de feromona depositado por las hormigas. El algoritmo combina ambas informaciones para ir construyendo cada uno de los componentes que forman la solución. La importancia que se le da la información heurística y la información del rastro de la feromona es determinada de acuerdo a varios parámetros que el usuario debe fijar de antemano.

La fase $ActualizacionFeromona()$ consta de dos pasos. El primer paso consiste en la *evaporación* de parte del rastro de feromona existente, que consiste en el decrecimiento uniforme de todos los valores de feromonas almacenados. Con esta evaporación se evita la rápida convergencia hacia áreas del espacio de búsqueda subóptimas. La segunda parte consiste en el incremento de ciertos valores de la feromona. Los valores elegidos para ser incrementados corresponden a los que han producido los componentes que forman la mejor solución (o mejores soluciones) encontradas en esta iteración del algoritmo.

Por último, la fase $OtrasActivades()$ se corresponden a actividades centralizadas que no se pueden tomar considerando la información aportada por una única hormiga, sino por el conjunto de ellas. Ejemplos de este paso pueden ser la aplicación de métodos de búsqueda local para mejorar las soluciones o la incorporación adicional de feromonas.

Se han propuesto diferentes variantes de este esquema general de acuerdo a cómo se instancien cada una de las fases. Un repaso de estas variantes se puede encontrar en [81].

En este caso, de nuestra definición formal, lo más significativo es que $\Xi_{ACO} = \{\theta^*, \mathcal{F}, \eta\}$, indicando que entre la información adicional, este algoritmo utiliza una matriz de feromona e información heurística, y $\Phi_{ACO} : \mathcal{F} \times \eta \rightarrow \mathcal{T}^{\mu}$, indicando la construcción de las nuevas soluciones a partir de la información contenida en Ξ_{ACO}.

2.2. Metaheurísticas Paralelas

Aunque el uso de metaheurísticas permite reducir significativamente la complejidad temporal del proceso de búsqueda, este tiempo sigue siendo muy elevado en algunos problemas de interés real. Por lo tanto, el paralelismo puede ayudar no sólo a reducir el tiempo de cómputo, sino que también puede producir una mejora en la calidad de las soluciones encontradas. Para cada una de las categorías mostradas en la sección anterior se han propuesto diferentes modelos de paralelismo acorde con sus características. Cada uno de esos modelos ilustran un acercamiento alternativo para tratar con el paralelismo. En los siguientes secciones se mostrarán los modelos paralelos más utilizados para los diferentes metaheurísticas presentadas en la sección anterior.

2.2.1. Modelos Paralelos para Métodos Basados en Trayectoria

En la literatura podemos encontrar que generalmente los modelos de paralelismo de este tipo de metaheurística siguen tres posibles esquemas: ejecutar en paralelo de varios métodos (*modelo de múltiples ejecuciones*), la exploración en paralelo del vecindario (*modelo de movimientos paralelos*) o el cálculo de la función de fitness en paralelo (*modelo de aceleración del movimiento*).

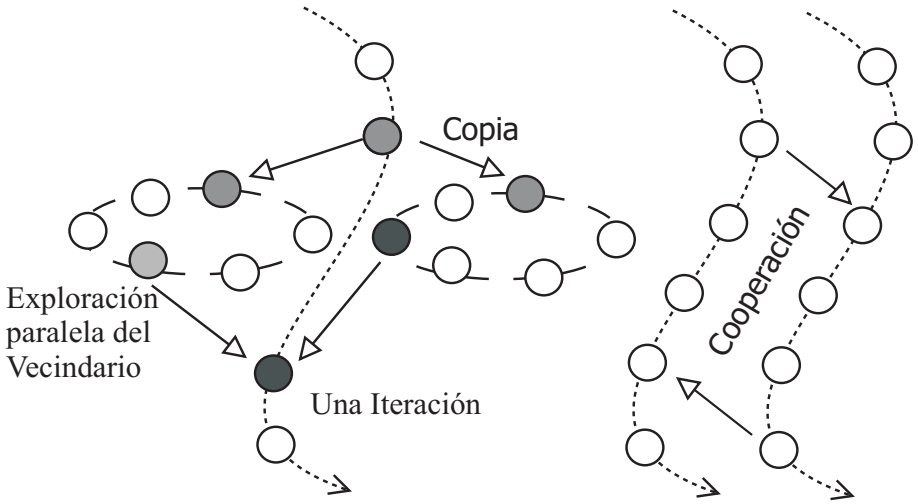


Figura 2.4: Modelos paralelos más usados en los métodos basados en trayectoria: en la izquierda se muestra en modelo de movimientos paralelos, donde se hace una exploración paralela del vecindario y el derecha, se detalla el modelo de ejecuciones múltiples, donde hay varios métodos ejecutando en paralelo y en este caso concreto cooperan entre ellos.

- **Modelo de múltiples ejecuciones:** Este modelo consiste en ejecutar en paralelo varios subalgoritmos ya sean homogéneos o heterogéneos. En general, cada subalgoritmo comienzo con una solución inicial diferente. Se pueden diferenciar diferentes casos dependiendo si los subalgoritmos colaboran entre sí o no. El caso de ejecuciones independientes se usa ampliamente porque es simple de utilizar y muy natural. En este caso, la semántica del modelo es la misma que la de la ejecución secuencial, ya que no existe cooperación. El único beneficio al utilizar este modelo respecto a realizar las ejecuciones en una única máquina, es la reducción del tiempo de ejecución total.

  Por otro lado, en el caso cooperativo (véase el ejemplo de la derecha de la Figura 2.2.1), los diferentes subalgoritmos intercambian información durante la ejecución. En este caso el comportamiento global del algoritmo paralelo es diferente al secuencial y su rendimiento se ve afectada por cómo esté configurada este intercambio. El usuario debe fijar ciertos parámetros para completar el modelo: qué información se intercambian, cada cuánto se pasan la información y cómo se realiza este intercambio. La información intercambiada suelen ser la mejor solución encontrada, o los movimientos realizados o algún tipo de información sobre la trayectoria realizada. En cualquier caso, esta información no debe ser abundante para que el coste de la comunicación no sea excesivo e influya negativamente en la eficiencia. También se debe fijar cada cuántos pasos del algoritmo se intercambia la información. Para elegir este valor hay que tener en cuenta que el intercambio no sea muy frecuente para que el coste de la comunicación no se nocivo ni muy poco frecuente, que podría provocar que el intercambio produzca poco efecto en el comportamiento global. Por último se debe indicar si las comunicaciones se realizarán de forma asíncrona o síncrona. En el caso síncrono, los subalgoritmos cuando llegan a la fase de comunicación, se esperan a que todos los subalgoritmo lleguen a este paso y cuando todos llegan se realiza la comunicación y cada subalgoritmo continua con la ejecución. En el caso asíncrono (que es el más usado), cuando cada subalgoritmo llega al paso de comunicación, realiza el intercambio sin esperar al resto.

- **Modelo de movimientos paralelos:** Los métodos basados en trayectoria en cada paso examinan parte de su vecindario y de él eligen a la siguiente solución a considerar. Este paso suele ser considerablemente pesado, ya que examinar el vecindario implica múltiples cálculos de la función de fitness. En este modelo tiene como objetivo acelerar este proceso mediante la exploración en paralelo del vecindario (véase el esquema de la izquierda de la Figura 2.2.1). Siguiendo un modelo granjero-trabajador, el granjero (el que ejecuta el algoritmo) pasa a cada trabajador la solución actual. Cada trabajador explora parte del vecindario de esta solución devolviendo la más prometedora. Entre todas estas soluciones devueltas el granjero elige una para continuar el proceso. Este modelo no cambia la semántica del algoritmo, si no que simplemente acelera su ejecución. Este modelo es bastante usado debido a su simplicidad.

- **Modelo de aceleración del movimiento:** En muchos casos, el proceso más pesado del algoritmo es el cálculo de la función de fitness, pero este cálculo se puede dividir en varios cálculos independientes más simples que, una vez calculados, se pueden combinar para computar el valor final de la función de fitness. En este modelo, cada uno de esos sub-cálculos se asignan a los diferentes procesadores y se procesan en paralelo, acelerando el cálculo total. Al igual que el anterior, este modelo tampoco modifica la semántica del algoritmo respecto a su ejecución secuencial.

Ahora daremos un breve estado del arte en las diferentes metaheurísticas basadas en trayectoria.

Enfriamiento Simulado Paralelo (PSA)

Según nuestro conocimiento, el SA fue la primera metaheurística basada en trayectoria en ser paralelizada. En la Tabla 2.1 se muestran algunas de las implementaciones paralelas más representativas del SA.

Tabla 2.1: Un resumen breve de SAs paralelos.

Artículo		**Modelo Paralelo**
[149]	(1987)	Aceleración del movimiento
[149]	(1987)	Movimientos paralelos
[224]	(1990)	Movimientos paralelos
[154]	(1996)	Múltiples ejecuciones (síncrono y asíncrono)
[131]	(2000)	Movimientos paralelos
[131]	(2000)	Múltiples ejecuciones (síncrono y asíncrono)
[185]	(2000)	Modelo de múltiples ejecuciones asíncrono
[53]	(2004)	Un framework para métodos paralelos (por ejemplo SA paralelo)
[22]	(2004)	Un framework para métodos paralelos (por ejemplo SA paralelo)

El modelo más investigado es el de movimientos paralelos [58, 224]. En este modelo, se evalúan de forma concurrente diferentes movimientos. Cada procesador genera y evalúa movimientos de forma independiente. El modelo puede sufrir inconsistencias, ya que los movimientos son realizados por procesadores aisladamente. Para manejar esta inconsistencia hay dos posibles aproximaciones: (1) la evaluación de los movimientos se realiza en paralelo pero sólo se aceptan movimientos no interactivos [131, 224]. Este esquema puede verse como un descomposición del dominio. Esta técnica permite preservar las propiedades de convergencia del algoritmo y produce buenos speedups [149, 131]. La dificultad de este esquema es cómo determinar los movimientos no interactivos. (2) El segundo esquema consiste en evaluar y aceptar en paralelo varios movimientos interactivos. Se permite ciertos errores en el cálculo de la función de aceptación. Estos errores son corregidos después de un cierto número de movimientos (por ejemplo, después de cada cambio de temperatura

[131]) mediante la sincronización entre los procesadores. Sin embargo, esto afecta a la convergencia del algoritmo paralelo comparado con la versión secuencial. Además, debido a la sincronización se produce una pérdida bastante importante de eficiencia [131].

Otro modelo ampliamente usado es el modelo basado en múltiples ejecuciones. Estas implementaciones paralelas usan múltiples cadenas de Markov [131, 154, 185]. Este modelo permite sobreponerse a los problemas de prestaciones de la estrategia de movimientos paralelos debido a que o bien sufren problemas por el error de aceptación o bien por restringir los movimientos posibles. En este modelo, cada procesador ejecuta un SA con una copia completa de los datos del problema. Los procesadores combinan dinámicamente sus soluciones mediante el intercambio de las mejores soluciones encontradas de forma asíncrono o síncrona.

Finalmente destacar [53] y [22] donde se presentan dos frameworks que ofrecen facilidades para generar SAs paralelos siguiendo cualquiera de los modeles. Por ejemplo, en MALLBA hemos desarrollado un SA donde estudiamos el comportamiento del modelo de múltiples ejecuciones para diferentes problemas [22].

Búsqueda Tabú Paralela (PTS)

En la Tabla 2.2 se resumen las implementaciones paralelas más importantes de la metaheurística búsqueda tabú. La mayoría de ellos están basados o en la múltiples ejecuciones o en la descomposición del vecindario siguiendo el esquema de movimientos paralelos [92, 209]. En [31, 46, 53] se usan diferentes modelos que han sido implementados como esqueletos de códigos generales y que pueden ser instanciados por el usuario.

Tabla 2.2: Un resumen breve de TSs paralelos.

Artículo		**Modelo Paralelo**
[92]	(1994)	Movimientos paralelos
[209]	(1995)	Movimientos paralelos
[249]	(1996)	Múltiples ejecuciones no cooperativo con balance de carga
[6]	(1998)	Múltiples ejecuciones cooperativas
[46]	(2001)	Múltiple ejecuciones no cooperativas, Maestro-esclavo con partición del vecindario
[70]	(2002)	Múltiples ejecuciones cooperativas
[31]	(2002)	Esqueletos paralelos para la búsqueda dispersa (todos los modelos)
[53]	(2004)	Esqueletos paralelos para la búsqueda dispersa (todos los modelos)

El modelo de movimientos paralelos ha sido aplicado en [92] y [209], obteniendo como conclusión que debido a la fuerte sincronización del modelo, sólo es rentable utilizarlo en aquellos problemas donde los cálculos requeridos en cada paso son muy costosos en tiempo. Por ejemplo [209] únicamente obtienen speedups lineales cuando resuelven instancias grandes del problema.

El modelo de múltiples ejecuciones ha sido ampliamente utilizado para realizar implementaciones paralelas de TS [6, 70, 249]. En la mayoría de ellas, cada procesador ejecuta un TS secuencial. Los diferentes TS pueden utilizar soluciones y parámetros diferentes. Estos TSs secuenciales pueden ser completamente independientes [249] o pueden cooperar de alguna forma [6, 70]. Por ejemplo en [6], la cooperación es realizada a través de un conjunto de soluciones de élite que está almacenado en un proceso maestro. En [70] muestra que estos modelos cooperativos pueden mejorar a las respectivas versiones secuenciales o paralelas de múltiples ejecuciones pero sin cooperación.

GRASP Paralelo

La Tabla 2.3 muestra que la mayoría de las implementaciones paralelas [157, 171, 200] de GRASP se basan en el modelo de múltiples ejecuciones. Muchas de esas implementaciones fueron propuestas por Resende y sus colaboradores. En estos casos, el paralelismo consiste en distribuir las iteraciones sobre diferentes procesadores. Cada procesador ejecuta una algoritmo secuencial con una copia completa de los datos del problema. Ya que las iteraciones del algoritmo son independientes y el intercambio de información entre los procesadores es escasa, estas implementaciones obtienen un rendimiento muy bueno. Por ejemplo, en [157] informan de un speedup casi lineal para el problema de la asignación cuadrática, en concreto consigue un speedup de 62 con 64 procesadores. En trabajos más recientes de paralelizaciones del GRASP, las iteraciones paralelas han utilizado el proceso de intensificación de path-relinking para mejorar las soluciones obtenidas [5, 8]. En [7] se propone una metodología para el análisis de GRASP paralelos.

Tabla 2.3: Un resumen breve de GRASPs paralelos.

Artículo		**Modelo Paralelo**
[157]	(1994)	Múltiples ejecuciones
[200]	(1995)	Múltiples ejecuciones
[171]	(1998)	Múltiples ejecuciones con balance de carga
[8]	(2000)	Múltiples ejecuciones con path-relinking
[5]	(2003)	Múltiples ejecuciones con path-relinking
[7]	(2003)	una metodología para el análisis de GRASP paralelos

El balance de la carga puede ser fácilmente equilibrada distribuyendo por igual las iteraciones entre los procesadores. Sin embargo, en casos heterogéneos este esquema básico puede provocar una fuerte pérdida de eficiencia. En [171], se presenta un modelo adaptativo y dinámico de la distribución de la carga. Se basa en una estrategia de granjero-trabajador. Cada trabajador ejecuta un pequeño número de iteraciones y cuando termina, pide al proceso granjero más iteraciones. Todos los trabajadores paran cuando se devuelve el resultado final. Como consecuencia de este modelo, los trabajadores más rápidos realizan más iteraciones que los otros. Esta aproximación permite reducir el tiempo de ejecución respecto a la planificación estática.

Búsqueda con Vecindario Variable Paralelo

Puesto que VNS es una metaheurística relativamente nueva, no ha sido demasiada investigada desde el punto de vista del paralelismo. Los dos principales trabajos relacionados en la literatura con el paralelismo del VNS son [107] y [71]. En [107], se proponen y comparan tres esquemas de paralelización: el primero sigue un esquema de bajo nivel que intenta acelerar la búsqueda mediante la paralelización de la búsqueda local; el segundo está basado en el modelo de múltiples ejecuciones independientes; y por último, la tercera estrategia propuesta implementa un modelo de múltiples ejecuciones independientes cooperativo y síncrono. Para probar los esquemas usan el problema del viajante con una instancia de 1400 ciudades. Los resultados muestran que los esquemas que siguen el modelo de múltiples ejecuciones obtienen mejores soluciones.

En [71] proponen una variante asíncrona y cooperativa del modelo de múltiples ejecuciones. Un proceso central mantiene actualizada la mejor solución encontrada. También se encarga de iniciar y terminar el proceso completo. A diferencia del modelo síncrono presentado antes, en esta propuesta las comunicaciones son iniciadas por los procesos trabajadores de manera asíncrona. Cuando un trabajador no puede mejorar más su solución, se la pasa al proceso maestro, para que actualice su estado si la solución proporcionada es mejor que la que tenía almacenada. El proceso maestro es el encargado de repartir la mejor solución para que sirva de punto de comienzo en los

Tabla 2.4: Un resumen breve de VNSs paralelos.

Artículo		Modelo Paralelo
[107]	(2002)	Búsqueda local paralela
[107]	(2002)	Múltiples ejecuciones independientes
[107]	(2002)	Múltiples ejecuciones cooperativo y síncrono
[71]	(2004)	Múltiples ejecuciones cooperativo y asíncrono

procesos trabajadores. Los autores prueban esta implementación con instancias del problema de la p-mediana de más de 1000 medianas y 11948 clientes. Los resultados muestran que el modelo propuesto permite reducir el tiempo de ejecución sin perder calidad en las soluciones encontradas respecto a la versión secuencial.

2.2.2. Modelos Paralelos para Métodos Basados en Población

El paralelismo surge de manera natural cuando se tratan con poblaciones, ya que cada individuo puede tratarse independientemente. Debido a esto, el rendimiento de los algoritmos basados en población se ve muy mejorado cuando se ejecuta en paralelo. A un alto nivel podríamos dividir las estrategias de paralelización de este tipo de métodos en dos: (1) Paralelización del cómputo, donde las operaciones que se llevan a cabo sobre los individuos son ejecutadas en paralelo; y (2) Paralelización de la población, donde se procede a la estructuración de la población.

Uno de los modelos más utilizados que siguen la primera de esas paralelizaciones es el conocido modelo de *Maestro-Esclavo* (también conocido como *paralelización global*). En este esquema, un proceso centrar realiza las operaciones que afectan a toda la población (como por ejemplo la selección de los algoritmos evolutivos) mientras que los procesos esclavos se encargan de las operaciones que afectan a los individuos independientemente (como la evaluación de la función de fitness, la mutación e incluso la recombinación). Con este modelo la semántica del algoritmo paralelo no cambia respecto al secuencial pero el tiempo global de cómputo es reducido. Este tipo de estrategias son muy usadas en los casos en que el cálculo de la función de fitness es un proceso muy costoso en tiempo. Otra estrategia muy utilizada es la acelerar el cómputo mediante la realización de múltiples ejecuciones independientes que obviamente se ejecutan más rápidamente usando múltiples máquinas que un sólo procesador. En este caso no existe ningún tipo de interacción entre las ejecuciones.

Sin embargo, la mayoría de los algoritmos basados en población encontrados en la literatura utilizan alguna clase de estructuración de los individuos de la población. Sobretodo este esquema es ampliamente utilizado en el campo de los algoritmo evolutivos. Entre los más conocidos tenemos los *distribuidos* (o de grano grueso) y los *celulares* (o de grano fino) [32].

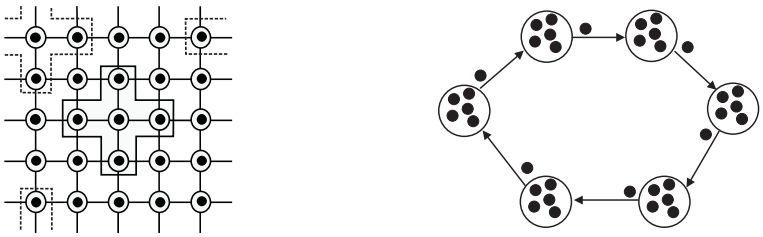


Figura 2.5: Los dos modelos más habituales para estructurar la población: a la izquierda el modelo celular y a la derecha el modelo distribuido.

En el caso de los algoritmo evolutivos distribuidos (véase esquema de la derecha en la Figura 2.5), la población está particionada en un conjunto de islas en cada una de la cual se ejecuta un EA secuencial. Las islas cooperan entre sí, mediante el intercambio de información (generalmente individuos aunque nada impide intercambiar otro tipo de información). Esta cooperación permite introducir diversidad en las sub-poblaciones, evitando caer así en los óptimos locales. Para terminar de definir este esquema el usuario debe dar una serie de parámetros como: la topología, indicando a donde puede enviar los individuos y de donde los puede recibir; el paso de migración, que indica cada cuantas iteraciones se produce el intercambio de información; la tasa de migración, indicando cuantos individuos son migrados; la selección con la que se eligen a los individuos a migrar y reemplazo que indica a que individuos de la población actual reemplazan los individuos inmigrantes; y finalmente, se debe decidir si estos intercambios se realizan de forma síncrona o asíncrona.

Por otro lado los algoritmos evolutivos celulares (véase esquema de la izquierda en la Figura 2.5) se basan en el concepto de vecindario. Los individuos tienen un pequeño vecindario donde se lleva a cabo la explotación de las soluciones. La exploración y la difusión de las soluciones al resto de la población se produce debido a que aunque cada individuo sólo opera con su vecindario, estos vecindario está solapados, lo que produce que las buenas soluciones se extiendan lentamente por toda la población.

A parte de estos modelos básicos, en la literatura también se han propuesto modelos híbridos donde se implementan esquemas de dos niveles. Por ejemplo es común la estrategia, donde en el nivel más alto tenemos un esquema de grano grueso, mientras que cada subpoblación se organiza siguiendo un esquema celular.

En los siguientes apartados daremos un breve estado del arte sobre los modelos más importantes que se han aplicado a cada una de las metaheurísticas basadas en población.

Algoritmos Genéticos Paralelos (PGA)

En la Tabla 2.5 mostramos algunos de los trabajos más importantes y representantes algoritmos genéticos paralelos. El modelo distribuido es el más utilizado en los GA paralelos, puesto que este modelo es fácilmente implementable en redes de computadores. dGA [250], DGENESIS [175] y GALOPPS [121] siguen el esquema básico del modelo distribuido aunque normalmente incluyen alguna característica para mejorar el rendimiento. Por ejemplo, dGA incluye un método de búsqueda local muy simple para mejorar las soluciones cuando se detecta cierto tipo de convergencia. Otros algoritmos siguiendo esta estrategia han sido diseñados para cumplir cierto objetivo específico. Por ejemplo GDGA [132] mediante la aplicación de diferentes operadores en las diferentes islas y una topología en hipercubo, logra un control explícito del balance de explotación/exploración seguido por el algoritmo de forma global. Otros algoritmos genéticos ejecutan modelos no ortodoxos del modelo de distribución, como por ejemplo GENITOR II [263] que está basado en una reproducción de estado estacionario.

Por otro lado, las implementaciones del modelo celular que se encuentran en la literatura son muy dependientes de las arquitecturas donde se ejecutan [123, 266]. Aunque son menos utilizados, también existen implementaciones que siguen otros modelos como EnGENEer [222] que siguen el modelo de maestro-esclavo. Finalmente, también existen bastante framework generales para el desarrollo de algoritmos genéticos paralelos como GAME [240], ParadisEO [53] o el que nosotros hemos desarrollado, Mallba [31]. Estos sistemas ofrecen facilidades para implementar cualquier modelo paralelo para GA.

Estrategias Evolutivas Paralelas (PES)

La Tabla 2.6 muestra un resumen de las implementaciones paralelas más representativas propuestas para las ESs. Muchos de estos trabajos [233, 238, 261] siguen el esquema celular, donde la

Tabla 2.5: Un resumen breve de GAs paralelos.

Algoritmo	Artículo		Modelo Paralelo
ASPARAGOS	[123]	(1989)	Grano fino. Aplica una búsqueda local si no se produce mejora
dGA	[250]	(1989)	Poblaciones distribuidas
GENITOR II	[263]	(1990)	Grano grueso
ECO-GA	[266]	(1991)	Grano fino
PGA	[193]	(1991)	Modelo distribuido con migración del mejor y búsqueda local
SGA-Cube	[88]	(1991)	Modelo de grano grueso. Implementado en nCUBE 2
EnGENEer	[222]	(1992)	Paralelización global
PARAGENESIS	[240]	(1993)	Grano grueso. Realizado para CM-200
GAME	[240]	(1993)	Un framework para la construcción de modelos paralelos
PEGAsuS	[221]	(1993)	Grano fino o grueso. Mecanismo de programación para MIMD
DGENESIS	[175]	(1994)	Grano grueso con migración entre las subpoblaciones
GAMAS	[210]	(1994)	Grano grueso. Usa 4 especies de cadenas (nodos)
iiGA	[159]	(1994)	GA con inyección de islas, heterogéneo y asíncrono
PGAPack	[155]	(1995)	Paralelización global (evaluaciones paralelas)
CoPDEB	[3]	(1996)	Grano grueso. Cada subpoblación aplica diferentes operadores
GALOPPS	[121]	(1996)	Grano grueso
MARS	[247]	(1999)	Entorno paralelo tolerante a fallos
RPL2	[213]	(1999)	Grano grueso. Muy flexible para definir nuevos modelos
GDGA	[132]	(2000)	Grano grueso con topología en hipercubo
DREAM	[39]	(2002)	Framework para EAs paralelos EAs
Mallba	[31]	(2002)	Un framework para la construcción de modelos paralelos
Hy4	[18]	(2004)	Grano grueso. Heterogéneo y con topología en hipercubo
ParadisEO	[53]	(2004)	Un framework para la construcción de modelos paralelos

población está estructurada y los individuos sólo pueden interactuar con sus vecinos más cercanos. Los resultados mostrados en estos trabajos muestran que este modelo es adecuado para problemas muy complejos, produciendo una probabilidad de convergencia a la solución óptima mucho mayor a la del GA panmíctico. También se han realizado trabajos teóricos [124] de este modelo, donde se analizan las propiedades de convergencia de este modelo. El modelo distribuido también ha sido usado frecuentemente a la hora de paralelizar este método [76, 227, 233], obteniendo speedup casi lineales en muchos casos para resolver tanto problemas continuos como de optimización combinatoria.

Tabla 2.6: Un resumen breve de ESs paralelos.

Artículo		Modelo Paralelo
[227]	(1991)	Modelo distribuido
[76]	(1993)	Modelo distribuido
[238]	(1994)	Modelo celular
[233]	(1996)	Modelo distribuido y modelo celular
[124]	(1999)	Modelo celular
[261]	(2004)	Modelo celular con una estructura del vecindario variable

Programación Genética Paralela (PGP)

La Tabla 2.7 resume las principales implementaciones paralelas de la GP. La programación genética en general no adecuada para las implementaciones masivamente paralelas celulares, ya que los individuos no son homogéneos variando ampliamente su complejidad y tamaño entre un

individuo y otro. Debido a esto, las implementaciones paralelas celulares de este método deben tener en cuenta estos aspectos. A pesar de esas dificultades existen implementaciones que siguen este modelo de grano fino, como por ejemplo [140]. También existen implementaciones de este modelo celular en plataformas de memoria distribuida, como la propuesta en [97] donde los autores muestras un GP celular paralelo adecuado para ser ejecutado en una red de computadores obteniendo un speedup casi lineal y con una escalabilidad muy buena.

Tabla 2.7: Un resumen breve de GPs paralelos.

Artículo		**Modelo Paralelo**
[140]	(1996)	Grano fino
[35]	(1996)	Modelo distribuido
[82]	(1996)	Maestro-esclavo
[36]	(1996)	Modelo distribuido
[211]	(1998)	Modelo distribuido
[91]	(2000)	Modelo distribuido
[97]	(2001)	Grano fino

El uso del modelo de grano grueso para el diseño de GPs paralelos es un poco controvertido. Mientras varios trabajos [35, 36, 91] muestran unos resultados muy buenos, otros [211] indican que este modelo no ayuda a la resolución de determinados problemas. A parte de estos dos grandes modelos, se han propuesto implementaciones siguiendo otras estrategias de paralelización pero ya en menor número que los otros dos. Los autores de [82] muestran un esquema de maestro-esclavo para la resolución de problemas con funciones de fitness cuyo cálculo requiere un tiempo muy elevado.

Colonias de Hormigas Paralelas (PACO)

Las colonias de hormigas debido a su estructura son muy buenos candidatos para ser paralelizados pero no existen demasiados estudios sobre el tema. A continuación mostramos brevemente los modelos e implementaciones paralelos más utilizados en la literatura de colonias de hormigas paralelas.

Tabla 2.8: Un resumen breve de ACOs paralelos.

Artículo		**Modelo Paralelo**
[50]	(1993)	Variantes de grano grueso y fino
[51]	(1997)	Intercambio de información cada k generaciones
[245]	(1998)	Ejecuciones independientes
[183]	(1998)	Modelo distribuido
[248]	(1999)	Maestro-esclavo
[177]	(2001)	Modelo distribuido
[184]	(2002)	Modelo distribuido
[214]	(2002)	Maestro-esclavo y ejecuciones independientes
[215]	(2002)	Maestro-esclavo muy acoplado
[78]	(2004)	Maestro-esclavo

En la Tabla 2.8 podemos observar que muchos de las implementaciones paralelas encontradas en la literatura siguen el modelo de maestro-esclavo [78, 214, 215, 248]. En este modelo, un proceso maestro envía algunas soluciones a cada proceso esclavo y después de un cierto número de generaciones, esas colonias envían información al proceso maestro que se encarga de calcular la matriz de feromonas. El modelo distribuido o en islas también se ha usado con bastante frecuencia

para realizar implementaciones paralelas de este método [177, 183, 184]. En estos estudios se ha llegado a la conclusión que el intercambio de poca información (en general la mejor solución de cada colonia) y con una topología simple (como el anillo) es el esquema que mejor rendimiento ofrece. Stützle [245] estudió el que quizás sea el modelo más sencillo de paralelizar un algoritmo (múltiples ejecuciones independientes sin cooperación), obteniendo unos resultados mejores que los ofrecidos por una implementación secuencial.

Algoritmos de Estimación de la Distribución Paralelos (PEDA)

Como vimos en la sección anterior, el paso más complejo del algoritmo es la estimación de la distribución $p^*(x,t)$ y la generación de nuevos puntos de acuerdo con ella. Por ello, muchas implementaciones paralelas intentan acelerar este paso. La paralelización de los EDAs puede hacerse a diferentes niveles: (1) nivel de estimación de la distribución (o nivel de aprendizaje), (2) nivel de muestreo de individuos (o nivel de simulación), (3) nivel de población, (4) nivel de evaluación de la función de fitness y (5) cualquier combinación de los niveles anteriores.

Tabla 2.9: Un resumen breve de EDAs paralelos.

Algoritmo	**Artículo**		**Modelo Paralelo**
pBOA	[197]	(2000)	Modelo híbrido entre nivel de aprendizaje y simulación
PA1BIC, PA2BIC	[165]	(2001)	Nivel de aprendizaje
dBOA	[198]	(2001)	Modelo híbrido entre nivel de aprendizaje y simulación
P^2BIL	[4]	(2003)	Nivel de población
$pEBNA_{BIC}$	[178]	(2003)	Nivel de aprendizaje
$pEBNA_{PC}$	[178]	(2003)	Nivel de aprendizaje
$pEGNA_{EE}$	[178]	(2003)	Nivel de aprendizaje
pcGA	[161]	(2004)	Nivel de aprendizaje
$pEBNA_{BIC}$	[179]	(2004)	Modelo híbrido entre nivel de aprendizaje y simulación

La Tabla 2.9 muestra un resumen de los trabajos más importantes en EDAs paralelos. Como hemos comentado antes, la mayoría de los trabajos [161, 165, 178, 197] intentan reducir el coste asociado a la estimación de la distribución, que es el paso más costoso. En general, estos algoritmos de aprendizaje de la distribución de probabilidad usan un procedimiento de puntuación+búsqueda principalmente definiendo una métrica que mide la bondad de cada red Bayesiana candidata con respecto a una base de datos de casos. Otra estrategia usada frecuentemente es la paralelización del muestreo de nuevos individuos para mejorar el rendimiento del comportamiento del algoritmo [179, 197, 198]. En menor medida, también se utilizan otros modelos [4]. En ese trabajo, los autores proponen un esqueleto para la construcción de EDAs distribuidos simulando la migración a través de vectores de probabilidades.

Búsqueda Dispersa Paralela (PSS)

Las principales estrategias para la paralelización de la búsqueda dispersa están resumidas en la Tabla 2.10. En paralelismo se puede emplear a tres niveles en el proceso de la búsqueda dispersa: al nivel del proceso de mejora, al nivel de combinación de las soluciones y al nivel del proceso completo. El primer nivel es de muy bajo nivel y consiste en paralelizar el proceso de mejora mediante el uso de los diferentes modelos de paralelización de métodos basados en trayectoria comentado en la sección anterior. En [106] usan el modelo de movimientos paralelos para distribuir el cálculo del proceso de búsqueda local. Han aplicado este modelo al problema de la p-mediana y los resultados muestran que además de reducir el tiempo de cómputo total, le permite mejorar la calidad de las soluciones.

Tabla 2.10: Un resumen breve de SSs paralelos.

Artículo		Modelo Paralelo
[106]	(2003)	Búsqueda local paralela con movimientos paralelos (SPSS)
[106]	(2003)	Modelo maestro-esclavo (RCSS)
[106]	(2003)	Ejecuciones independientes (RPSS)
[108]	(2004)	RCSS con diferentes métodos de combinación y parámetros

El segundo nivel de paralelismo se obtiene mediante la paralelización del mecanismo de combinación de soluciones. En esta estrategia se divide el conjunto de posibles combinaciones y estas porciones se distribuyen entre todos los procesadores y calcularlos en paralelo. Este modelo fue aplicado para el problema de la p-mediana y han obtenido los mejores resultados conocidos para las instancias que utilizan [106]. Otra variante de este modelo que ha sido implementada es que cada procesador utilice métodos de combinación y parámetros diferentes [108].

En el tercer nivel, el proceso completo es paralelizado y en cada procesador ejecuta una SS secuencial. Este modelo es similar al distribuido aunque hay que tener en cuenta los aspectos específicos de la metaheurística SS [108].

2.3. Conclusiones

En este capítulo hemos ofrecido una introducción al campo de las metaheurísticas, centrándonos con más detalle en los algoritmos evolutivos, que son los métodos que utilizaremos principalmente en el resto de la tesis.

En la primera parte del capítulo, hemos comenzado ofreciendo una formulación matemática genérica para cualquier metaheurística y después hemos un repaso por las técnicas más importantes y populares dentro de este campo. Estas breves descripciones han sido realizadas siguiendo una clasificación de las metaheurísticas que las dividen atendiendo al número de puntos del espacio de búsqueda con los que se trabajan en cada iteración.

En la segunda parte, nos hemos dedicado al análisis de estas técnicas desde el punto de vista del paralelismo, que es el aspecto más importante de la tesis. Se muestran los modelos paralelos genéricos para cada tipo de metaheurísticas y cómo se han aplicado en la literatura a las diferentes algoritmos, dando un breve estado del arte para cada una. Hemos visto como en general, las estrategias de paralelismo más utilizadas son las más desacopladas, es decir, la de múltiples ejecuciones con cooperación entre los subalgoritmos, lo que es bastante lógico, ya que este modelo se adapta muy bien a los sistemas paralelos clásicos donde existen una serie de máquinas conectadas por una red. El modelo maestro-esclavo también ha sido utilizado en bastantes ocasiones, ya que es una extensión inmediata de la versión secuencial, donde la semántica del algoritmo no varía. En general, esos dos modelos son los más extendidos en la comunidad científica, aunque no son los únicos como se ha visto a lo largo de este capítulo.

Capítulo 3

Construcción y Evaluación de Metaheurísticas

En el capítulo anterior describimos las metaheurísticas desde un punto de vista algorítmico. El siguiente paso es pasar de esa descripción a una implementación concreta con la que poder aplicarlos a diferentes problemas. En la literatura se han propuesto muchas implementaciones y muchas de ellas están accesibles desde la Web. Este paso es bastante importante, ya que hay que tener en cuenta muchos aspectos para conseguir una implementación correcta y eficiente, y al mismo tiempo, esta implementación debe ser lo más genérica posible para que pueda ser aplicada a un gran número de problemas sin necesidad de grandes modificaciones. Además esta fase se complica en el caso del desarrollo de algoritmo paralelos, ya que el manejo de varios procesos y los mecanismos de comunicación dificultan la codificación. La Ingeniería del Software define unos estándares para el desarrollo de software de calidad y libre de errores. Es este un aspecto muy ignorado en el software para metaheurísticas hasta el momento, debido principalmente a que los investigadores suelen proceder de otros campos como las Matemáticas o en menor grado de la Física, Ingenierías, etc. En la primera parte de este capítulo abordaremos esta fase, primero describiendo las diferentes alternativas de software y maneras de codificar el algoritmo y después presentando la aproximación que se ha seguido en esta tesis, la biblioteca MALLBA, que sigue un esquema orientado a objetos usando patrones software.

Conseguir esta implementación es sólo un paso intermedio más, para obtener el objetivo final que es la resolución de problemas complejos mediante metaheurísticas. Por eso, dicho paso final, de la resolución del problema y el análisis y difusión de los resultados es el más importante y crítico. Debido al carácter no determinista de las metaheurísticas y de la heterogeneidad de los sistemas paralelos, el análisis de los resultados se complica y es necesario un guía concreta y exhaustiva de cómo se debe realizar la experimentación y cómo informar de sus resultados. Esta guía metodológica la ofreceremos en la segunda mitad de este capítulo.

3.1. Implementación de Metaheurísticas

Un aspecto clave en el desarrollo del software es la reutilizabilidad. La reutilizabilidad puede ser definida como la habilidad de los componentes software para construir diferente aplicaciones [93]. A la hora de reutilizar metaheurísticas paralelas se nos ofrecen tres alternativas: *no reutilizar* (empezar desde cero), *sólo reutilizar el código* o *reutilizar ambos, el código y el diseño*. La aproximación basada en no reutilizar puede parecer atractiva, debido a que aparentemente la implementación de

las metaheurísticas parece simple. Debido a eso, muchos programadores han preferido desarrollar su propio software antes que utilizar el de terceras personas. Si embargo, este esquema tiene algunos inconvenientes, el esfuerzo en tiempo y energía para el desarrollo es alto, el proceso entero es propenso a errores, se dificulta el mantenimiento, etc. La aproximación de reutilizar sólo el código consiste, como su nombre indica, en reutilizar el código perteneciente a otras personas disponible en la Web, ya sea como programas aislados o como bibliotecas. Este tipo de código normalmente tienen partes dependientes de la aplicación que debe ser extraídas con anterioridad a la inserción del nuevo código perteneciente al nuevo problema a resolver. El cambio de esas partes es a menudo propenso a errores. La reutilización de código a través de bibliotecas [260] obviamente es mejor ya que han sido más testadas y suelen estar documentadas, siendo por lo tanto más fiables. Sin embargo, aunque las biblioteca permiten la reutilización del código, no facilitan la reutilización completamente de la parte invariante del algoritmo, especialmente de su diseño.

El esquema que reutilice ambos (diseño y código) está pensado para superar este problema, es decir, estas implementaciones están desarrollados para que se deba rehacer el menor código posible cada vez que se desee añadir un nuevo problema de optimización. La idea básica es capturar en componentes especiales la parte fija (o invariante) de los métodos correspondientes al método implementado. Los *framework* o *marcos de trabajo* [137] proporcionan un control total de la parte invariante de los algoritmos, y el usuario sólo debe incorporar los detalles dependientes del problema. Para conseguir esta característica, el diseño debe aportar una clara separación conceptual entre la parte correspondiente al algoritmo y la parte dependiente del problema.

En la literatura se han presentado múltiples implementaciones de metaheurísticas paralelas. La mayoría de ellas están accesible por Web, y pueden ser reutilizadas y adaptadas para un problema concreto. Sin embargo, para aprovechar esta aproximación, el usuario debe reescribir las secciones específicas del código. Esta tarea es tediosa, propensa a errores y produce un alto consumo de tiempo y energía, y además, el nuevo código desarrollado es difícil de mantener.

3.1.1. Necesidad del Paradigma de Orientación a Objetos

Como se ha comentado antes, una mejor forma de reutilizar el diseño y el código para las metaheurísticas y sus modelos paralelos existentes es mediante el uso de *frameworks*. El objetivo de éstos tiene dos vertientes: en primer lugar, son más fiables, ya que han sido más testados y están documentados, y en segundo lugar, permiten un mejor mantenimiento y eficiencia. Para que un diseño sea satisfactorio debe cumplir ciertos criterios. Los siguientes son los más importantes:

- **Maximizar la reutilización de código y diseño:** El marco de trabajo debe proporcionar al usuario un arquitectura completa (diseño) para que el usuario pueda incorporar su código. Además, este código se debe poder incorporar haciendo las mínimas modificaciones posibles. Este objetivo requiere una clara y completa separación entre los métodos correspondientes al algoritmo y los del problema que se desea resolver.

- **Utilidad y extensibilidad:** El framework debe permitir al usuario cubrir un amplio número de metaheurísticas, problemas, modelos paralelos, etc. La inclusión de nuevas características o la modificación de la existentes se debe poder realizar de forma sencilla por parte del usuario y sin implicar la totalidad de los componentes del sistema software.

- **Transparencia de uso de los métodos paralelos:** En el caso del que el software desarrollado aporte la posibilidad de la ejecución en paralelo, ésta se debe realizar de forma transparente para el usuario. Es más, esta ejecución distribuida debe ser robusta para garantizar la fiabilidad y la calidad de los resultados.

- **Portabilidad:** Para poder satisfacer al máximo número de usuario, el framework debe soportar el máximo número de plataformas de cómputo y sistemas operativos.

El paradigma de orientación a objetos permite el desarrollo de estos frameworks de manera adecuada y de hecho la mayoría están desarrollados siguiendo esta metodología de programación. Para la separación conceptual entre componente requerida por los marcos de trabajo, en general, es recomendable el uso de la composición más que el uso de la herencia [223]. Con esto se consigue que las clases sean más fáciles de reutilizar. La implementación resultante estará compuesto por un conjunto de clases reutilizables con una parte fija y otra variable. La parte fija encapsula el esquema genérico del algoritmo y es proporcionado por el framework. La parte variable depende del problema, siendo fija en el framework, pero debe ser implementada por el usuario con los detalles de la aplicación a tratar.

3.1.2. Software Paralelo

El campo de la computación paralela es una disciplina en continua evolución, y esta evolución involucra tanto al software como al hardware. En esta sección daremos una breve introducción a las herramientas más utilizadas en la construcción de software paralelo.

Arquitecturas para el Cómputo Paralelo

En la literatura se han dado varias clasificaciones sobre arquitecturas paralelas. La más usada es la taxonomía realizada por Flynn en 1972 [94]. En esa clasificación divide las plataformas atendiendo a dos aspectos: primero, si las unidades de cálculo ejecutan el mismo código o no y segundo, si todas las unidades de cómputo tiene los mismos datos o no. Entre las cuatro posibles categorías ofrecidas en esa categoría, la más usada (y en la que centraremos esta sección) es la *MIMD* (*Multiple Instruction, Multiple Data*), en la que cada procesador puede ejecutar un código diferente sobre datos diferentes. Si especializamos más esta categoría, se pueden encontrar dos tipos de plataformas MIMD: los multiprocesadores y los sistemas distribuidos.

En los multiprocesadores todas las unidades de cómputo que la forman pueden acceder de forma directa y completa a una memoria común. Dependiendo del tipo de acceso podemos seguir dividiendo esta categoría. Para los lectores interesados en este aspecto, pueden ver la clasificación completa en [28]. Para la comunicación entre los diferentes elementos de proceso, usan un mecanismo de *memoria compartida*, donde el intercambio de datos se produce mediante la lectura/escritura en partes comunes de la memoria.

Por otro lado, los sistemas distribuidos están compuestos por una serie de computadores interconectados, cada uno con su propio procesador, memoria privada e interfaz de red. Una organización típica de estos sistemas es los *COWs* (*Clusters of Workstations*), donde existen varios PC o estaciones de trabajo conectados mediante algún tipo de red. Debido a la tecnología de red, estos sistemas están limitados a estar compuestos a lo sumo de unos pocos cientos de máquinas. Como una extensión de los COWs surgieron los sistemas *grid*, que permiten unos miles de máquinas, pertenecientes a diferentes organizaciones y dominios, aprovechando la infraestructura de Internet.

Herramientas para Sistemas de Memoria Compartida

Existen diferentes herramientas para programar sistemas donde los diferentes procesadores se comunican mediante el acceso a una zona de memoria común. Podemos clasificarlos según al nivel en que sean proporcionados:

- **Sistema Operativo:** Los sistemas operativos modernos ofrecen dos alternativas principales la hora de programar aplicaciones siguiendo este modelo. Se pueden utilizar procesos independientes que usen algunos de los recursos del sistema operativo para la comunicación (semáforos, ficheros, etc). Pero en la actualidad, se imponen el uso de las hebras debido a sus ventajas (cambio de contexto más rápido, consumo de recursos menor, etc). El concepto de hebra consiste en tener diferentes flujos de control independientes en un único proceso. Aunque diferentes sistemas operativos empezaron desarrollando sus propias bibliotecas de hebras, a mediados de los 90 surgió una propuesta de estándar denominada hebras POSIX (*Pthreads*), que ofrece un interfaz unificado y portable de rutinas para C.

- **Lenguaje de Programación:** Muchos lenguajes de programación actuales ofrecen sus propios mecanismos para desarrollar estas aplicaciones paralelas. Un ejemplo de estos lenguajes es Java (`java.sun.com`) con sus *Java Threads*. Estas hebras funcionan de la misma manera que las explicadas en el párrafo anterior, pero tienen las ventajas inherentes al lenguaje Java, como son la portabilidad y las características orientadas a objetos ofrecidas por el lenguaje

- **Directivas del Compilador:** En este enfoque, se parte de un código secuencial, donde se "anota" con directivas del compilador, aquellas partes susceptibles de ser paralelizadas y la forma en la que deben serlo. Después el compilador atendiendo a esas directivas genera un código que puede ser ejecutado en paralelo. Un claro ejemplo de estos compiladores es OpenMP (`www.openmp.org`) que ofrece interfaces para C++ y Fortran.

Herramientas para Sistemas Distribuidos

Para estos sistemas encontramos un conjunto bastante amplio de herramientas que se pueden emplear a la hora de desarrollar aplicaciones siguiendo este esquema. Las principales son:

- **Bibliotecas de Paso de Mensaje:** Este esquema se basa en el intercambio explícito de mensajes entre los procesos a la hora de compartir la información. Este mecanismo está indicado principalmente para el desarrollo de aplicaciones paralelas en COWs. Existen varias herramientas que permiten este modelo. A más bajo nivel tenemos los *socket*, que son ofrecidos tanto por los sistemas operativos (*BSD Sockets* en Unix y *winsock* en Windows) como por los lenguajes de programación (Java o la plataforma .NET de Microsoft). A más alto nivel podemos encontrar bibliotecas como PVM [242] o MPI [99] que ofrecen un conjunto más rico y completo de primitivas para la comunicación entre procesos y la sincronización.

- **Sistemas Basados en Objetos:** Este sistema es una evolución de los sistemas *RPC* (*Remote Procedure Call*). En estos sistemas, el programador utiliza el modelo orientado a objetos como si la aplicación fuese secuencial, aunque después en la ejecución, los objetos pueden estar distribuidos en diferentes máquinas. Normalmente existe un nivel intermedio llamado *middleware* que oculta las características de bajo nivel como son las comunicaciones y las características del sistema operativo. Como paradigmas dentro de esta categoría se pueden distinguir CORBA (*Common Object Request Broker Architecture*) y Java RMI (*Remote Method Invocation*).

- **Sistemas Basados en Internet:** También llamados sistemas grid, aprovechan la infraestructura de Internet para utilizar recursos de todo el mundo y que pueden estar en diferentes dominios e instituciones. Este tipo de sistemas ofrecen una gran complejidad, debido a la gran heterogeneidad existente, problemas de seguridad, inestabilidad de los componentes, etc. Globus [101] se ha convertido en el estándar *de facto* para estos sistemas, aunque también existen otros que son muy utilizados, como Condor (`www.cs.wisc.edu/condor`).

3.1.3. La Biblioteca MALLBA

La utilización de bibliotecas de algoritmos genéricos u otro tipo de herramientas similares es una manera de reducir el esfuerzo y obtener resultados eficientes a la hora de solucionar problemas de alta dificultad. En el pasado se han propuesto varias herramientas que ofrecen implementaciones paralelas de técnicas de optimización genéricas como enfriamiento simulado, ramificación y poda o algoritmos genéticos (véase por ejemplo [84, 146, 155, 253]). También, algunos marcos de trabajos existentes, tales como *Local++* [52], su sucesor *EasyLocal++* [110], *Bob++* [73], y el proyecto de código abierto COIN de IBM [136] proporcionan implementaciones secuenciales y paralelas para algoritmos exactos, heurísticos e híbridos, pero carecen de características para integrarlos (véase unas buenas revisiones de bibliotecas en [259] y [176]). Sin embargo, estos marcos e implementaciones no integran técnicas de optimización de varios tipos a la vez, y no ofrecen la posibilidad de ejecución de forma conjunta y transparente en ambos entornos, secuencial y paralelos. Otros marcos, tal como el proyecto DREAM [39] o *OpenBeagle* [103] tiene un objetivo similar al que persigue la biblioteca MALLBA, aunque están orientados principalmente a la computación evolutiva. *ParadisEO* [54] y *TEMPLAR* [138] proporcionan un marco mucho más general para metaheurísticas paralelas y realiza una aproximación similar a que seguimos en nuestra librería que presentamos. Otras bibliotecas escapan de este ámbito: por ejemplo *Sutherland* [174] y LEAP [85] ano son paralelas. mientras que PISA [47] es demasiado específica (enfocada a la optimización multi-objetivo).

El proyecto MALLBA[1] tiene como plan el desarrollo de una biblioteca de esqueletos para la optimización combinatoria que incluye técnicas exactas, heurísticas e híbridas. Soporta de forma amigable y eficiente tanto plataformas secuenciales como paralelas. Respecto a los entornos paralelos, son considerados tanto las redes de área local (LANs) como las de área extensa (WANs).

Las tres principales características que se pueden destacar de la biblioteca MALLBA son la integración de todos los esqueletos bajo unos mismos principios de diseño, la facilidad de cambiar entre entre entorno secuenciales y paralelos, y por último, la cooperación entre los esqueletos para proporcional nuevos híbridos.

En los siguientes párrafos, describiremos en mayor detalle cada uno de los aspectos de este biblioteca.

La Arquitectura MALLBA

Desde un punto de vista del hardware, la infraestructura de MALLBA está compuesta por computadores y redes de comunicación de las Universidades de Málaga (UMA), La Laguna (ULL) y Barcelona (UPC), todas ellas de España. Esas tres universidades están conectadas a través de RedIRIS, la red española de la comunidad académica y científica. Esta red está controlada por el CSIC (*Consejo Superior de Investigaciones Científicas*) que conecta a las principales universidades y centro de investigación en España. RedIRIS es una WAN que usa tecnología ATM de 34/155 Mbps. La versión actual de la biblioteca MALLBA funciona sobre clusters de PCs que usen Linux.

Respecto al software, todos los esqueletos que he desarrollado en MALLBA son implementados siguiendo el esquema de *esqueletos software* (similar al patrón de estrategia [105]) con un interfaz interno y común y otro público. Un esqueleto es una unidad algorítmica, que en forma de plantilla, implementa un algoritmo genérico. Un algoritmo se instanciará para resolver un problema concreto, rellenado los requerimientos especificados en su interfaz. Esta característica permite el rápido prototipado de aplicaciones y el acceso transparente a plataformas paralelas. En la biblioteca MALLBA, cada esqueleto implementa una técnica de los campos de la optimización exacta, heurística e híbrida.

[1] `http://neo.lcc.uma.es/mallba/easy-mallba/index.html`

Toda la biblioteca está desarrollada en C++. Se eligió este lenguaje debido a que proporciona las características de un lenguaje de alto nivel orientado a objetos y al mismo tiempo, genera ejecutables muy eficientes que un punto muy importante en esta biblioteca.

Como ya se comentó en las secciones anteriores un aspecto clave es la separación conceptual de la parte fija del algoritmo y la parte correspondiente al problema. El diseño basado en esqueletos soluciona este aspecto, ya que por consiste en crear un conjunto de clases separadas en dos apartados: unas contienen el comportamiento del algoritmo y son independientes del problema y cooperan con el otro conjunto de clases a través de un pequeño interfaz fijo pero lo suficientemente genérico para no limitar las características del problema. Este último conjunto de clases sólo tienen definido este interfaz y es el usuario el encargado de instanciarlas con los datos del problema concreto. En la otra parte además del comportamiento del algoritmo también están incluidos todos aspectos que pueden suponer mayor problema al usuario como puede ser los aspectos de paralelismo, que de esta forma quedan ocultos al usuario y pueden ser usado de manera transparente.

La Interfaz de Usuario

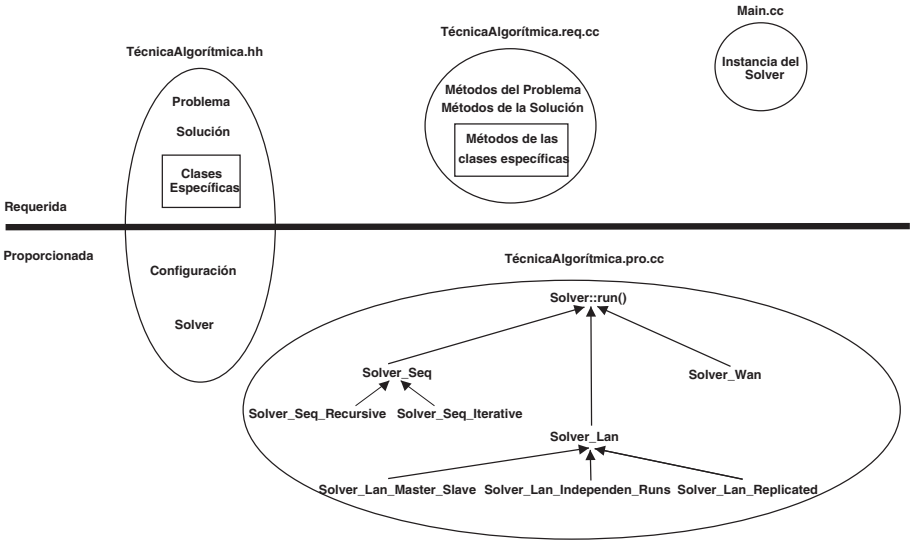
Como se comenta antes, todo el código MALLBA ha sido desarrollado en C++; para cada algoritmo de optimización se proporciona un conjunto de clases que –en función de su dependencia del problema objetivo– pueden agruparse en dos categorías: clases *provistas* y clases *requeridas* (véase la Figura 3.1(a)):

- **Clases Provistas**: Las clases englobadas dentro de esta categoría son las responsables de implementar toda la funcionalidad básica del algoritmo correspondiente. En primer lugar, la clase `Solver` encapsula el motor de optimización del algoritmo que se trate. Este motor de optimización es plenamente genérico, interactuando con el problema a través de las clases que el usuario debe proporcionar, y que serán descritas más adelante. En segundo lugar, existe la clase `SetupParams`, encargada de contener los parámetros propios de la ejecución del algoritmo, por ejemplo, el número de iteraciones, el tamaño de la población en un algoritmo genético, el mecanismo de gestión de la cola de subproblemas en un algoritmo de ramificación y poda, etc. Otra clase provista es `Statistics`, cuya finalidad es la recolección de estadísticas propias del algoritmo empleado. Finalmente, existen las clases `StateVariable` y `StateCenter` que facilitan la hibridación de algoritmos (gestión del estado).

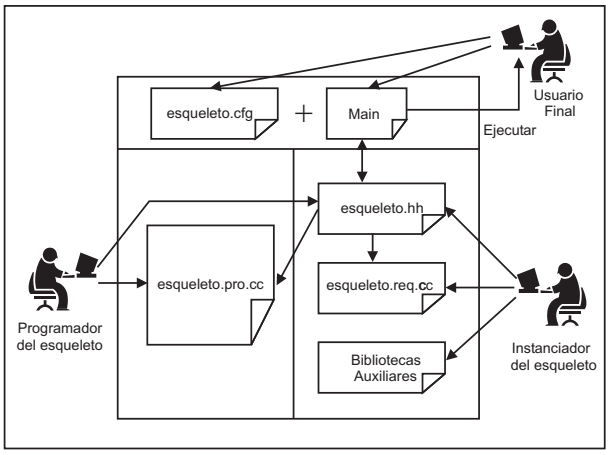
- **Clases Requeridas**. Estas clases son las responsables de proporcionar al esqueleto detalles sobre todos los aspectos dependientes del problema objetivo. No obstante, debe reseñarse que a pesar de esta dependencia del problema, la interfaz de las mismas es única y prefijada, permitiendo de esta manera el que las clases provistas puedan usarlas sin necesidad de relacionarse directamente con los detalles del problema. Entre las clases requeridas están la clase `Problem` (que debe proporcionar los métodos necesarios para manipular los datos propios del problema), la clase `Solution` (que encapsulará el manejo de soluciones parciales al problema tratado), la clase `UserStatistics` (que permitirá al usuario recoger aquella información estadística de su interés que no estuviera siendo monitorizada por la clase provista `Statistics`) así como otras clases que dependen del esqueleto algorítmico elegido.

Así pues, el usuario de MALLBA sólo necesita implementar las estructuras de datos dependientes del problema, así como dotar de un comportamiento específico a los métodos incluidos en las interfaces de las clases requeridas.

En relación de los esqueletos de código podemos distinguir tres perfiles de usuarios (Figura 3.1(b)) aunque nada impide que los tres perfiles se den en una misma persona o en varias. Los perfiles son:



(a) Arquitectura de un esqueleto Mallba. La línea horizontal establece la separación entre las clases C++ que el usuario debe rellenar (parte superior) y las clases del esqueleto que ya están incluidas y operativas (parte de abajo).



(b) Interacción entre los diferentes tipos de usuarios y tipos de ficheros conforme a los esqueletos de Mallba.

Figura 3.1: Esqueletos Mallba: Estructura e interacción.

- **Programador del Esqueleto,** que es el experto en el dominio del algoritmo desarrollado, y se encarga de diseñar e implementar el esqueleto a partir del conocimiento del algoritmo. Este perfil es el encargado de decidir qué clases son proporcionadas por el esqueleto y cuales deben ser facilitadas por el experto del problema. También debe decidir el mecanismo de comunicación existente para permitir la ejecución paralela.

- **Experto en el Dominio del Problema,** que es el encargado de instanciar el esqueleto con los datos adecuados al problema del cual es experto. Aunque no tiene por qué conocer la implementación del algoritmo, debe saber cómo evoluciona desde un punto de vista numérico, ya que debe ser capaz de identificar la necesidad y finalidad de las clases y métodos que debe rellenar para el problema que está resolviendo.

- **Usuario Final,** es el encargado de utilizar el esqueleto de código una vez que ha sido instanciado por el usuario correspondiente al perfil anterior. El usuario correspondiente al perfil que se está comentando será el encargado de dar la configuración a los parámetros del esqueleto (fichero `<esqueleto>.cfg`) de acuerdo con la ejecución que desee llevar a cabo. Este usuario tendrá a su disposición toda la información sobre a ejecución del algoritmo, permitiéndole hacer un seguimiento, obtener estadísticas, conocer el tiempo consumido hasta la mejor solución, etc.

En la Figura 3.1(b), se muestra gráficamente el papel de cada uno de estos perfiles.

La Interfaz de Comunicación

El proporcionar una plataforma paralela ha sido uno de los objetivos principales de Mallba. Las redes de de área local de computadores son actualmente muy baratas y gozan de gran popularidad en los laboratorios y departamento. Más aún, la tecnología actual de Internet nos permite comunicar esas redes para poder explotar los recursos de forma conjunto de numerosos sitios geográficamente distribuidos.

Para este fin, es necesario disponer de un mecanismo que permita comunicar los esqueletos entre sí, tanto en redes de área local (LANs) como de área extensa (WANs). Como los esqueletos los hemos desarrollado a un alto nivel, sería deseable desarrollar un mecanismo de comunicación también de alto nivel pero sin pérdida de rendimiento.

Son necesarios varios pasos para construir este sistema de comunicación: primero, hay que examinar el software paralelo existente; entonces, debemos elaborar nuestra propuesta y finalmente, implementarla en C++ para que pueda ser utilizada por los esqueletos de Mallba.

En el apartado 3.1.2 de este mismo capítulo ya hemos realizado una revisión sobre el software paralelo más relevante a la hora de implementar aplicaciones paralelas. Tras evaluarlos, nuestra primera conclusión era que necesitamos de nuestro propio sistema de comunicaciones, adaptado a las necesidades de nuestra librería, pero basado en un estándar eficiente, capaz de ser válido en el futuro.

Esas características las podemos encontrar (casi en exclusiva) si desarrollamos nuestro software basado en MPI. MPI reúne varias características muy interesantes: en primer lugar es un estándar y posee varias implementaciones muy eficientes; en segundo lugar su uso se está volviendo muy frecuente en la comunidad de software paralelo y por último, está integrado con los sistemas más modernos y prometedores como es Globus.

Aunque no existe ningún inconveniente en usar MPI directamente, hemos preferido desarrollar una capa intermedia muy ligera a la que hemos denominado `NetStream` (véase la Figura 3.2), que permite un uso más sencillo de MPI sin pérdida de eficiencia. Con este herramienta, un programador de Mallba puede evitar la gran cantidad de parámetros que se necesitan en MPI e interactuar

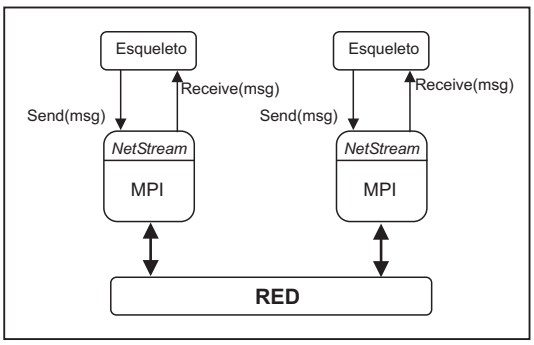


Figura 3.2: Sistema de comunicación NetStream sobre MPI.

con la red simplemente con *modificadores de flujo*, lo que permite realizar operaciones complejas con la red simplemente mediante el uso de los operadores habituales de flujo `<<` y `>>`.

Entonces, el objetivo de `NetStream` es permitir a los esqueletos Mallba intercambiar datos de forma eficiente y estructurada, manteniendo a la vez su alto nivel de abstracción y facilidad de uso. Los servicios ofrecidos por este sistema pueden clasificarse en dos grupos: por un lado unos servicios básicos para los no expertos en comunicaciones y uno avanzada para satisfacer los requerimientos de los usuarios más expertos. Entre los servicios básicos podemos encontrar:

- Envío y recepción de datos simples de forma empaquetada y no empaquetada a través de operadores de flujo
- Servicios de sincronización (barreras, comprobación de recepción asíncrona, etc.) mediante manipuladores de flujo.
- Servicio básico de manejo de procesos
- Otras características como el control del inicio y fin del programa

Entre las características avanzadas de este sistema destacan:

- Manejo avanzado de procesos, permitiendo la creación de grupos de procesos independientes.
- Servicios de adquisición on-line del estado de la red, muy apropiados para el desarrollo de aplicaciones en redes WAN donde el estado de carga de los enlaces es muy variable.

Todos estos servicios proporcionan un marco de programación de alto nivel para desarrollar nuestros esqueletos paralelos de forma eficiente.

La Interfaz de Hibridación

En esta sección discutimos los mecanismos disponibles en Mallba para tratar con la combinación de los esqueletos en búsqueda de métodos nuevos y más eficientes. En un sentido amplio, la hibridación [75] consiste en la inclusión de conocimiento del problema en el método de búsqueda general Estos algoritmos híbridos se suelen clasificar en débiles o fuertes, según el nivel de solapamiento existente entre los esqueletos que intervienen en la hibridación [69]:

- un híbrido es **fuerte** cuando el nivel de solapamiento entre los algoritmo es alto, como por ejemplo, la inclusión de un algoritmo como parte del problema a resolver por otro algoritmo, o al incluir un algoritmo como operador de un algoritmo evolutivo.

- un híbrido es **débil** si el nivel de solapamiento entre los algoritmo involucrados es bajo. Este tipo de híbridos consiste en la ejecución alternada de los algoritmos participantes, de forma que el resultado de la ejecución de uno de ellos sirve como punto de partida en la ejecución del siguiente, con lo que se avanza en el proceso de exploración del espacio de búsqueda.

El término "hibridación" ha sido utilizado en la literatura con significados muy diferentes. En esta tesis lo utilizamos para referirnos a la combinación de diferentes métodos de búsqueda para producir uno nuevo (hibridación débil). Tanto en la teoría [265] como en la práctica [75], la hibridación se ha convertido en un mecanismo esencial para el diseño de algoritmos eficientes para dominios específicos. Por esta razón, en MALLBA hemos desarrollado algunas herramientas para facilitar la creación de esqueletos híbridos.

Para la interacción entre esqueletos definimos el *estado del esqueleto*. En este estado se almacena de forma genérica toda la información necesaria para influir en su comportamiento . Estos valores son manejados de forma abstracta por las clases `StateVariable` y `StateCenter`. Mediante estas clases, un algoritmo puede obtener información de otro con el que coopere o influir en su comportamiento de forma genérica.

3.2. Diseño Experimental y Evaluación de Metaheurísticas

A diferencia de los algoritmos exactos, que son métodos deterministas y siempre ejecutan los mismos pasos y en el mismo orden, las metaheurísticas son algoritmos típicamente no determinista y pueden producir diferentes resultados al ser ejecutados varias veces aún teniendo la misma configuración. Esta característica dificulta ya no sólo la comparación entre diferentes metaheurísticas sino también cómo mostrar los resultados.

En general para analizar las metaheurísticas se pueden usar dos aproximaciones diferentes: un análisis teórico (caso peor, caso medio, etc.) o un análisis experimental. Varios autores [125, 143] han desarrollado análisis teóricos para un gran número de heurísticas y problemas. Pero, su dificultad, que complica obtener resultados para algoritmos y problemas más realistas, limita en gran medida su ámbito de aplicación. Como consecuencia de esto la mayoría de los algoritmos son evaluados *ad-hoc* de forma *empírica*.

Un análisis experimental normalmente consiste en aplicar el algoritmo desarrollado a un conjunto de problemas bien conocidos y comparar la calidad de las soluciones obtenidas y los recursos que han necesitado (en general, su tiempo de cómputo). Otros investigadores [225, 216] han intentado ofrecer alguna clase de marco metodológico para tratar con la evaluaciones experimentales de los heurísticos, que es de lo que trata el resto de capítulo. Hay muchos aspectos de la evaluación de algoritmos que deben ser tenidos en cuenta: el diseño experimental, la elección de instancias de prueba, la medición del rendimiento del algoritmo (de forma significativa), análisis estadísticos y una clara presentación de los resultados. En [173] se puede ver un excelente resumen sobre simulaciones algorítmicas y análisis estadísticos. En ese documento, McGeoch incluye un conjunto muy extenso sobre referencias básicas en el campo de los métodos estadísticos y una guía general para el diseño de experimentos.

En el resto del capítulo, no centraremos en cómo se deberían realizar los experimentos, y cómo los resultados deberían ser mostrados para permitir comparaciones justas con otras metaheurísticas. En concreto, le dedicaremos especial al caso de los algoritmos paralelos, ya que esta tesis se engloba en ese campo y además al tratar este tema, también se abarca el caso secuencial. Acabaremos el capítulo con una pequeña guía sobre los métodos estadísticos que debemos tener en cuenta para asegurar que nuestras conclusiones son correctas.

3.2.1. Medidas de Rendimiento Temporal Paralelo

Existen diferentes métricas para medir el rendimiento de los algoritmos paralelos. Inicialmente discutiremos en detalle las métricas más usadas en en ámbito de los algoritmos paralelos, es decir, el speedup y la eficiencia, y de cómo se deben usar para obtener resultados significativos y correctos. Después resumiremos el resto de medidas que se encuentran en la literatura.

Speedup

La métrica más importante para los algoritmos paralelos es sin ninguna duda el *speedup*. Esta métrica compara dos tiempos dando como resultado el ratio entre los tiempo de ejecución secuencial y el paralelo. Por lo tanto, el primer aspecto que deberíamos aclarar es lo que se entiende por tiempo. En un sistema uni-procesador, una medida típica de rendimiento es el *tiempo de CPU* para resolver el problema; se considera que este tiempo es el que el procesador ha estado ejecutando el algoritmo, excluyendo el tiempo de lectura de los datos del problema, la escritura de los resultados y cualquier tiempo debido a otras actividades del sistema. En el caos paralelo, el tiempo no es la suma de los tiempos de CPU de cada procesador, ni el más largo de ellos. Puesto que el objetivo del paralelismo es la reducción del tiempo total de ejecución, este tiempo debería incluir claramente cualquier sobrecarga debida al uso de algoritmos paralelos. Por lo tanto, la elección más prudente para medir el rendimiento de un código paralelo es el *tiempo real* para resolver el problema. Es decir, debemos medir el tiempo desde que comienza hasta que termina el algoritmo completo.

El speedup compara el tiempo de la ejecución secuencial con el tiempo correspondiente para el caso paralelo a la hora de resolver un problema. Si notamos por T_m el tiempo de ejecución para un algoritmo usando m procesadores, el speedup es el ratio entre la ejecución más rápida en un sistema mono-procesador T_1 y el tiempo de ejecución en m procesadores T_m:

$$s_m = \frac{T_1}{T_m} \tag{3.1}$$

Para algoritmos no deterministas no podemos usar esta métrica directamente. Para esa clase de métodos, se debería comparar el tiempo *medio* secuencial con el tiempo *medio* paralelo:

$$s_m = \frac{E[T_1]}{E[T_m]} \tag{3.2}$$

Con esta definición se puede distinguir entre: speedup *sublineal* ($s_m < m$), *lineal* ($s_m = m$) y *superlineal* ($s_m > m$). La principal dificultad con esta medida es que los investigadores no se ponen de acuerdo en el significado de T_1 y T_m. En un estudio realizado por Alba [11] distingue entre diferentes definiciones de speedup dependiendo del significado de esos valores (véase la Tabla 3.1).

Tabla 3.1: Taxonomía de las medidas de speedup propuesta por Alba [11].

I. Speedup Fuerte
II. Speedup Débil
A. Speedup con parada por calidad de soluciones
1. Versus panmixia
2. Orthodoxo
B. Speed con un esfuerzo predefinido

El *speedup fuerte* (tipo I) compara el tiempo de ejecución paralelo respecto al mejor algoritmo secuencial. Esta es la definición más exacta de speedup pero debido a lo complicado que es conseguir el algoritmo más eficiente actual, la mayoría de los diseñadores de algoritmos paralelos no la usan.

El *speedup débil* (tipo II) compara el algoritmo paralelo desarrollado por el investigador con su propia versión secuencial. En este casos, se pueden usar dos criterios de parada: por la calidad de soluciones y por máximo esfuerzo. El autor de esta taxonomía descarta esta última definición debido a que compara algoritmos que no producen soluciones de similar calidad, lo que va en contra del espíritu de esta métrica. Para el speedup débil con parada basada en la calidad de soluciones se proponen dos variantes: comparar el algoritmo paralelo con la versión secuencial canónica (tipo II.A.1) o compara el tiempo de ejecución del algoritmo paralelo en un procesador con el tiempo que tarda el mismo algoritmo pero en m procesadores (tipo II.A.2). En el primer es claro que se están comparando dos algoritmos claramente diferentes.

Barr y Hickman [43] presentan otra taxonomía diferente: *speedup*, *speedup relativo* y *speedup absoluto*. En esta clasificación, el *speedup* mide el ratio entre el tiempo del código secuencial más rápido en una maquina paralela con el tiempo del código paralelo usando m procesadores de la misma máquina. El *speedup relativo* es el ratio del tiempo de ejecución del código paralelo en un procesador con respecto al tiempo de ejecución de ese código en m procesadores. Esta definición es muy similar al tipo II.A.2 presentado antes. El *speedup absoluto* compara la ejecución secuencial más rápida en cualquier ordenador con el tiempo en paralelo en m procesadores. Esta métrica es la misma que el speedup fuerte definido por [11].

Como conclusión, es claro que las metaheurísticas paralelas deben computar con similar precisión que las secuenciales. Sólo en esos casos es válido comparar tiempo. Entonces se deberían tomar tiempos medios y comparar el tiempo del código paralelo ejecutado en una máquina y el tiempo que tarde ese mismo código en m máquinas. Con todo esto se define un sólido modelo para comparaciones, tanto prácticas (no se necesita el mejor algoritmo) como ortodoxo (mismos códigos, misma precisión).

Speedup Superlineal

Aunque varios autores han presentado resultados con speedup superlineal [45, 159], su existencia provoca todavía controversia. De hecho, se pueden señalar varias fuentes que pueden llevar a conseguir de este tipo de speedup:

- *Implementación:* El algoritmo ejecutado en un procesador es "ineficiente" por algún motivo. Por ejemplo, si el algoritmo usa listas de datos, en paralelo pueden ser manejadas más rápido ya que son más cortas. Además, el paralelismo puede simplificar varias operaciones del algoritmo.

- *Numérica:* Puesto que el espacio de búsqueda es normalmente muy grande, el algoritmo secuencial puede requerir recorrer una porción grande de él antes de encontrar el óptimo. Por otro lado, la versión paralela la puede encontrar más rápido ya que la exploración realizada en el espacio de búsqueda es diferente.

- *Física:* Cuando cambiamos de una plataforma secuencial a una paralela, normalmente se piensa únicamente en un incremento en el poder de cálculo, pero también se produce un incremento en otros aspectos, como la memoria principal, la caché, etc. Estos nuevos recursos pueden ser aprovechados de forma eficiente por el algoritmo produciendo un speedup superlineal (véase la Figura 3.3).

Por lo tanto, se puede concluir que aunque no es lo habituales posible obtener speedups superlineales mediante una configuración eficiente del algoritmo paralelo [79, 128, 29].

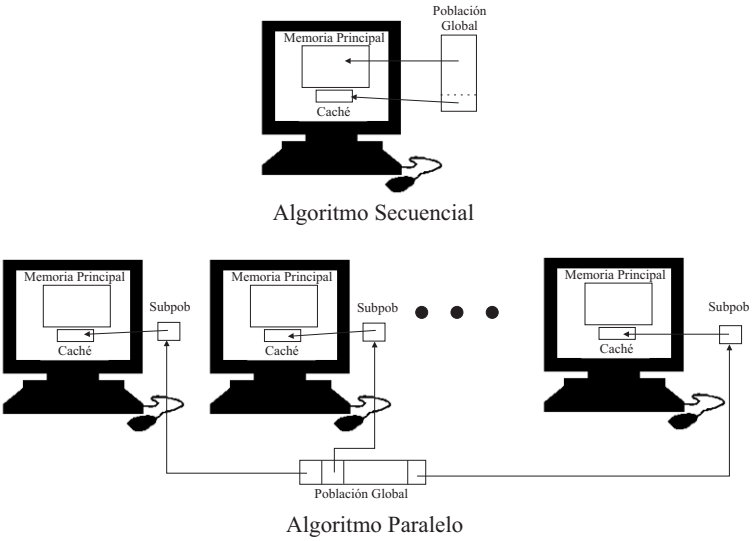


Figura 3.3: Fuente física para lograr un speedup superlinear. Por ejemplo, la población de un EA no coge en una única caché, pero cuando se paraleliza, al disminuir el tamaño de la población de cada algoritmo, si coge, pudiendo producir valores superlineales de speedup.

Otras Métricas Paralelas

Aunque el speedup es la medida más utilizada, también se han definido otras métricas que pueden ser útiles para medir el comportamiento del algoritmo paralelo.

La *eficiencia* (Ecuación 3.3) es una normalización del speedup y permite comparar diferentes algoritmos (el valor $e_m = 1$ se alcanza cuando el speedup es lineal).

$$e_m = \frac{s_m}{m} \tag{3.3}$$

Existen múltiples variantes de esta métrica. Por ejemplo, la *eficiencia incremental* (Ecuación 3.4) muestra la mejora en el tiempo al añadir un nuevo procesador, y es muy útil cuando el tiempo de ejecución en un único procesador es desconocido y no se puede calcular de forma sencilla. Esta medida fue más tarde generalizada (Ecuación 3.5) para medir la mejora conseguida al pasar de una ejecución con n procesadores a una con m unidades de proceso.

$$ie_m = \frac{(m-1) \cdot E[T_{m-1}]}{m \cdot E[T_m]} \tag{3.4}$$

$$gie_{n,m} = \frac{n \cdot E[T_n]}{m \cdot E[T_m]} \tag{3.5}$$

Las métricas que acabamos de explicar indicar la mejora que se produce cuando se incrementa la capacidad de cómputo, pero no mide la utilización de la memoria disponible. El *speedup de escalado* (Ecuación 3.6) trata con este aspecto y nos permite medir la utilización de todos los recursos de las máquinas:

$$ss_m = \frac{Tiempo\ estimado\ para\ resolver\ un\ problema\ de\ tamaño\ nm\ en\ un\ procesador}{Tiempo\ para\ resolver\ un\ problema\ de\ tamaño\ nm\ en\ m\ procesadores} \tag{3.6}$$

donde n es el tamaño del problema más grande que puede ser almacenado en la memoria de un procesador. La principal dificultad de este medida es que realizar una estimación del tiempo de la ejecución secuencial puede ser complicado e impracticable en muchos casos.

Muy relacionada con esta última métrica tenemos la *escalabilidad* que ha diferencia de la anterior no necesita estimar ningún tiempo:

$$su_{m,n} = \frac{Tiempo\ para\ resolver\ k\ problemas\ en\ m\ procesadores}{Tiemp\ para\ resolver\ nk\ problemas\ en\ nm\ procesadores} \tag{3.7}$$

Esta métrica mide la habilidad del algoritmo para resolver una tarea n veces más grande en un sistema paralelo n veces más grande que es sistema original. Por lo tanto según esta métrica el escalado lineal se produce cuando $su_{m,n} = 1$.

Finalmente, Karp y Flatt [142] desarrollaron una interesante métrica para medir el rendimiento de cualquier algoritmo paralelo que puede ayudarnos a identificar factores más sutiles que los proporcionados por el speedup por si sólo. Esta métrica se denomina *fracción serie* de un algoritmo (Ecuación 3.8).

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m} \tag{3.8}$$

Idealmente, la fracción serie debería permanecer constante para un algoritmo aunque se cambie la potencia de cálculo de la plataforma donde se ejecuta. Si el speedup es pequeño pero el valor de la fracción serie se mantiene constante para diferentes números de procesadores (m), entonces podemos concluir que la perdida de eficiencia es debida al limitado paralelismo del programa. Por otro lado, un incremento ligero de f_m puede indicar que la granularidad de las tareas es demasiado fina. Se puede dar un tercer escenario en el que se produce una reducción significativa en el valor de f_m, lo que indica que alguna clase de speedup superlineal.

3.2.2. Diseño de Experimentos

La meta general de cualquier investigación es presentar una nueva propuesta o algoritmo que trabaja mejor, en algún sentido, que el resto de los métodos existentes. Esto requiere que las nuevas técnicas sean comparadas con el resto de alguna forma. En general, es muy complicado realizar comparaciones totalmente justas e imparciales. Por ejemplo, son comunes los casos en los mismos resultados pueden aportar diferentes conclusiones dependiendo de cómo se utilicen las métricas y cuales se usen. Estos problemas son especialmente importantes en los métodos no deterministas como las metaheurísticas con la que trabajamos en esta tesis. En este apartado trataremos todos estos problemas y daremos una guía que nos sirva para diseñar nuestros experimentos e informemos de los resultados de la forma más adecuada. El esquema que vamos a seguir se detalla en la Figura 3.4.

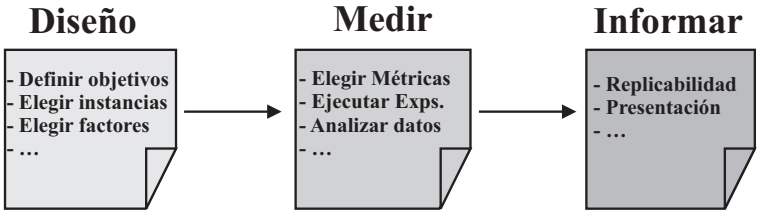


Figura 3.4: Pasos principales a la hora de planificar el diseño experimental.

Experimentación

La primera elección a la que debe enfrentarse el investigador es decidir que dominio e instancias va a utilizar para realizar los experimentos. Esta decisión depende de la meta de la experimentación. Se pueden distinguir dos objetivos claramente: (1) la resolución del problema y (2) el estudio del comportamiento del algoritmo

La resolución de problemas es el motivo básico por el que se diseñan nuevos algoritmos y el objetivo es batir a los demás en uno o varios problemas. Esta clase de investigación concluye generalmente estableciendo la superioridad de cierto método sobre el resto. Pero aparte de superar a los otros algoritmos, una actividad muy importante es investigar el *porqué* de esa superioridad. En [134] se describe en detalle este tipo de experimentación.

Una de las más importantes decisiones que se debe tomar es qué instancias van a ser utilizadas. El conjunto de instancias debe ser lo suficientemente complejo para que los resultados obtenidos sean interesantes y debe haber la suficiente variedad de escenario para que las conclusiones sean generalizables. Los generadores de problemas [139] son unos buenos candidatos ya que cumplenn ambas características. En los siguientes párrafos mostramos las principales clases de instancias que pueden ser utilizadas (una clasificación más completa y exhaustiva se puede encontrar en [86, 216]).

- **Instancias del mundo real:** Estas instancias son tomadas de problemas de aplicación real. Tristemente, es muy difícil obtener datos reales para realizar los experimentos y cuando se obtienen son escasos debido a consideraciones de carácter legal. Una alternativa consiste en crear variaciones aleatorias a partir de las reales, manteniendo la estructura del problema pero modificando aleatoriamente los detalles para producir una nueva instancia.

  Otro acercamiento es usar *instancias naturales* [86], que representan a instancias que surgen de situaciones del mundo real, como puede ser decidir el horario de una escuela. Esta clase de instancias tiene la ventaja de se puede disponer de ellas de forma gratuita.

- **Instancias estándar:** En esta clase están incluidos las instancias, bancos de prueba, y generadores de problemas que, debido a su amplia utilización en la experimentación, se han vuelvo estándar en la literatura especializada. Por ejemplo Reinelt [220] ofrece la *TSPLIB*, un conjunto de instancias estándar para el problema del viajante, y Demirkol *et al.* [83] ofrecen algo similar para los problemas de planificación de tareas. Tales librerías permiten probar aspectos específicos del algoritmo y también comparar nuestros resultados con respecto a otros métodos. La *OR Library* [44] es un excelente ejemplo de esta clase de instancias.

- **Instancias aleatorias:** Finalmente, cuando ninguno de las anteriores fuentes proporcionan instancias adecuadas para los tests, la alternativa que queda es la generación aleatoria de las mismas. Este método es una forma rápida de obtener un grupo diverso de instancias de prueba, pero también es el más controvertido.

Después de haber seleccionado el problema o grupo de instancias que se van a usar, hay que diseñar los experimentos a realizar. Generalmente, el diseño empieza con el análisis del efecto que tiene varios factores en el rendimiento del problema. Estos factores incluyen los del problema, como tamaño del problema, número de restricciones, etc. y los factores debido al algoritmos, como son los parámetros que lo configuran. Si el coste de los experimentos no es demasiado elevado, podemos hacer un diseño *factorial completo*, pero en general, esto no es posible debido al gran número de experimentos que supone. Por lo tanto generalmente se realiza un diseño *factorial fraccionario* que intenta obtener las mismas conclusiones que el anterior pero sin que sea necesario realizar todas las combinaciones [191].

Los siguientes pasos es ejecutar los experimentos planeados, elegir las métricas de rendimiento que van a ser utilizados y analizar los resultados. Estos pasos serán tratados en los próximos apartados.

Medidas de Rendimiento

El objetivo de las metaheurísticas es encontrar *buenas* soluciones en un tiempo *razonable*. Por lo tanto, la elección de las medidas de rendimiento para los experimentos con heurísticos necesariamente deben incluir tanto la calidad de las soluciones como el esfuerzo computacional necesario para alcanzarla. Debido a la naturaleza no determinista de las metaheurísticas, son necesarias realizar varias ejecuciones independientes para obtener unos resultados estadísticamente consistentes.

Calidad de las Soluciones Este es uno de los factores más importantes a la hora de evaluar el rendimiento de un algoritmo. Para instancias de problemas donde la solución óptima es conocida, es fácil definir una métrica para controlar este aspecto: el número de veces que se alcanza. Esta medida generalmente es definida como el porcentaje de veces que se alcanza la solución óptima respecto al total de ejecuciones realizadas. Desgraciadamente conocer la solución óptima no es un caso habitual para problemas realistas, o aunque se conozca, su cálculo es tan pesado computacionalmente, que lo que interesa es encontar una buena aproximación en un tiempo menor. De hecho, es común que los experimentos con metaheurísticos estén limitados a realizar a lo sumo un esfuerzo computacional definido de antemano (visitar un máximo número de puntos del espacio de búsqueda o un tiempo máximo de ejecución)

En estos casos, cuando el óptimo no es conocido, se suelen usar métricas estadísticas. La más populares son la media y la mediana del mejor valor de fitness encontrada en cada ejecución independiente.

En los problemas donde el óptimo es conocido, se pueden usar ambas métricas, tanto el número de éxitos como la media/mediana del fitness final (o también del esfuerzo). Es más, al usar ambas se obtiene más información: por ejemplo, un bajo número de éxito pero una alta precisión indica que raramente encuentra el óptimo pero que es un método robusto.

En la práctica, la comparación simple entre dos medias o medianas podría no dar el mismo resultado que comparar las dos distribuciones asociadas a los resultados de los algoritmos. En general, es necesario ofrecer otros datos estadísticos como la varianza o la desviación estándar y realizar un análisis estadístico global para asegurar que las conclusiones son significativas y no son sólo debidas a variaciones aleatorias. Este tema será tratado con más detenimiento en el último apartado de este capítulo.

Esfuerzo Computacional Otro aspecto clave a la hora de evaluar las metaheurísticas es la velocidad de computación con la que encuentra las soluciones. Dentro del campo de las metaheurísticas, el esfuerzo computacional es medido típicamente por el número de evaluaciones que realizan o/y el tiempo de ejecución. En general, el número de evaluaciones se define en término del número de puntos del espacio de búsqueda visitados.

Muchos investigadores prefieren el número de evaluaciones como manera de medir el esfuerzo computacional ya que elimina los efectos particulares de la implementación, del software y del hardware, haciendo así que las comparaciones sean independientes de esos factores. Pero esta medida puede ser engañosa en algunos casos en especial en el campo del cómputo en paralelo que es el que nos interesa. Por ejemplo, si algunas evaluaciones consumen más recursos que otras (un ejemplo típico de esto ocurre en la programación genética [148]) o si el cálculo es muy rápido comparado con otros procesos del algoritmo, entonces este valor no representa de forma correcta la

velocidad del algoritmo. También debemos tener en cuenta que la meta tradicional del paralelismo es reducir el tiempo de cómputo no el número de evaluaciones. Por lo tanto, los investigadores deben usar las dos métricas (evaluaciones y tiempo) para obtener una medida realista del esfuerzo computacional.

Informando de los Resultados

El paso final de la proceso experimental es documentar los detalles de los experimentos y los resultados obtenidos para comunicarlos a la comunidad internacional. En los próximos párrafos mostramos los principales puntos a tener en cuenta al realizar este paso.

- **Reproducibilidad:** Una parte que debería ser obligatoria en cualquier presentación es un comentario sobre como se han realizado los experimentos. La reproductibilidad es una parte esencial en la investigación científica, y debe permitir que los resultados puedan ser verificados de forma independiente por parte de la comunidad científica. Por lo tanto, los algoritmos y los aspectos de su implementación deben ser descritos con el suficiente detalle para permitir su replicación, incluyendo los parámetros (probabilidades, constantes, etc.), codificación del problema, generación de números pseudo-aleatorios, etc. También se debe documentar la fuente y las características de las instancias utilizadas. Finalmente, los factores del entorno de ejecución que puedan influir en el rendimiento también deben ser documentados: número, tipo y velocidad de los procesadores, tamaño y configuración de la memoria, tipo de red de comunicación existente, sistema operativo, etc.

- **Presentación de los Resultados.** El último detalle importante es la presentación de los resultados. La mejor manera de avalar las conclusiones que se exponen es mostrando los datos de alguna manera que enfatice la tendencia que exhiben. Hay muy buenas técnicas para mostrar los datos dependiendo de los puntos que se quieran destacar (véanse por ejemplo [61] o [255]).

  Las tablas por sí mismas son interesantes para mostrar los datos pero en general son insuficientes, ya que la gran cantidad de datos mostrados en ellas pueden confundir al lector. Por lo tanto, si hay alguna forma de resumir de forma gráfica los resultados, suele ser preferible aunque sea una representación más inexacta que los datos presentados en las tablas. Por otro lado, aunque con la imágenes o figuras se pueda dar una idea rápida de las conclusiones que se quieren mostrar, siempre deben venir acompañadas por las tablas con los datos exactos.

3.2.3. Análisis Estadístico

Como hemos comentado antes, en la mayoría de los trabajos el objetivo es probar que algoritmo supera a otro. Pero como ya vimos, la comparación de dos medias o medianas no es una medida concluyente y en muchos casos puede ser erróneas las conclusiones deducidas de esa comparativa. Por lo tanto, se deben utilizar métodos estadísticos que permitan confirmar que nuestras conclusiones son correctas.

Generalmente, los investigadores usan t-test o análisis de la varianza (ANOVA) para asegurar la *significancia estadística* de los resultados, es decir, para determinar si los efectos observados en las medias son reales o son debido a errores en el muestreo realizado. Para poder asegurar la correcto se deben aplicar diferentes métodos estadísticos de acuerdo con las condiciones de los datos experimentales obtenidos (ver la Figura 3.5). Inicialmente, hay que decidir si los tests que hay que realizar deben ser paramétricos o no paramétricos; cuando los datos son no normales o hay pocos datos experimentales (menor de 30 ejecuciones independientes) se deben usar test no paramétricos en otro caso los análisis paramétricos son suficientes. El test de Kolmogorov-Smirnov

es una herramienta estadística muy potente, precisa y de bajo coste para comprobar la normalidad de los datos. El t-test de Student es ampliamente usado para comparar datos, cuando estos son normales y sólamente tenemos dos colecciones de datos a comparar. En otro caso, se debe aplicar primero un test de ANOVA y posterior test para comparar y ordenar las medias. En caso de que los datos no sean normales se han propuesto una serie de métodos que se pueden ver en la Figura 3.5, aunque ya son más complejos en su aplicación e interpretación de resultados. Todos esos métodos asumen algunas hipótesis para que se puedan aplicar de forma adecuada, por ejemplo, estos análisis asumen una relación lineal entre las causas y los efectos.

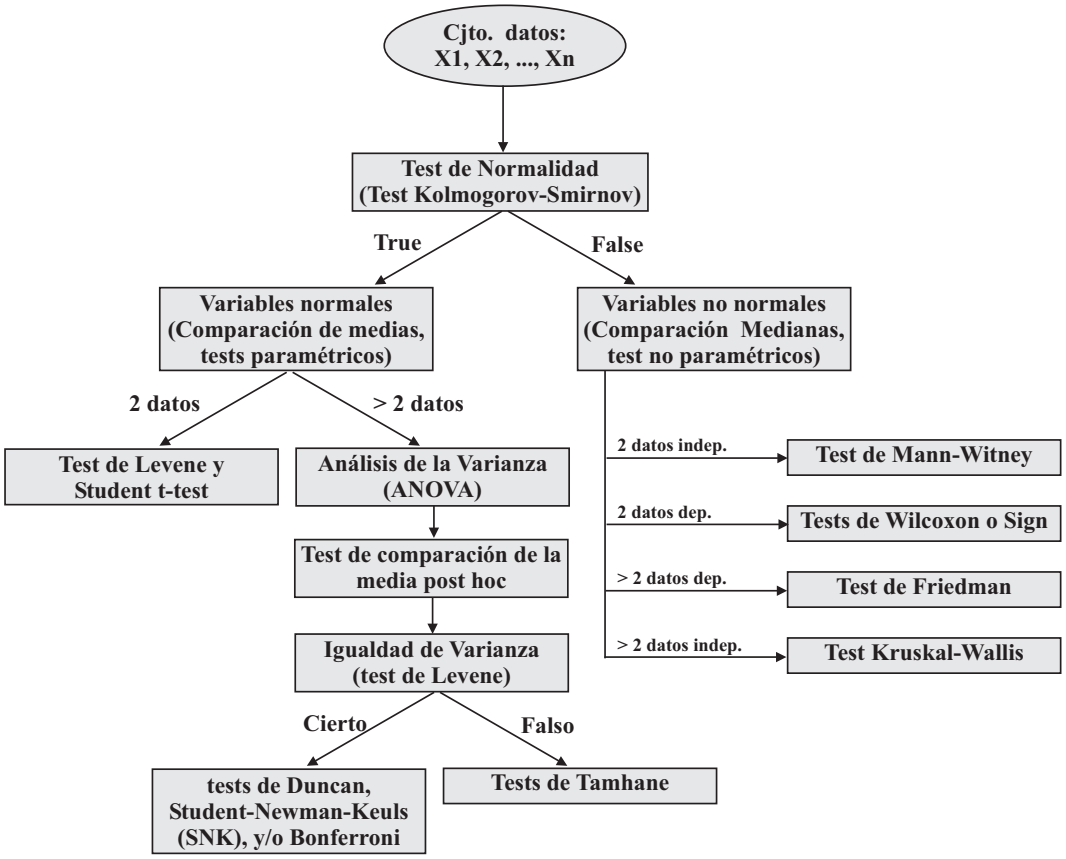


Figura 3.5: Esquema de aplicación de los métodos estadísticos.

Tanto el t-test como ANOVA sólo pueden ser aplicados si las distribuciones a comparar son normales. En el campo de las metaheurísticas, la no normalidad de la distribuciones asociadas a los resultados puede darse con bastante facilidad. Para estos casos, hay un teorema que nos puede ayudar. El *Teorema Central del Límite* dice que la suma de muchas distribuciones idénticas de variable aleatoria tienden a una distribución Gausiana. Debido a ese teorema se puede concluir que la media de un conjunto de muestras tiende a una distribución normal. Pese a eso, en muchos casos este teorema no es aplicable [120]. En el caso de poder asegurar la normalidad de los datos debemos utilizar otros métodos como puede ser el análisis de Kruskal-Wallis.

3.3. Conclusiones

Este capítulo tiene dos grandes partes. En la primera se discute acerca del diseño e implementación software. Este es un aspecto importante en especial para las versiones paralelas. Por ello, hemos descrito los requisitos que debe tener un software de estas características, haciendo hincapié en la necesidad de utilizar las metodologías de la Ingeniería del Software. También hemos analizado las principales herramientas con las que disponemos en la actualidad para el desarrollo de software paralelo, mostrando sus principales características. Terminamos esta sección, con nuestra propuesta que sigue todas las recomendaciones que hemos indicado antes, siendo desarrollada siguiendo la metodología orientada a objetos y utilizando el patrón software de esqueletos de código.

En la segunda parte del capítulo hemos tratado otro aspecto muy importantes dentro de la investigación utilizando metaheurísticas en general y sus versiones paralelas en concreto. Hemos descrito los principales aspectos que se deben tener en cuenta a la hora de realizar un buen diseño experimental. También hemos definido de forma específica las principales métricas que se deben utilizar a la hora de analizar los resultados, indicando cuando se deben utilizar cada una y la información que se puede extraer de ella. Finalmente, se han descrito los análisis estadísticos que se deben realizar para asegurar que los resultados son consistentes y las conclusiones correctas.

En resumen en este capítulo hemos dado dos guías; una primera de indica los aspectos a tener en cuenta en el diseño e implementación de una metaheurísticas y cómo realizarlo para que el resultado sea un código eficiente pero bien estructurado y reusable. En la segunda guía metodológica hemos indicado cómo diseñar los experimentos y analizar los resultados de forma que las conclusiones que se puedan extraer de ellas sean correctas e interesantes.

Capítulo 4

Modelos Paralelos Propuestos

Tras el estudio de los modelos paralelos existentes, en este capítulo definiremos los que proponemos y usamos a lo largo del resto de la tesis. Para ello inicialmente propondremos una formalización matemática que permita caracterizar los diferentes modelos propuestos para las metaheurísticas y a partir de él definiremos nuestras propuestas. En concreto nos centramos en los modelos paralelos que modifican la dinámica respecto a la versión secuencial canónica de la metaheurística, ya que inicialmente no sólo se buscan algoritmos más eficientes sino también más eficaces.

4.1. Modelos Previos

En la actualidad ya existen varios modelos matemáticos caracterizar a parte de los algoritmos englobados en el campo de las metaheurísticas.

Por ejemplo Sprave en 1999 [239] propuso una descripción unificada para cualquier tipo de EA estructurado, que incluso podría acabar en un modelo bastante preciso para poblaciones panmícticas (ya que puede considerarse como un modelo estructurado totalmente conectado). Sprave modeló la estructura de la población por medio de *hipergrafos*. Un hipergrafo es una extensión del grafo canónico. La idea básica de los hipergrafos es la generalización de los ejes tradicionales que sólo tiene un par de vértices a un conjunto de arbitrario de vértices (véase un ejemplo en la Figura 4.1a).

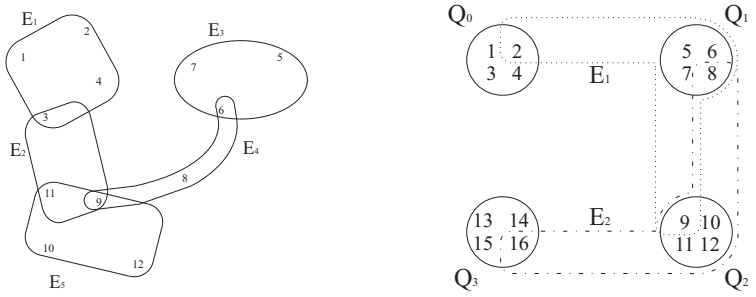


Figura 4.1: Dos hipergrafos: (a) un hipergrafo básico siendo $X = \{1, \ldots, 12\}$ los vértices y $\{E_1, \ldots, E_5\}$ los ejes y (b) un hipergrafo que representa un dEA de cuatro islas y topología en anillo (E_0 y E_3 se han omitido para dar claridad al gráfico).

DEFINICIÓN **3** *Un hipergrafo para describir un EA descentralizado está caracterizado por:*

$$
\begin{array}{lcll}
\Pi & = & (X, \mathcal{E}^t, Q) & \\
X & = & (0, \ldots, \lambda - 1) & \\
Q_i & = & (i\nu, \ldots, i\nu + \nu - 1) & \\
\mathcal{E}^t & = & (E_0^t, \ldots, E_{r-1}^t) & \\
E_i^t & = & \begin{cases} Q_i \cup \bigcup_{(s,i)\in G} M_{s\to i} & \textit{para } t \equiv 0 \textit{ mod } \eta \\ Q_i & \textit{en otro caso} \end{cases} & \\
\multicolumn{4}{l}{\textit{con } M_{s\to i} \subset Q_i}
\end{array}
$$

Este hipergrafo representa un EA general multipoblacional con un periodo de migración η, donde X denota la población completa, Q_i representa la subpoblación i, $M_{s\to i}$ es el conjunto de individuos que migran desde la subpoblación s a la i, E_i^t son los padres potenciales en el paso generacional t de los individuos de la i-ésima subpoblación, y finalmente $\mathcal{E}^t$ es el número de todos los E_i^t en el paso t. En la Figura 4.1b, se puede ver un ejemplo de un hipergrafo que representa un dEA.

Quizás el mayor problema de este modelos es que puede ser demasiado específico, limitando los esquemas paralelos que se pueden definir con él y también que es complicado extenderlo para metaheurísticas basadas en trayectoria.

También otro modelo interesante es el de los sistemas adaptativos granulado [74]. Este modelo es muy genérico y se basa en la siguiente definición de sistema adaptativo:

DEFINICIÓN **4** *Definimos un sistema adaptativo como una tupla*

$$\mathcal{S} = (A, \Omega, \tau, B, \Omega_B, \tau_B, E)$$

donde

- $A = \{A_1, A_2, \ldots\}$ *es el conjunto de estructuras con las que trabaja* $\mathcal{S}$.
- $\Omega = \{\omega_1, \omega_2, \ldots\}$ *es un conjunto de operadores para modificar estructuras de* A.
- $\tau : E \to \Omega$ *es un plan adaptativo para determinar qué operador aplicar dada la entrada* E *en un instante determinado.*
- $B = \{B_1, B_2, \ldots\}$ *es el conjunto de artefactos manejados por* $\mathcal{S}$.
- $\Omega_B = \{\omega_{B1}, \omega_{B2}, \ldots\}$ *es un conjunto de operadores para modificar artefactos* B.
- $\tau_B : E \to \Omega_B$ *es un plan adaptativo para determinar qué operador aplicar dada la entrada* E *en un instante determinado.*
- E *es el conjunto de posibles entradas al sistema (problema que resolver).*

Este sistema adaptativo es un modelo muy general y que comparte algunas características con el que nosotros presentamos en la Definiciones 1 y 2, siendo una de las diferencias más importante la distinción que hace este modelo entre el genotipo (A) y fenotipo (B). El genotipo es la representación interna del fenotipo. Utilizando el concepto de sistema adaptativo, realizan un modelo para sistemas paralelos mediante la definición de lo que denominan sistema adaptativo granulado:

DEFINICIÓN **5** *Se define un sistema adaptativo granulado como una tupla:*

$$\Xi = (\Delta, \Pi, \xi)$$

donde:

- $\Delta = \{\Delta_1, \Delta_2, \ldots, \Delta_m\}$ *es un conjunto de gránulos, cada uno consistente en un sistema adaptativo* $\Delta_i = (A^i, \Omega^i, \tau^i, B^i, \Omega^i_B, \tau^i_B, E^i)$
- Π *es la política de activación de los gránulos.*
- $\xi : \Delta \times \Delta \rightarrow 2^B$ *es una función que controla la comunicación entre gránulos (que artefactos son compartidos entre ellos).*

Por tanto, un sistema adaptativo granulado comprende varios gránulos, siendo cada uno a su vez un sistema adaptativo que usa sus propias estructuras y lleva a cabo su propio plan adaptativo. Las conexiones entre gránulos se especifican mediante ξ, mientras que el flujo de control queda determinado por Π. Este flujo puede ser secuencial $\Delta_i \Delta_j$ o concurrente $\Delta_i || \Delta_j$.

El principal inconveniente de este modelo es que muy general y aunque permite definir los principales esquemas paralelos que pueden utilizarse a la hora de diseñar una metaheurística paralela, no introduce en su formulación los detalles más específicos con lo que es difícil caracterizar ciertas aspectos de esquemas paralelos muy concretos, como por ejemplo el sincronismo en las comunicaciones, u operaciones de variación que involucran a estructuras de diferentes sistemas granulados.

4.2. Modelo Propuesto

A continuación presentamos nuestro modelo para caracterizar los sistemas paralelos que se basa en el concepto del metaheurística presentada en las Definiciones 1 y 2. Nos interesaba un modelo lo suficientemente genérico para poder contener cualquier esquema de paralelización y a la vez, algo que sea lo suficientemente concreto para poder extraer conclusiones interesantes a partir de él. Para diseñar nuestro modelo extraímos las características más interesantes de los modelos previos presentados en la sección anterior y proponer uno más acorde a nuestras necesidades.

DEFINICIÓN **6** *Definimos un sistema metaheurístico paralelo como una tupla:*

$$\mathcal{P} = \langle \mathbf{M}, \lambda, \Psi, \gamma, \chi, \Upsilon \rangle$$

donde:

- $\mathbf{M} = \{\mathcal{M}_0, \ldots, \mathcal{M}_{m-1}\}$ *es un conjunto de metaheurísticas* $\mathcal{M}_i = \langle \Theta^i, \Phi^i, \sigma^i, \mu^i, \lambda^i, \Xi^i, \tau^i \rangle$ *que están evolucionando de forma concurrente.*
- $\lambda : \mathbf{M} \times \Upsilon \rightarrow 2^\Theta$ *una función que controla la vecindad de cada metaheurística, indicando en que estructuras pueden influir en la metaheurística tratada, lo que refleja de forma implícita la topología.*
- $\Psi = \{\psi_1, \psi_2, \ldots\}$ *son un conjunto de operadores que trabajan a sobre las estructuras de diferentes metaheurística. Cada operador es definido como* $\psi_i : 2^\Theta \times \Upsilon \rightarrow 2^\Theta$.
- $\gamma : \Theta \times \Xi \times \Upsilon \rightarrow \{true, false\}$ *es una función de activación, indicando si una metaheurística puede interactuar con las demás dependiendo de su estado interno.*

- $\chi \in \{síncrono, asíncrono\}$ *indica si la cooperación entre las metaheurísticas se produce de forma síncrona o no. Si es síncrona, cuando una metaheurística* $\mathcal{M}_i$ *cumple la condición* $\gamma(\Theta^i, \Xi^i)$*, no avanza su evolución ni aplica los operadores de* Ψ *hasta que se cumpla* $\gamma(\Theta^j, \Xi^j)$ $\forall j \in \{0, \ldots, m-1\}$.

- $\Upsilon = \{\upsilon_1, \upsilon_2, \ldots\}$ *es un conjunto de variables de estado compartido de forma global o parcial por las metaheurísticas.*

Este modelo comparte similitudes con los dos modelos previos; por ejemplo, incluye explícitamente una función para indicar cuándo se produce la cooperación entre los componentes como el modelo de hipergrafos, aunque en este caso es más genérico, ya que esta decisión no está controlada únicamente por el paso de evolución o tiene una función de vecindad, muy similar a la función ξ de los sistemas adaptativos granulados.

Con este modelo conseguimos los dos objetivos que buscábamos, por un lado es general permitiendo caracterizar cualquier esquema paralelo para metaheurísticas (véase la siguiente sección) y por otro, al estar diseñado específicamente para metaheurísticas paralelas, incluye los suficientes detalles para poder caracterizar las diferencias entre los esquemas paralelos de forma simple.

4.3. Formalización de los Esquemas Paralelos Utilizados

A continuación, veremos como podemos modelizar los esquemas paralelos propuestos en esta tesis y que serán utilizados en los próximos capítulos usando este modelo matemático.

4.3.1. Esquema Distribuido

En este esquema tenemos múltiples metaheurísticas ejecutándose en paralelo y que cooperan entre si intercambiando información (Figura 4.2).

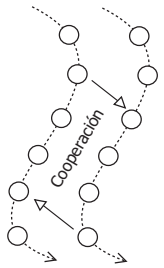


Figura 4.2: Esquema paralelo distribuido.

Para describir este modelo debemos indicar la política de migración que queremos utilizar. En concreto los esquemas utilizados una topología en anillo unidireccional, una frecuencia de migración f, un esquema de selección de emigrantes s y un esquema de reemplazo para los inmigrante r. Este esquema se caracteriza mediante la tupla:

$$\mathcal{P}_d = \langle \mathbf{M}_d, \lambda_d, \Psi_d, \gamma_d, \chi_d, \Upsilon_d \rangle$$

donde:

- No se incluye ninguna restricción sobre $\mathbf{M}_d$, únicamente se requiere que $t_i \in \Xi_i$ $\forall i \in \{0, \ldots, m-1\}$.

- $\lambda_d(\mathcal{M}_i) = \Theta_{i+1 \ mod \ m}$.
- $\Psi = \{\psi_1\}$, donde $\psi_1(\Theta_i, \Theta_j) = r(\Theta_j \cup s(\Theta_i))$.
- $\gamma_d(\Theta_i, \Xi_i) = \begin{cases} true & si \ t_i \equiv 0 \ mod \ f \\ false & en \ otro \ caso \end{cases}$

Como se ve en esta caracterización, no se ha establecido ninguna restricción sobre las metaheurísticas a la que se aplica, y de hecho hemos empleado este modelo tanto métodos basados en población (GA, CHC o SS) como en métodos basados en trayectoria (SA).

4.3.2. Esquema Distribuido Celular

Este esquema se puede considerar como un caso particular del anterior. En este caso tenemos un esquema distribuido como el anterior pero el comportamiento global es igual que un esquema celular (véase Sección 2.2.2). En la Figura 4.3 se muestra gráficamente este esquema paralelo.

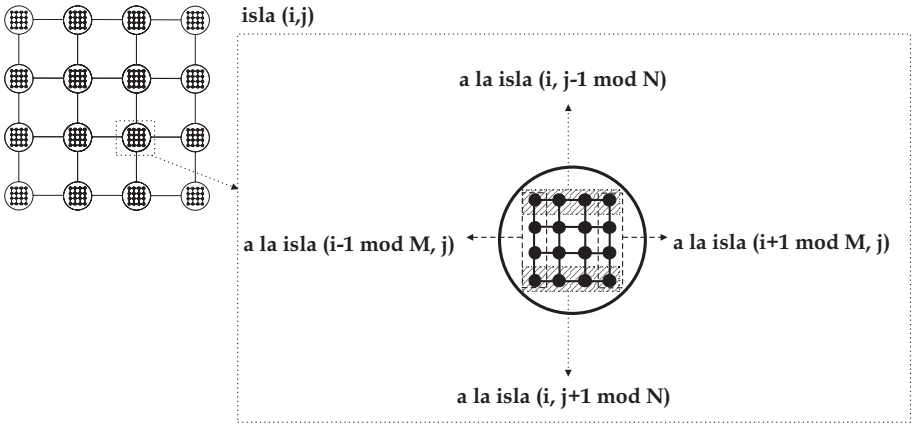


Figura 4.3: Esquema paralelo distribuido celular.

Este esquema puede ser caracterizado por la siguiente tupla:

$$\mathcal{P}_{dc} = \langle \mathbf{M}_{dc}, \lambda_{dc}, \Psi_{dc}, \gamma_{dc}, \chi_{dc}, \Upsilon_{dc} \rangle$$

donde:

- Para aplicar este modelo, se debe establecer una restricción sobre los $\mathcal{M}_i \in \mathbf{M}_{dc}$. En concreto se debe cumplir que $\mu_0 = \mu_2 = ... = \mu_{m-1} = \mu = \mu' \cdot \mu''$. Para simplificar los cálculos posteriores supondremos $m = N \cdot M$ y nos referiremos a los metaheurísticas $\mathcal{M}_k$ como $\mathcal{M}_{i,j}$, donde $i = k \ div \ N$ y $j = k \ mod \ N$.
- $\lambda_{dc}(\mathcal{M}_i) = \{\Theta_{i+1,j}, \Theta_{i-1,j}, \Theta_{i,j+1}, \Theta_{i,j-1}\}$.
- $\Psi = \{\psi_1\}$ donde:

$$\psi_1(\Theta_{i,j}, \Theta_{i+1,j}, \Theta_{i-1,j}, \Theta_{i,j+1}, \Theta_{i,j-1}) = \begin{array}{lll} \{ & \{\theta^1_{i,j}, \theta^{\mu'+1}_{i,j}, \ldots, \theta^{(\mu''-1)\cdot\mu'+1}_{i,j}\} & \cup \Theta_{i-1,j}\} \quad , \\ & \{\theta^{\mu'}_{i,j}, \theta^{2\cdot\mu'}_{i,j}, \ldots, \theta^{\mu}_{i,j}\} & \cup \Theta_{i+1,j}\} \quad , \\ & \{\theta^1_{i,j}, \theta^2_{i,j}, \ldots, \theta^{\mu'}_{i,j}\} & \cup \Theta_{i,j-1}\} \quad , \\ & \{\theta^{\mu''}_{i,j}, \theta^{\mu''+1}_{i,j} \ldots, \theta^{\mu}_{i,j}\} & \cup \Theta_{i,j+1}\} \quad \} \end{array} .$$

- $\gamma(\Theta_i, \Xi_i) = true$.
- $\chi = síncrono$.

4.3.3. Esquema Maestro/Esclavo

En este esquema tenemos un componente distinguido (maestro) que es el que genera tareas que son transferidas al resto de componentes (esclavos), que tras completarlas se las vuelven a pasar al maestro (Figura 4.4).

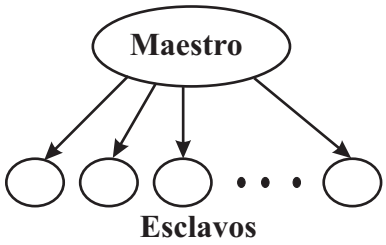


Figura 4.4: Esquema paralelo maestro/esclavo.

Este esquema puede ser caracterizado por la siguiente tupla:

$$\mathcal{P}_{ms} = \langle \mathbf{M}_{ms}, \lambda_{ms}, \Psi_{ms}, \gamma_{ms}, \chi_{ms}, \Upsilon_{ms} \rangle$$

donde:

- En este caso distinguimos el $\mathcal{M}_0$ como maestro con $\mu_0 = \mu$ y el resto $\mathcal{M}_i$ $(i > 0)$ como esclavo con $\mu_i = 1$. Por simplificar el resto de los cálculos supondremos que $m = \mu + 1$, aunque el resultado es extrapolable para cualquier $m > 1$.
- $\lambda_{ms}(\mathcal{M}_0) = \{\Theta_i | i > 0\}$ y $\lambda_{ms}(\mathcal{M}_i) = \{\Theta_0\}$ $\forall i > 0$.
- $\Psi = \{\psi_1\}$, donde $\psi_1(\Theta_0, \Theta_1, \ldots, \Theta_{m-1}) = \{\{\theta_0^1\}, \{\theta_0^2\}, \ldots, \{\theta_0^\mu\}\}$ y $\psi_1(\Theta_i, \Theta_0) = \{\theta_0^1, \ldots, \theta_0^{i-1}, \theta_i^1, \theta_0^{i+1}, \ldots, \theta_0^\mu\}$ $\forall i > 0$.
- $\gamma(\Theta_i, \Xi_i) = true$.
- $\chi = asíncrono$.

4.4. Conclusión

En este capítulo hemos definido un modelo matemático que permite caracterizar de forma bastante precisa los diferentes esquemas paralelos que existen para las metaheurísticas.

También hemos mostrado como caracterizar los esquemas paralelos que hemos utilizado y propuesto a lo largo de la tesis según este modelo. Cada uno de los esquemas presentados tienen sus características propias que lo hacen más apropiado para diferentes tipos de problemas o arquitecturas paralelas.

Parte II

Análisis y Formalización

Capítulo 5

Análisis Experimental del Comportamiento de las Metaheurísticas Paralelas

En los capítulos previos, hemos realizado una introducción al campo de las metaheurísticas, los modelos paralelos existentes y los que proponemos en esta tesis, su diseño e implementación. Una vez conocidos en bastante profundidad, el siguiente paso es estudiar su comportamiento ante diferentes condiciones, como puede ser diferentes parametrizaciones o diferentes plataformas de cómputo, para con ese conocimiento poder diseñar algoritmos eficientes y eficaces para poder abordar problemas complejos y de utilidad en el mundo real. En este y el próximo capítulo abordaremos estos estudios del comportamiento aunque de diferente punto de vista, mientras que en este lo examinamos desde un punto de vista experimental, en el siguiente lo realizaremos desde un punto de vista teórico, esperando que utilizando ambos enfoques logremos conseguir una visión completa de la dinámica de estos algoritmos paralelos.

En este capítulo vamos a examinar el comportamiento de diferentes metaheurísticas al ser ejecutadas en diferentes plataformas paralelas y con diferentes parametrizaciones. Para hacer este análisis, hemos diferentes estudios experimentales con diferentes algoritmos, problemas y plataformas para asegurar una mínima generalidad de los resultados.

Primero, empezamos analizando el comportamiento cuando las metaheurísticas se ejecutan sobre tres plataformas paralelas de área local diferentes, cada una de ellas con sus características propias. Una vez realizado este análisis, extenderemos el estudio para abarcar también a redes de área extensa, que es un escenario que está cobrando un alto auge en los últimos tiempos con la aparición de Internet. Empezamos con un caso canónico que sirva de base para los estudios, después extendemos el escenario a varias configuraciones WAN, para completar el estudio con un estudio preliminar del comportamiento de un algoritmo paralelo utilizando un Grid de tamaño mediano.

5.1. Análisis Sobre Diferentes Clases de Redes de Área Local

En este apartado analizamos el comportamiento de un GA distribuido (dGA) (Sección 4.3.1) cuando se ejecuta en varias redes área local: una Fast Ethernet, una Gigabit Ethernet y un Myrinet. Las dos primeras redes son muy habituales y conocidas. Ambas tiene similares características,

pero la red Gigabit Ethernet multiplica la velocidad de la red Fast Ethernet por 10, pasando de 100 Mbps en Fast Ethernet a 1Gbps en la red Gigabit. Myrinet [49] es una tecnología de altas prestaciones y que también se está extendiendo ampliamente como mecanismo de interconexión entre clústers de máquinas en centros científicos. Las características que distinguen a la Myrinet de otras redes incluyen una comunicación full-duplex de 1 Gps, control de flujo y una baja latencia. En este estudio nos enfocamos en la influencia de los parámetros más importantes de la política de migración (periodo de migración y ratio de migración) en el comportamiento de un dGA cuando se ejecuta en diferente plataformas de área local.

Para hacer este estudio, utilizamos el problema de ensamblado de fragmentos de ADN (los detalles concretos de este problema se pueden obtener en el Capítulo 8). Nuestro objetivo era hacer las pruebas en un escenario real con una instancia compleja para obtener un resultado que pueda ser interesante a la hora de resolver problemas de interés real.

En los experimentos usamos diferentes valores para el periodo de migración (1, 2, 16, 128, 512 y 1024 generaciones) y ratio de migración (1, 2, 16, 32 y 64 individuos). Debemos aclarar que un bajo periodo (por ejemplo, 1) significa un alto acomplamiento entre los diferentes subalgoritmos, mientras un valor alto (por ejemplo, 64) indica una ejecución con poca comunicación entre los subalgoritmos. La migración en el dGAs ocurre de forma asíncrona en una topología de anillo unidireccional, enviando una solución elegida de forma aleatoria a su vecino. La población destino incorpora estos individuos sólo si es mejor que la peor solución existente en la población. En la primera fase de estos experimentos, usaremos un $(\mu + \mu)$-dGA con 8 islas (una isla por procesador). En la fase siguiente, haremos el análisis cambiando el número de islas/procesadores a 4 y 2. El resto de los parámetros se muestran en la Tabla 5.1. En estos experimentos, el entorno software y hardware donde se ejecutan los algoritmos tiene una gran influencia, por lo que se describen en la Tabla 5.2. Debido a la naturaleza estocástica del dGA, hemos realizado 30 ejecuciones independientes para cada prueba para reunir datos lo suficientemente significativos para extraer conclusiones de ellos.

Tabla 5.1: Parámetros utilizados.

Parámetros	**Valores**
Tamaño de las subproblaciones	1024/Número de procesadores
Representación	Permutación (442 enteros)
Recombinación	OX ($\rho_c = 0{,}8$)
Mutación	Intercambio ($\rho_m = 0{,}2$)
Método de selección	Ranking
Ratio de migración	1, 2, 16, 32 y 64
Periodo de migración	1, 2, 64, 128, 512 y 1024

Tabla 5.2: Entorno software y hardware utilizado.

Parámetros	**Valores**
Procesador	PIV 2.4 GHz
Memoria Principal	256 MB
Versión de Linux	2.4.19-6 (SuSE)
Versión del g++	3.2
Versión de MPICH	1.2.2.3

5.1.1. Caso base: 8 procesadores

Las Tablas 5.3, 5.4 y 5.5 muestran los tiempos de ejecución medios para el dGA (con 8 islas), en las redes Fast Ethernet, Gigabit Ethernet y Myrinet, respectivamente. Debido a la gran cantidad de datos mostrados en las tablas que pueden confundir, hemos destacado varios valores importantes en la Figura 5.1.

Tabla 5.3: Tiempo medio de ejecución (en μsegundos) de un dGA con 8 procesadores sobre la red de área local Fast Ethernet.

		periodo					
		1	2	64	128	512	1024
	1	37269683	34967250	33963737	33799607	33890313	33836158
	2	37092247	35766270	34101287	33730040	33998213	33843180
ratio	16	58294070	47610630	35971320	33784687	33962590	33860780
	32	89803950	62581517	37543203	34065667	33904353	33915620
	64	175254844	98745559	40857867	34482773	33872323	33956670

Tabla 5.4: Tiempo medio de ejecución (en μsegundos) de un dGA con 8 procesadores sobre la red de área local Gigabit Ethernet.

		periodo					
		1	2	64	128	512	1024
	1	35842227	34923417	33772577	33681037	33676473	33826972
	2	36481183	35312083	33746657	33668543	33598087	33647733
ratio	16	51429240	44642810	33895067	33769983	33723950	33726360
	32	64471883	50845050	34250270	33989930	33709327	33690097
	64	80958070	59404767	34678460	34160270	33732820	33710255

Tabla 5.5: Tiempo medio de ejecución (en μsegundos) de un dGA con 8 procesadores sobre la red de área local Myrinet.

		periodo					
		1	2	64	128	512	024
	1	35888297	34778717	33666510	33765507	33720580	33716997
	2	36137960	35003820	33644037	33579163	33656493	33647457
ratio	16	50569180	45055760	33979783	33874037	33735097	33685963
	32	65492757	51465307	34434347	33982707	33754890	33694487
	64	83340960	60676630	34518957	34027937	33801083	33778647

Ahora procedemos a analizar los resultados mostrados en esas tablas y figuras. El primer detalle que se destacar, es que un periodo bajo produce un tiempo de ejecución mucho mayor que cuando se utilizan periodos más largos. Este comportamiento era esperado, ya que si se envían mensajes a través de red muy frecuentemente (se envía información en cada generación), la sobrecarga de la red es mayor y la sobrecarga debido a la comunicación (tratamiento de los mensajes) es mucho mayor, con lo que el tiempo total también se ve incrementado de forma muy alta. Ahora observamos un efecto equivalente (pero invertido) con el ratio de migración, los valores altos de este parámetro producen un tiempo de ejecución más alto. Tampoco este resultado es muy sorprendente, ya que si intercambiamos paquetes muy grandes con varios individuos en cada mensaje, la sobrecarga aumenta y por tanto también el tiempo total. Los tamaños medios de los paquetes utilizados son:

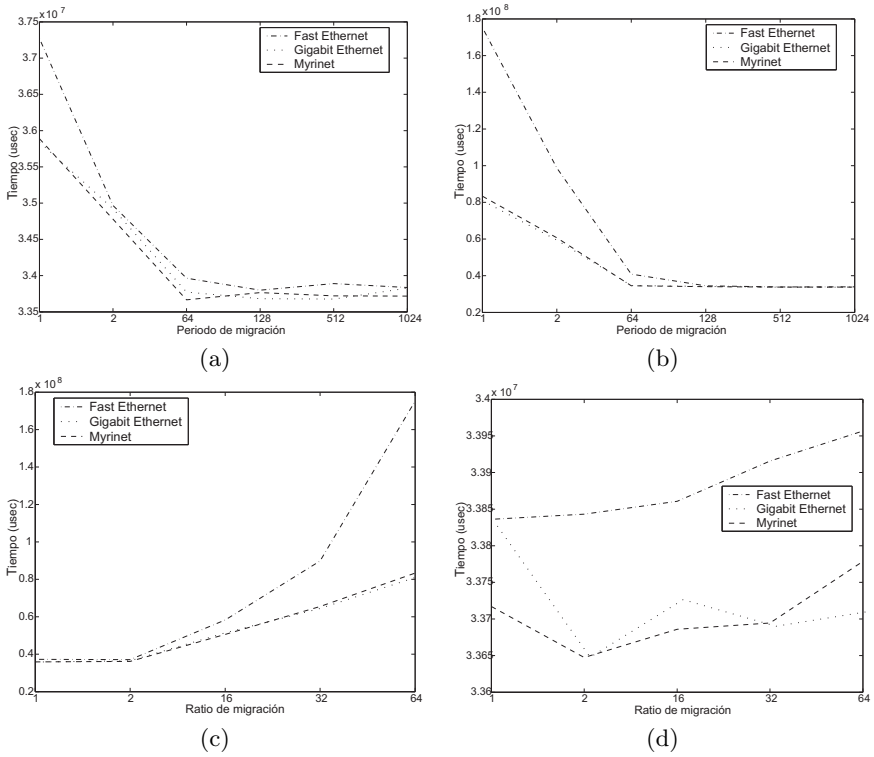


Figura 5.1: Representación gráfica del tiempo de ejecución (en μsegundos) de un dGA con (a) ratio de migración = 1, (b) ratio de migración = 64, (c) periodo de migración = 1 y (d) periodo de migración = 1024.

1.7 KB, 3.4 KB, 27.6 KB, 55.2 KB y 110 KB (para los ratios de migración = 1, 2, 16, 32 y 64, respectivamente).

Tras observar el comportamiento genérico, ahora lo analizamos los resultados por redes. Podemos observar que la red Fast Ethernet es la más lenta en todos los casos, aunque para configuraciones con baja comunicación (un alto periodo y un bajo ratio), donde el tiempo de ejecución es similar en todas las redes (por ejemplo, véase los valores de la derecha de las figuras 5.1(a) y 5.1(b)). En general, los tiempos recogidos utilizado la red Myrinet son menores que los de la Gigabit Ethernet cuando las comunicaciones entre las islas son escasas (un periodo mayor de 64 y un ratio menor de 16). Sin embargo, conforme las comunicaciones se hacen más intensas, la diferencia de tiempos entre las redes se va reduciendo. E incluso en el límite (es decir, con el periodo más bajo y el ratio más alto) se ha visto que las ejecuciones en la red Gigabit Ethernet es más eficiente que en la Myrinet. Pensamos que la razón de este comportamiento es debido al sistema operativo o la capa software de comunicación (MPI) que no permiten aprovechar todas las características de la red Myrinet. De todas formas las diferencias entre la Myrinet y Gigabit Ethernet no es demasiado grande para ninguna configuración.

Las Tablas 5.6, 5.7 y 5.8 muestra la proporción que supone la fase de comunicación sobre el tiempo total de ejecución en todas las redes. Confirmando las conclusiones de las tablas anteriores, observamos que no existe mucha diferencia en este valor entre las redes Gigabit Ethernet y Myrinet, mientras que la red Fast Ethernet tiene un porcentaje mayor en todas las configuraciones. Esto indica que la red Fast Ethernet pierde más tiempo en el intercambio de paquetes de datos que las

Tabla 5.6: Proporción entre el tiempo de comunicación y el tiempo total de de ejecución en la red Fast Ethernet.

		periodo					
		1	2	64	128	512	1024
	1	2.2673	1.5839	0.3195	0.14822	0.12907	0.11877
	2	3.3623	1.9471	0.3861	0.15295	0.1273	0.12076
ratio	16	32.2	21.951	4.9972	0.39005	0.25331	0.15037
	32	46.921	32.135	6.7842	1.0607	0.30336	0.21534
	64	62.818	48.974	12.015	1.8709	0.39075	0.30922

Tabla 5.7: Proporción entre el el tiempo de comunicación y el tiempo total de de ejecución en la red Gigabit Ethernet.

		periodo					
		1	2	64	128	512	1024
	1	2.1843	1.3771	0.23906	0.14338	0.12619	0.11838
	2	3.1538	1.7501	0.25413	0.15678	0.13078	0.12024
ratio	16	29.014	20.814	0.55587	0.31725	0.1659	0.13569
	32	42.069	28.958	1.7742	0.95777	0.31941	0.20433
	64	55.247	39.093	2.4848	1.2916	0.39944	0.2393

Tabla 5.8: Proporción entre el tiempo de comunicación y el tiempo total de de ejecución en la red Myrinet.

		periodo					
		1	2	64	128	512	1024
	1	2.7112	1.4757	0.18691	0.1439	0.12697	0.1175
	2	3.5726	1.9975	0.21191	0.16187	0.12788	0.11827
ratio	16	27.75	20.862	1.2862	0.73792	0.18074	0.16755
	32	41.047	28.37	1.6797	0.91508	0.34357	0.19961
	64	56.626	40.119	2.3893	1.2745	0.52415	0.2338

otras redes. Se puede observar que existe un comportamiento global independiente de la plataforma de ejecución: para valores bajos del ratio de migración (menor o igual a 2) y valores altos de periodo de migración (mayor o igual de 128), el tiempo de comunicación es casi despreciable (representa menos del 3 % del tiempo total de la ejecución). Sin embargo, en las configuraciones opuestas, o sea, aquellas que implican mayor comunicación, el tiempo de esta fase crece e incluso domina al tiempo de computación. Incluso, en algunas configuraciones, el algoritmo pasa más tiempo en la fase de comunicación (55-62 % del tiempo total de ejecución) que en el resto de las fases de forma conjunta.

Ahora vamos a realizar un análisis más profundo sobre los tiempos. Para esto vamos utilizar la medida más importante a la hora de analizar los tiempos de ejecución de los algoritmos paralelos, el *speedup*. En la Figura 5.2 comparamos el algoritmo paralelo (con 8 procesadores) con la versión secuencial canónica secuencial (que se denominó "speedup débil versus panmixia" en el Capítulo 3). Podemos observar que cuando dGA no tiene muchas comunicaciones, el speedup resultante es muy buena, incluso en algunos casos obteniendo valores super-lineal (véanse las figuras 5.2(b) y 5.2(c)). Sin embargo, el speedup decrece de forma rápida cuando usamos valores más altos de ratio de migración o un periodo de migración más bajo. Para las configuraciones con pocas comunicaciones, todas las redes obtienen un valor similar de speedup, pero en las otras configuraciones el algoritmo experimenta un considerable pérdida de eficiencia (véanse los valores de la derecha

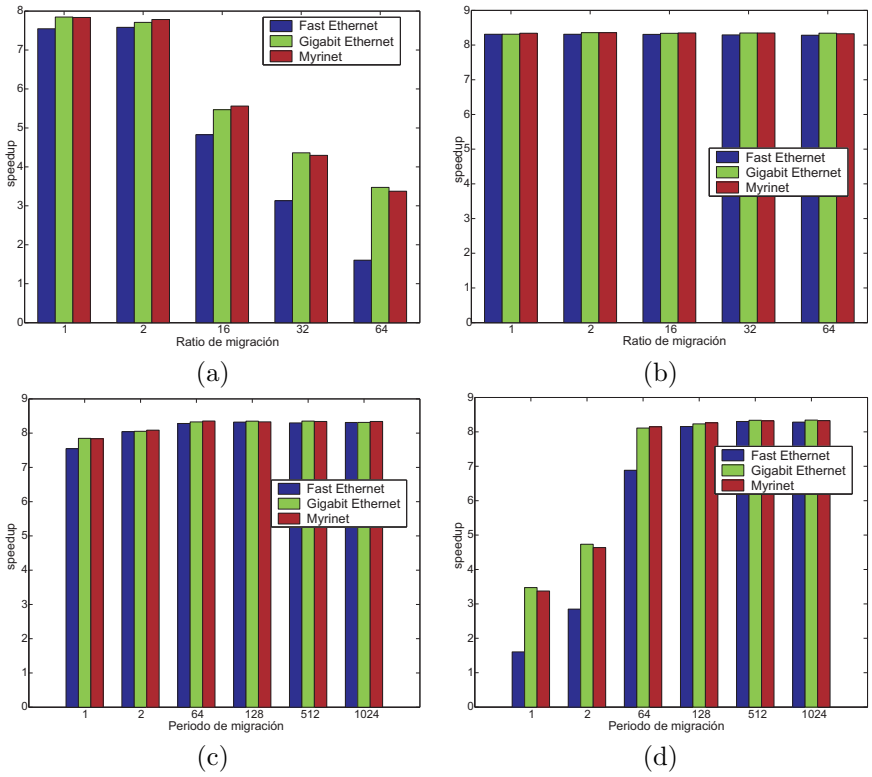


Figura 5.2: Speedup débil (versus panmixia) de un dGA con (a) ratio de migración = 1, (b) ratio de migración = 64, (c) periodo de migración = 1 y (d) periodo de migración = 1024.

de la Figura 5.2(a) y los valores izquierdos de la Figura 5.2(d)). Este perdida de eficiencia es más importante en la red Fast Ethernet que en las otras.

Aunque la comparación anterior es útil para examinar el comportamiento del modelo paralelo respecto al secuencial, hay que tener en cuenta que estamos comparando algoritmos claramente diferentes. En la Figura 5.3, realizamos una comparación más justa; esta vez compararemos los tiempos de ejecución del algoritmo paralelo con 8 islas ejecutado tanto en una plataforma con 8 procesadores, como en otra plataforma monoprocesadora (véase la definición de speedup ortodoxo en Capítulo 3). En este caso, también se puede llegar a una conclusión similar a las de la comparación anterior: configuraciones que producen pocas comunicaciones, obtienen un speedup lineal o cercano a él (previamente era super-lineal) y en otro extremo, las configuraciones con alta comunicación obtiene un speedup moderado pero mejor que el obtenido en los resultados del speedup no ortodoxo (Figura 5.2). Otra vez se da la situación que esta pérdida de eficiencia es más acusada en la red Fast Ethernet que en las otras redes.

5.1.2. Caso extendido: 2 y 4 procesadores

En los análisis precedentes hemos tratado con un algoritmo genético distribuido con ocho islas (y ocho procesadores). Ahora, extendemos este estudio para cubrir otro número de islas y procesadores. Nuestra intención es comprobar si las conclusiones que hemos extraido sobre el comportamiento del algoritmo, se mantienen cuando se modifica el número de procesadores a 4 o 2. Para simplificar el estudio, sólo analizaremos los casos extremos de los parámetros: ratio de

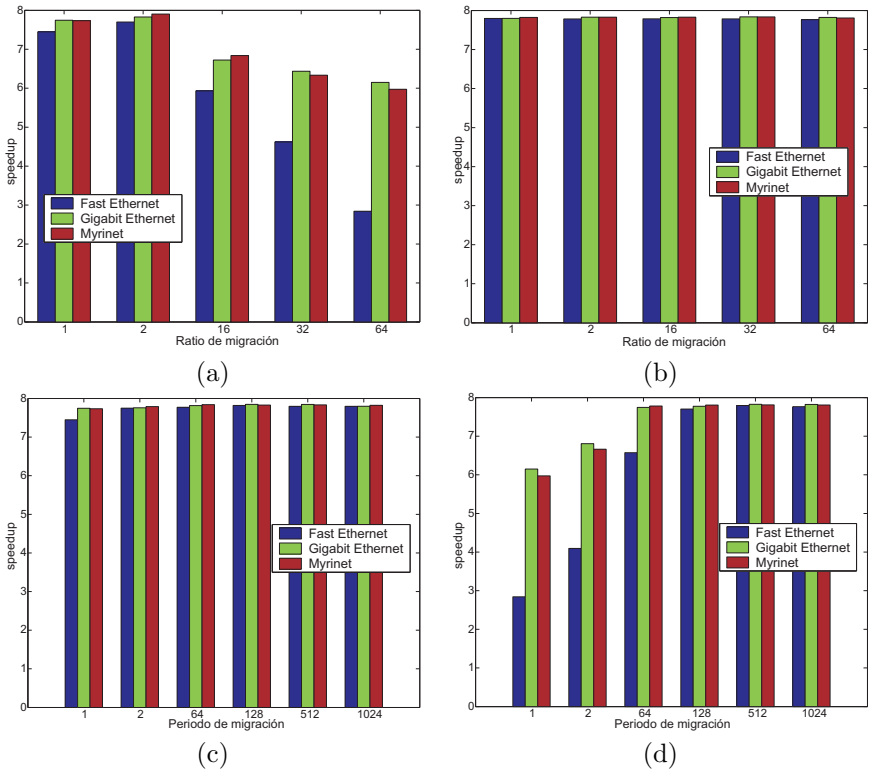


Figura 5.3: Speedup débil (ortodoxo) del dGA con (a) ratio de migración = 1, (b) ratio de migración = 64, (c) periodo de migración = 1 y (d) periodo de migración = 1024.

migración (1 y 64) y el periodo de migración (1 y 1024).

Como hicimos en el apartado anterior, en esta sección analizaremos el tiempo de ejecución, el coste de las comunicaciones y el speedup (ortodoxo) para todas las configuraciones y redes. Todos esos valores se muestran en la Figura 5.4. De todas esas figuras, podemos distinguir claramente dos comportamientos: el primero se produce cuando la fase de comunicación es ligera (parte derecha de las gráficas), donde la diferencia entre las redes es escasa, debido a las escasos mensajes enviados; el segundo comportamiento se produce para aquellas configuraciones que una fase de comunicación más costosa. En este segundo escenario las diferencias entre las redes son más importantes, especialmente entre la red Fast Ethernet y las otras redes. Como ocurría en los experimentos previos, las diferencias de tiempo (Figuras 5.4(a) y 5.4(b)) entre las redes sólo son significativas cuando el intercambio de información es frecuente y el tamaño de los mensajes es grande. Sin embargo esas diferencia entre las redes son más pequeñas cuando se disminuye el número de islas. Esto ocurre ya que debido al existir un menor número de procesadores que envíen mensajes por la red, las colisiones en la red Ethernet serán menores y por tanto el rendimiento se incrementa. El comportamiento del speedup (Figuras 5.4(e) y 5.4(f)) es también similar al mostrado anteriormente, es decir, cuando no existen apenas comunicaciones, el speedup es lineal, pero se deteriora conforme aumenta el uso de la red.

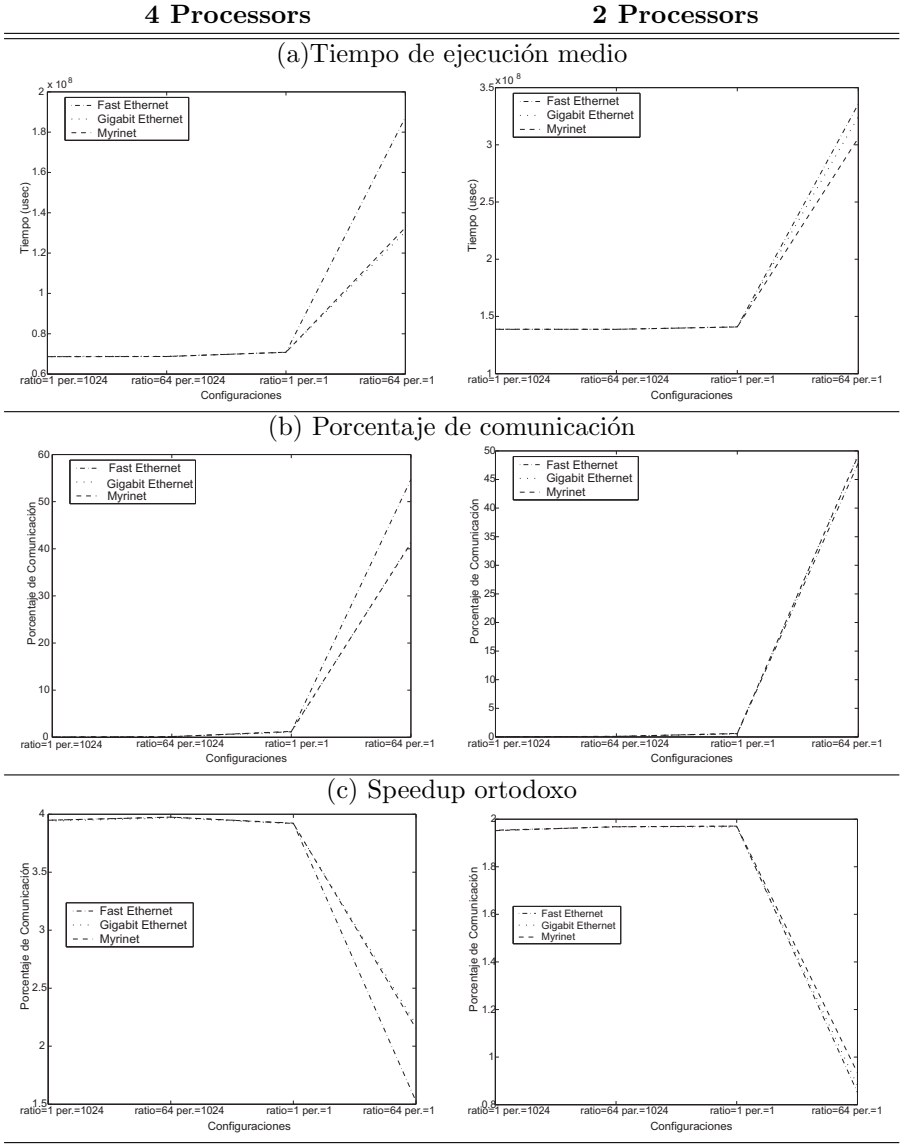


Figura 5.4: Resumen de todos los resultados del dGA con 2 y 4 procesadores.

5.1.3. Resumen

En este apartado vamos a resumir los resultados obtenidos en los experimentos anteriores. Hemos probado el comportamiento de un GA distribuido sobre tres redes de área local usando diferentes configuraciones de los parámetros. Hemos observado que el rendimiento del dGA sobre las diferentes redes dependen mucho del ratio de migración, el periodo de migración y el número de procesadores. El comportamiento del algoritmo varía dependiendo de la intensión de la fase de comunicación. Para fases de comunicación ligeras (es decir cuando se da alguna situación de las siguientes: un ratio de migración bajo, un periodo de migración alto, o un número de pequeño de procesadores) las diferencias entre las redes es escasa. Para el resto de configuraciones (las

que producen altas comunicaciones), la red Fast Ethernet rinde peor que las redes Myrinet y Gigabit. La Myrinet rinde ligeramente mejor que la red Gigabit Ethernet en las configuraciones con comunicación alta, mientras que la Gigabit es ligeramente mejor en los otros casos. Aunque, en general esas diferencia son muy pequeñas.

5.2. Análisis Sobre Redes de Área Extensa

En este apartado presentamos un amplio conjunto de resultados que nos permitan analizar el comportamiento de algunos heurísticos paralelos (tanto puros como híbridos) a la hora de resolver problemas de optimización. Concretamente nos enfocamos en varios algoritmos evolutivos y también en algún método basado en trayectoria como el enfriamiento simulado. Nuestra meta es ofrecer un primer estudio sobre los posibles cambios que se producen en el mecanismo de búsqueda de los algoritmos cuando pasamos de ejecutarlos en una red LAN a otra WAN. Para ello utilizaremos seis tareas de optimización de considerable complejidad.

Inicialmente ofreceremos una breve descripción de los modelos algorítmicos utilizados y los problemas que usamos como banco de pruebas, y finalizamos con los resultados obtenidos al ejecutarlos en varias plataformas paralelas; una primera sección donde definimos un caso canónico WAN y estudiamos su comportamiento respecto al caso LAN y una segunda sección donde se amplía es estudio con nuevos escenarios WAN

5.2.1. Algoritmos

En este trabajo hemos usado varios algoritmos evolutivos, en particular con algoritmos genéticos (GAs), un algoritmo CHC y con estrategias evolutivas (ES), y también mecanismos de búsqueda local como el enfriamiento simulado (SA). A partir de estos métodos hemos obtenido dos híbridos, uno que combina el algoritmo genético con el recocido simulado y otro en el que participan el CHC y ES. Los algoritmos puros ya fueron definidos en la Capítulo 2, por lo que pasamos a describir directamente las aproximaciones híbridas.

En este trabajo hemos implementado dos algoritmos híbridos débiles: uno que está formado por un algoritmo genético y el recocido simulado (GASA) y otro formado por el método CHC y una estrategia evolutiva (CHCES). La idea que nos ha movido a este diseño es que mientras que el GA o el CHC nos permiten movernos a buenas regiones del espacio de búsqueda (exploración), el SA o la ES nos permiten explotar en detalle esas regiones.

Para cada algoritmo híbrido se han realizado varios esquemas de hibridación:

- Una primera versión (GASA1/CHCES1) donde el algoritmo GA/CHC utiliza al otro algoritmo (SA/ES) como operador para ser aplicado tras las tradicionales operaciones de cruce y mutación, atendiendo a una cierta probabibabilidad de uso. Esto supone un esquema de hibridación coercitiva (ejemplo GASA1, Figura 5.5).

- Un esquema basado en el concepto de hibridación débil que ejecuta de manera continua un GA/CHC hasta terminar, selecciona individuos de la población final y ejecuta sobre ellos un algoritmo SA/ES. Se han implementado dos tipos de algoritmos con este esquema cuya diferencia reside en el criterio de selección para elegir los individuos sobre los que aplicar el segundo algoritmo. Concretamente, hemos realizado una versión (GASA2/CHCES2) que tiene como método de selección el torneo (esquema 2.1 de la Figura 5.6) y otra versión (GASA3/CHCES3) que utiliza una selección aleatoria (esquema 2.2 de la Figura 5.6).

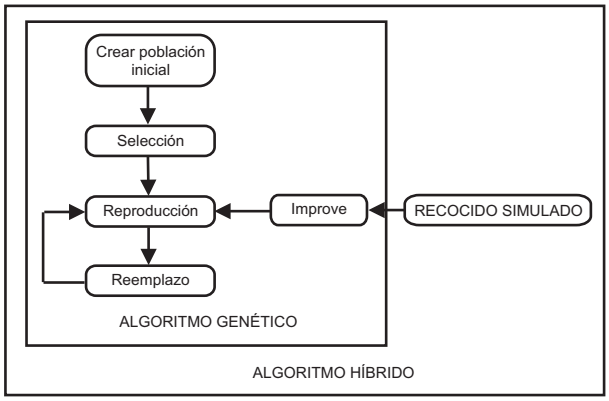


Figura 5.5: Esquema de hibridación 1 (GASA1).

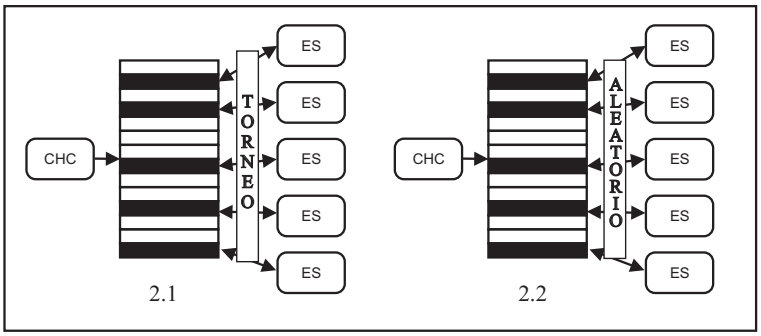


Figura 5.6: Esquema de hibridación 2 (CHCES2/3).

Debido al éxito constatable de los algoritmos híbridos, se está también apreciando su aparición en el dominio paralelo, como en el caso por ejemplo de [55] donde se estudia un algoritmo paralelo similar a GASA sobre el problema de tabla de horarios.

Versiones Paralelas

A la hora de paralelizar los EAs (ya sean puros o híbridos), hemos seguido un modelo distribuido (Sección 4.3.1). En este modelo un se ejecutan en paralelo pequeño número de EAs independientes, que periódicamente intercambian individuos para cooperar en la búsqueda global. Puesto que entre nuestros objetivos está comparar el comportamiento de este algoritmo con respecto a la versión secuencial, se ha configurado para que la población total del algoritmo paralela sea del mismo tamaño que la del algoritmo secuencial.

Para terminar de caracterizar nuestra propuesta de algoritmo genético distribuido para este problema, debemos indicar como se lleva a cabo la migración. Nuestra implementación usa como topología un anillo unidireccional, donde cada EA recibe información del EA que le precede y la envía a su sucesor. La migración se produce cada 20 generaciones, enviando una única solución seleccionada por torneo binario. La solución que llega sustituye a la peor solución existente en la población si es mejor que ella.

Para la paralelización del SA, hemos seguido el modelo de múltiples ejecuciones con cooperación (Capítulo 2). En concreto el SA paralelo (PSA) consta de múltiples SAs que se ejecutan de manera

asíncrona. Cada uno de estos SAs empiezan de una solución aleatoria diferente, y cada cierto número de evaluaciones (*fase de cooperación*) intercambian la mejor solución encontrada hasta el momento. El intercambio de información se produce en una topología de anillo unidireccional. Para el proceso de aceptación de la nueva solución inmigrante, se aplica el mecanismo típico del SA, es decir, si es mejor, se acepta inmediatamente y si es peor, puede ser aceptada dependiendo de cierta distribución de probabilidad que depende del fitness y la temperatura actual.

5.2.2. Problemas

Para analizar el comportamiento de los diferentes algoritmos en sus distintas versiones, hemos elegido los siguientes problemas: la minimización de la función de Rastrigin, el problema FMS (*Frequency Modulation Sounds Problem*), el problema del corte máximo (o *MaxCut*), el problema MTTP (*Minimum Tardy Task Problem*), el diseño de códigos correctores de errores y el enrutamiento de vehículos. Los dos primeros son problemas con representación continua, utilizado para estudiar el algoritmo ES y el híbrido CHCES. Los otros problemas son de tipo combinatorio y representan un amplio espectro de tareas en las áreas de teoría de grafos, planificación, teoría de codificación y transporte. La mayoría tiene un uso directo en el mundo real.

A continuación damos una breve descripción de alta nivel de cada uno de los problemas (se puede encontrar una definición más detallada de estos problemas en el Capítulo 7 y en [19, 20, 22]):

- **La función Generalizada de Rastrigin (RAS):** Este problema consiste en encontrar el mínimo a una función que tiene un espacio de búsqueda muy amplio con numerosos óptimos locales (función multimodal) [252].
- **Modulación de Frecuencia de Sonidos (FMS):** Este problema consiste en la identificación de ciertos parámetros para ajustar un modelo general $y(t)$ a uno concreto $y_0(t)$ [254].
- **Corte Máximo en un Grafo (MaxCut):** El problema consiste en dado un grafo ponderado, dividir el grafo en dos subgrafos disjuntos, de tal manera que la suma de los pesos de los ejes que unen los vértices de cada uno de los subgrafos entre sí sea máxima [241].
- **Problema de Tarea con Espera Mínima (MTTP):** El problema consiste en planificar unas tareas que minimice los pesos de las tareas no realizadas en su plazo límite [241].
- **Diseño de Código Corrector de Errores (ECC):** El problema consiste en la construcción de un código binario de tal forma que se minimice la longitud de los mensajes transmitidos, a la vez que se intenta proveer de la máxima capacidad de detección y/o corrección de eventuales errores que pueden aparecer durante la transmisión [104].
- **Enrutamiento de Vehículos (VRP):** El VRP consiste en un conjunto de cliente que deben ser servidos por una flota de vehículos desde un único depósito [152].

5.2.3. Resultados

En este aparatado describimos y analizamos los resultados obtenidos al resolver los problemas con los algoritmos presentados anteriormente. Este proceso lo realizaremos en tres fases; primero, analizamos el comportamiento del GA paralelo en entornos LAN y WAN. En esta fase estudiamos las prestaciones del GA paralelo a la hora de resolver los problemas combinatorios (MaxCut, MTTP, ECC y VRP). Después, incluimos una segunda fase donde se tratarán los problemas de dominio continuo. En esta segunda fase analizamos las estrategias evolutivas paralelas en nuestros dos problemas continuos (RAS y FMS). La última fase tiene como meta validar las conclusiones a

las que hemos llegado, pero esta vez considerando los algoritmos híbridos (las variantes del CHCES con el problema RAS y las variantes del GASA con MaxCut). Aplicaremos estas tres fases en dos etapas; primero compararemos los algoritmos en entornos LAN y WAN, y después, usaremos estas fases para estudiar el comportamientos de los heurísticos en otras dos plataformas WAN.

Tabla 5.9: Parámetros de los Algoritmos.

Problema	|Pob.|	ρ_c	ρ_m	Otros
MaxCut, VRP (GA)	100	0.8	0.01	-
MTTP, ECC (GA)	200	0.8	0.02	-
RAS, FMS (ES)	100	0.3	0.80	-
MaxCut (GASA)	100	0.8	0.01	El operador SA se aplica con prob 0.1, MarkovChainLen = 10, 100 iteraciones
RAS (CHCES)	100	0.8	-	35 % reinicio de la población, operador (1+10)-ES (prob 0.01)

Los parámetros de los algoritmos usados en los experimentos se presentan en la Tabla 5.9. La Tabla 5.10 muestra la configuración software y hardware de los equipos usados. Todos las pruebas realizan 30 ejecuciones independientes. De los resultados obtenidos se mostrarán la media de las mejores soluciones encontradas en cada ejecución independiente (**avg opt**), el número medio de evaluaciones para encontrar la solución óptima (**#evals**), el tiempo medio (**time**, en segundos), el porcentaje de éxito (**hits**), es decir, el número relativo de ejecuciones independientes que encuentra el óptimo, y el valor de confianza estadístico t-test para las evaluaciones y los tiempos (**p-value**).

Tabla 5.10: Configuración software y hardware.

	MA-cluster	**LL-cluster**	**BA-cluster**
Procesador	PIII, 500 MHz	AMD Duron, 800 Mhz	AMD K6-2, 450 Mhz
Memoria Principal	128 MB	256 MB	256 MB
Disco Duro	4 GB	18 GB	4 GB
Kernel de Linux	2.4.19-4	2.2.19	2.4.19
Version de g++	3.2	2.95.4	
Version de MPICH	1.2.2.3		

Hemos organizado esta sección en dos subsecciones. La primera analiza los cambios se producen en los mecanismos de búsqueda cuando pasamos de una red LAN a una entorno canónico WAN, mientras que la segunda parte estudia el comportamientos de los heurísticos en diferentes plataformas WAN con diferente número de procesadores o diferente conectividades.

Resultados: Caso Canónica

En esta sección realizamos un análisis comparativo del comportamiento de las versiones canónicas *LAN* y *WAN* de cada algoritmo. Para los experimentos en *WAN*, usamos una máquina de cada uno de los clusters de MALLBA (tres en total) como muestra la Figura 5.7. Para las pruebas *LAN*, las tres máquinas pertenecen al cluster de Málaga (MA-cluster).

En la Tabla 5.11 incluimos los resultados de aplicar el GA paralelo en los cuatro problemas combinatorios de nuestro banco de pruebas. Si comparamos los resultados *LAN* y *WAN* del algoritmo paralelo, en los tres primeros problemas se puede detectar una clara tendencia en la que se

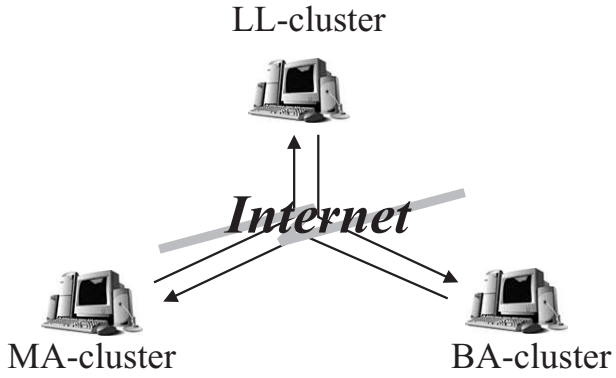


Figura 5.7: Configuración *WAN*.

observa que las ejecuciones *WAN* tardan mucho más tiempo en encontrar la solución óptima que las respectivas ejecuciones *LAN*.

Tabla 5.11: Resultados medios del GA paralelo.

	LAN				*WAN*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
MaxCut	1031	23580	49.1	17 %	1014	14369	89.1	10 %	0.91	8.13e-9
MTTP	201	40002	5.2	97 %	200	32546	25.7	100 %	0.09	3.7e-10
ECC	0.0642	18279	9.1	7 %	0.0657	26041	238.4	10 %	0.43	1.1e-14
VRP 1	696.15	12843	78.9	100 %	690.693	7168	68.5	100 %	0.92	4.78e-4
VRP 2	1080.67	5873	391.1	100 %	1077.83	8104	358.7	100 %	0.73	4.99e-6

Un detalle interesante es que el GA paralelo parece incrementar el número de aciertos cuando ejecuta en *WAN* para el MTTP y el ECC, aunque, como esperábamos, el tiempo real de ejecución para esos problemas y para el MaxCut es mayor en *WAN* debido a la sobrecarga de las comunicaciones. Numéricamente hablando, esto significa que el entorno *WAN* retarda las migraciones, y este efecto es perceptible en algunos de los problemas. De hecho, la configuración *WAN* reduce el número de evaluaciones en tres de las cinco instancias, pero los t-tests (con un nivel de significancia del 0.05) revela que esas diferencias en término de esfuerzo numérico no son significativas. Por lo tanto, la conclusión es que ambas versiones, *LAN* y *WAN*, son similares en esfuerzo computacional, para todos los problemas en el banco de pruebas. Puesto que un gran nivel de aislamiento (escaso intercambio de información entre los subalgoritmos, es decir, poco acoplamiento) no es siempre adecuado para un problema arbitrario, es lógico pensar que en algunos casos los resultados ofrecidos por el sistema *WAN* sean peores que los que obtenidos en *LAN*, como ocurre en el caso del MaxCut, cuyo porcentaje de éxito cae de un 17 % en *LAN* a un 10 % en *WAN*. Claramente, el gran desacoplamiento inducido por la *WAN* en el GA asíncrono es beneficioso para algunos problemas pero perjudicial para otros.

Sin embargo, en el caso del VRP, aunque ambas las versiones *LAN* y *WAN* obtienen similares resultados numéricos, los tiempos de ejecución con la configuración *WAN* son más pequeños para las dos instancias de este problema. Hemos realizado tests estadísticos para esos resultados, que muestran que realmente la ejecución *WAN* es más rápida que la *LAN* en este problema. Esto es un resultado muy prometedor; la explicación es que el mayor aislamiento producido por la *WAN*, es

beneficioso para un problema como el VRP, cuyo coste computacional es considerablemente mayor al resto de problemas que hemos usamos. Estos dos aspectos (mayor coste y mayor aislamiento) producen un cambio en el patrón de búsqueda seguido por el algoritmo genético paralelo cuando se ejecuta en *WAN* respecto a su misma ejecución en *LAN*, con una importante mejora en tiempo para esta primera plataforma.

Tabla 5.12: Resultados para el ES paralelo.

	LAN				*WAN*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
RAS	0	10763	5.7	97 %	0	11546	67.5	93 %	0.171	2.20e-16
FMS	0.099	10904	4.7	100 %	0.093	11372	13.0	100 %	0.450	3.26e-5

Ahora procederemos con la segunda fase en la cual analizaremos el comportamiento de un anillo de estrategias evolutivas ejecutándose en paralelo para resolver los problemas RAS y FMS. Los resultados se muestran en la Tabla 5.12. Se puede observar que las conclusiones que extrajimos para el caso del GA paralelo se mantienen en este caso donde el algoritmo base es la ES. Esta concordancia puede notarse en que los tiempos de ejecución de la versión *WAN* son más altos que los de la versión *LAN*. Otra vez, como ocurría con el GA paralelo, el fitness final (columna **avg opt**) es aproximadamente igual en ambas versiones. Además también se observa que el esfuerzo numérico es casi el mismo en ambas plataformas, lo que es un indicador de una correcta implementación.

Tabla 5.13: Resultados para los algoritmos híbridos.

	LAN				*WAN*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
CHCES1	0	13048	3.7	100 %	0	13681	10.0	100 %	0.09	8.62e-6
CHCES2	0	8093	3.4	100 %	0	8593	7.2	100 %	0.85	3.97e-5
CHCES3	0	8182	3.4	100 %	0	9635	7.9	100 %	0.13	7.9e-11
GASA1	1038	40682	89.6	17 %	1031	28956	298.5	10 %	0.53	2.97e-5
GASA2	1031	19477	45.7	7 %	1024	17412	123.5	8 %	0.22	0.0029
GASA3	1038	21651	43.8	8 %	1021	12162	202.9	7 %	0.75	0.0344

Finalmente, en la tercera fase, queremos probar nuestras conclusiones en un nuevo escenario, con algoritmos híbridos paralelos. En la Tabla 5.13 se incluyen los resultados para el problema RAS (las tres filas superiores) y el MaxCut (las tres filas inferiores). Seleccionamos esos problemas como representantes de la clases de problemas continuos y de problemas de optimización combinatoria, respectivamente. Ese es el motivo por el que usamos el CHCES para el RAS y el híbrido GASA para el MaxCut. A partir de estos resultados, podemos comprobar que se siguen cumpliendo las conclusiones anteriores que dedujimos para los algoritmos puros (GA paralelo y ES paralelo). Parece que todos los algoritmos presentan un mayor tiempo de ejecución en el caso *WAN* y que necesitan un número de iteraciones estadísticamente similar para alcanzar el óptimo. Por lo tanto se puede asumir que el comportamiento *LAN* es similar al *WAN* desde el punto de vista numérico para la mayoría de los problemas.

Concluimos con un resumen general de todos los resultados presentados en esta sección. Hemos mostrado que sea cual sea el método de búsqueda básico (GA, ES o híbrido), las ejecuciones *WAN* obtiene unos resultados numéricos similares a los de las ejecuciones *LAN*, pero que necesitan un tiempo de computo bastante superior. Por supuesto, en el caso de un mayor número de procesadores

puede que estas conclusiones no se mantengas; por lo que esta cuestión queda abierta, y adquiere una gran importancia en el caso del *grid computing*, pero su importancia decae cuando se usa número bajo o moderado de máquinas (que es lo que muchos investigadores usan en la práctica).

Nos encontramos con un resultado sorprendente cuando analizamos el problema VRP, ya que la ejecución *WAN* reduce el tiempo de búsqueda con respecto a la ejecución *LAN*. Este escenario es explicable debido al considerable tiempo de cómputo necesario por la función de fitness (que hace más similares las búsquedas *LAN* y *WAN*). También es debido a que el mayor desacoplamiento que induce los sistemas WAN (que se produce al utilizar el mismo valor para la frecuencia de migración en *LAN* que en *WAN*), representa una parametrización más optima para este problema. Esto se debe a que el heurístico es asíncrono, ya que la frecuencia de migración en heurísticos síncronos no admite mejoras numéricas en la WAN; con algoritmos síncronos, este parámetro sólo afectaría al tiempo de ejecución.

Resultados: Caso Extendido

En esta sección continuamos con nuestro análisis extendiendo los resultados de la sección previa con dos nuevos escenarios. Nuestra plataforma canónica *WAN* previa sólo incluía tres procesadores separados por Internet, lo que nos permitía establecer una base para las comparaciones. Ahora nos podríamos preguntar la influencia que tiene sobre las anteriores conclusiones, el número de procesadores que utilicemos y la conectividad existente entre ellos. En esta sección analizamos el comportamiento de los heurísticos en dos nuevos escenarios: *WAN1* (Figura 5.8a), y *WAN2* (Figura 5.8b). En *WAN1* maximizamos el número de enlaces que pasan a través de Internet de nuestro anillo lógico de máquinas (vecinos contiguos están en lugares físicamente separados), y por lo tanto, todos los enlaces están en la WAN. En *WAN2* dividimos el anillo en dos partes, cada una de ellas ejecutada en una LAN separada, con lo que sólo hay dos enlaces a través de la WAN. *WAN1* y *WAN2* usan ocho máquinas, cuatro del clúster de Barcelona (BA-clúster) y otras cuatro de La Laguna (LL-clúster).

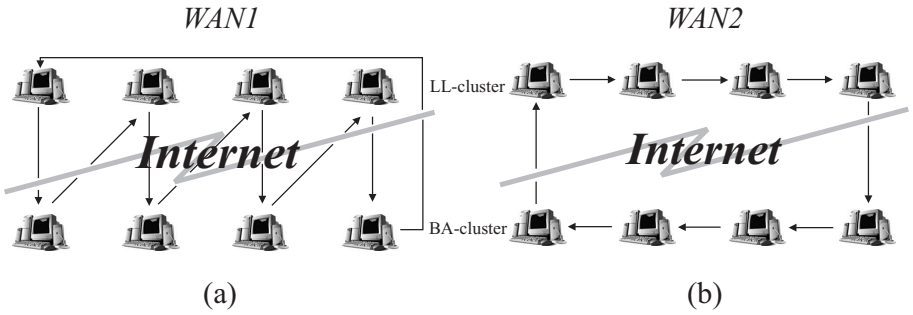


Figura 5.8: Configuraciones WAN usadas: (a) configuración *WAN1* y (b) configuración *WAN2*.

En esta sección seguiremos la misma organización que utilizamos en la sección 5.2.3. Primero, analizaremos el comportamiento del GA paralelo; después, estudiaremos los resultados del ES paralelo, y por último, examinaremos las versiones híbridas.

La Tabla 5.14 muestra los resultados cuando usamos el GA paralelo sobre estas nuevas plataformas. Una primera conclusión que se puede deducir de esos resultados es que la ejecución en *WAN2* es en la mayoría de los casos mucho más rápida que en *WAN1*. Al minimizar *WAN2* el número de mensajes por los enlaces de comunicación más lentos (es decir, a través de Internet), consigue reducir la sobrecarga de las comunicaciones, produciendo un tiempo de ejecución menor.

Tabla 5.14: Resultados para el GA paralelo.

	WAN1				*WAN2*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
MaxCut	1041	14246	37.1	17 %	1040	12283	21.0	17 %	0.14	1.52e-4
MTTP	200	32886	21.7	100 %	201	35326	7.1	97 %	0.47	4.31e-7
ECC	0.0653	26603	423.1	10 %	0.0659	23243	117.2	17 %	0.22	2.7e-10
VRP 1	662	13160	55.5	100 %	657	17066	32.1	100 %	0.16	0.246
VRP 2	1017	7166	186.3	100 %	1011.1	10604	144.9	100 %	0.17	0.0243

Las dos plataformas obtienen un número similar de aciertos, y sólo en el ECC se tiene un incremento significativo en el índice de éxito cuando se ejecuta en *WAN2*. Por lo tanto, la tabla muestra que el esfuerzo numérico es bastante similar en ambos casos.

Si comparamos estos resultados contra los obtenidos para las configuraciones iniciales (*LAN* y *WAN*), podemos observar que *WAN1* y *WNA2* mejoran a las versiones previas numéricamente (Tabla 5.11) para el MaxCut y el VRP, ya que las nuevas configuraciones obtienen un mejor fitness medio final que en el caso de la versión *WAN*. Si comparamos los tiempos de ejecución, vemos que también los mejoran (excepto para la instancia de ECC). Esta mejora es debido al incremento de capacidad de cálculo (más número de procesadores); pero analizar y comparar resultados cuando se trabaja con hardware y software heterogéneo es difícil, pero debemos tratar con esta heterogeneidad que es una característica intrínseca de Internet.

Tabla 5.15: Resultados para el ES paralelo.

	WAN1				*WAN2*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
RAS	0	9213	38.3	100 %	0	9407	3.1	100 %	0.887	2.2e-16
FMS	0.089	13772	11.6	100 %	0.09	10727	7.2	100 %	0.089	1.5e-4

Ahora, en la segunda fase, analizaremos los resultados para el ES paralelo (Tabla 5.15). Los experimentos con el algoritmo ES paralelo confirman las conclusiones obtenidas en el caso del GA paralelo de esta misma sección. Es decir, *WAN2* es claramente más rápida que *WAN1*, y ambos obtienen un número similar de aciertos con un número similar de evaluaciones. El análisis estadístico muestra que en general, *WAN2* es más rápido que *WAN1*, mientras que las diferencias en esfuerzo numérico muestran una tendencia a presentar a *WAN2* como numéricamente mejor (aunque sólo es una tendencia, ya ninguno de los t-tests son positivos para la columna **eval**). Como esperábamos, estos resultados reducen el tiempo de ejecución con respecto a los tiempos de la *WAN* que se presentaron en la anterior sección para este algoritmo, ya que hemos incrementado el número de procesadores. Desde un punto de vista numérico, estas ejecuciones no presentan diferencias significativas con las previas.

Por último, en la última fase estudiamos el caso de los algoritmos híbridos (Tabla 5.16). Estos resultados confirman de nuevo las conclusiones que se han obtenido en todos los experimentos anteriores; es decir, *WAN2* es más rápida que *WAN1*, y la diferencia en cantidad de esfuerzo numérico no es significativa en ningún caso (los t-tests siempre son negativos).

Los resultados de esta tabla vuelven a mejorar a los de la sección anterior (Tabla 5.13), es decir, se vuelve a producir una reducción en el tiempo de cálculo y además, para la instancia del MaxCut (las tres columnas inferiores), las nuevas plataformas consiguen mejorar tanto el índice de aciertos

Tabla 5.16: Resultados de los algoritmos híbridos.

	WAN1				*WAN2*				p-value	
	avg opt	#evals	time	hits	avg opt	#evals	time	hits	eval	time
CHCES1	0	14587	6.2	100 %	0	12091	1.9	100 %	0.23	0.0039
CHCES2	0	10126	4.8	100 %	0	8520	2.9	100 %	0.12	2.2e-11
CHCES3	0	8826	4.1	100 %	0	9013	1.8	100 %	0.83	2.5e-12
GASA1	1046.5	37873	92.9	17 %	1046.6	39753	57.4	17 %	0.15	0.036
GASA2	1046.8	24066	68.6	29 %	1052.4	25446	24.4	24 %	0.32	3.1e-09
GASA3	1045.7	25213	76.2	24 %	1049	23820	36.4	24 %	0.19	6.4e-05

como el fitness medio final de las versiones canónicas *LAN* y *WAN*.

Finalizaremos la sección resumiendo los resultados y las conclusiones obtenidas. En esta sección hemos probado el comportamiento de los heurísticos sobre dos nuevos escenarios WAN que usan ocho procesadores. La Figura 5.9 muestra los tiempos de ejecución para todos los problemas, algoritmos y escenarios. La topología en anillo dividida (*WAN2*) representa la configuración que más reduce la sobrecarga de las comunicaciones y, por lo tanto, es la que más reduce el tiempo de ejecución total. También se ha mostrado que *WAN1* y *WAN2* producen unos resultados numéricos muy similares a los que se obtenían con la plataforma canónica *WAN*, pero con una importante reducción en el tiempo de cálculo. La configuración *WAN2* nos permite tener la ventaja de poder usar redes geográficamente distribuidos sin producir un gran perjuicio en el tiempo de ejecución total.

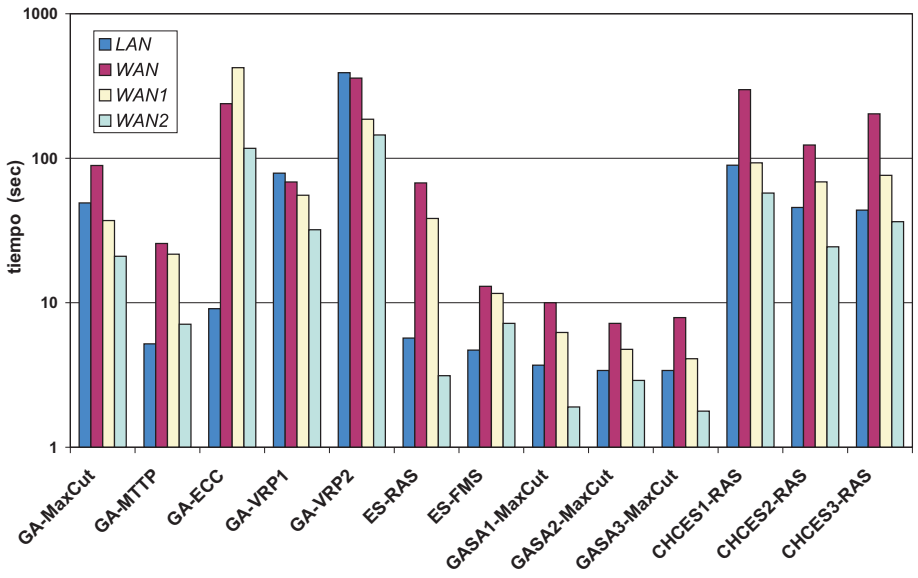


Figura 5.9: Tiempos de ejecución para todos los problemas, algoritmos y escenarios.

5.3. Análisis sobre una Plataforma Grid

Finalmente, extendemos nuestro análisis a la ejecución de las metaheurísticas en una plataforma de Grid Computing. Los sistemas de Grid Computing (o simplemente Grid) aparecen como una plataforma que proporciona el poder de cómputo de cientos de miles de computadoras, así habilitando la ejecución de algoritmos que de otra forma hubiesen sido inabordables por su alto tiempo de cómputo.

En nuestro caso queremos resolver el problema de ensamblado de fragmentos de ADN (Capítulo 8). Aunque posteriormente será definido más formalmente, este problema consiste en conseguir una ordenación que los fragmentos de ADN obtenidos en los laboratorios biológicos, que permitan reconstruir la cadena de ADN completa. Aunque en la literatura se han propuesto diferente alternativas, nosotros proponemos una nueva más precisa basada en una algoritmo evolutivo, pero que tiene el incoveniente que su función de evaluación es muy costosa en tiempo de cómputo (Ecuación 8.4). De hecho, para la instancia elegida, esta función tarda unos 15 segundos en un Pentium M 1.6 GHz usando Suse Linux 10.0. Este tiempo puede que no parezca muy alto, pero si el algoritmo se ejecutase durante medio millón de evaluaciones, el tiempo total rondaría los 86 días (suponiendo que el tiempo de cómputo de la función de evaluación es constante). Este alto tiempo de cómputo hace que sea muy desaconsejable su ejecución en plataformas secuenciales o en plataformas paralelas clásicas. Por lo tanto es un buen candidato para ser ejecutada sobre una plataforma Grid.

En concreta, examinamos el comportamiento de un algoritmo evolutivo diseñado para el Grid (GrEA, *Grid based EA*). Este algoritmo está basado en el modelo maestro-esclavo, el maestro trabaja sobre una única población que distribuye las evaluaciones de los individuos en paralelo usando los esclavos. La idea subyacente en nuestro GrEA es el diseño de un algoritmo simple pero lo suficientemente poderoso para adaptarse al entorno dinámino Grid, donde frecuentemente se añaden y se eliminan nodos del sistema. GrEA ha sido implementado sobre el sistema Condor [251], una herramienta muy conocida para este tipo de aplicaciones, usando la biblioteca MW (Master-Worker) [160]. Nuestra meta es analizar el comportamiento del algoritmo sobre este tipo de plataforma compuesta por un número pequeño/medio de unidades de cómputo (hastas 150 máquinas).

Requisitos Impuestos por los Sistemas Grid

Un sistema Grid se puede definir como una conjunto normalmente de un gran tamaño de recursos distribuidos en una red. En este contexto, se pueden distinguir dos niveles software diferentes. En el nivel superior encontramos las aplicaciones Grid, que se ejecutan sobre un sistema Grid; en nivel inferior tenemos el software del sistema Grid que se encarga de manejar la infraestructura subjacente y facilita el desarrollo de este tipo de aplicaciones.

Los recursos de los sistemas Grid comparten algunas de las siguientes características [127]:

1. Son numerosos,
2. pertencen y son administradas por diferentes personas o/y organizaciones,
3. son propensos a fallos,
4. pueden añadirse al Grid de forma dinámica,
5. tienen diferentes requisitos y políticas de seguridad,
6. son heterogéneos,

7. están conectados por diferentes arquitecturas de red,

8. tienen diferentes políticas de mantenimiento, y

9. pueden estar separadas geográficamente.

La mayoría de estos aspectos deben ser administrados por el sistema Grid, y sólo unos pocos deberían ser tenidos en cuenta por las aplicaciones. Ahora pasaremos a analizar estos elementos viendo como pueden influir en el diseño de un GA orientado para este tipo de plataforma.

El hecho de que los recursos sean numerosos (1) es el leitmotiv de los sistemas Grid, y es la principal razón por la que se pueden descartar los modelos paralelos distribuidos y celulares a la hora del diseño de nuestro acercamiento. Por otro lado, que los recursos sean propensos a fallos (3) y que se puedan agregar de forma dinámica (4), hacen que las topologías habituales (anillo, hipercubos, mallas, etc.) sean inadecuadas y difíciles de implementar). Estas son las principales razones que nos llevaron a considerar un modelo panmíctico en nuestro diseño algorítmico.

Nuestro GA panmíctico basado en el modelo de maestro/esclavo ofrece varias ventajas. Primero, el modelo es conceptualmente simple; el maestro envía iterativamente la evaluación de los individuos a los esclavos, los cuales realizan el cómputo devolviendo el valor de fitness; segundo, requiere el uso de una topología en estrella, que es muy fácil de implementar en un Grid; y finalmente, debido a su naturaleza no determinista, los principios de funcionamiento del GA no se ven afectados por la pérdida potencial de un esclavo

Una implementación de este esquema centralizado basado en el modelo panmíctico debe tener en cuenta los siguientes problems:

- Fallos en la computadora que ejecuta el maestro (3).
- Fallos en alguna de las computadoras que ejecutan los esclavos (3).
- El algoritmo debería ser capaz de utilizar lo antes posible los recursos añadidos al sistema de forma dinámica (4).
- Diferente tiempo de respuesta en los esclavos debido al diferente poder de cómputo de las máquinas (6) o debido a retrasos en la red (7).
- Ajustar el grano de cómputo de los esclavos para evitar que el proceso maestro sea un cuello de botella (1).

Esos problema pueden ser tratados a diferentes niveles: por el sistema Grid, por el algoritmo, y por el problema abordado. En la Tabla 5.17 se muestran las diferentes opciones. Una opción entre paréntesis indica que es una opción secundaria. Por ejemplo, la ruptura de la máquina que ejecuta el proceso maestro puede ser manejada por el sistema Grid si proporciona *checkpointing* automático; o en caso contrario el algoritmo debe manejar este tipo de fallos. Este mismo argumento se puede ofrecer para los fallos en los esclavos, aunque en este caso consideramos que el GA puede tratar este problema de diferentes maneras (mandando a volver a evaluar el individuo o simplemente, ignorando el fallo). Si existe la posibilidad de incorporar nuevos procesadores al sistema, el software del sistema Grid es el responsable de proporcionar un mecanismo para notificar este evento al algoritmo, el cual debe reaccionar en consecuencia. El hecho de que el tiempo de respuesta de los esclavos sea variable, es un aspecto que sólo puede ser tratado por el algoritmo. Finalmente el ajuste del grano de cómputo de los esclavo depende en exclusiva de la complejidad de la función de evaluación, que depende a su vez del problema abordado.

Tabla 5.17: Aspectos a considerar a la hora de diseñar un GA panmíctico para Grids y los niveles software donde deben abordarse.

	Sistema Grid	**Algoritmo Grid**	**Problema**
Fallos del maestro	Sí (No)	No (Sí)	No
Fallos del esclavo	No (Sí)	Sí (No)	No
Detección de nuevos procesadores	Sí	Sí	No
Diferentes tiempos de respuesta	No	Sí	No
Grano de cómputo del esclavo	No	No	Sí

5.3.1. El Algoritmo GrEA

Nuestro GrEA es un GA de estado estacionario que sigue el modelo paralelo maestro/esclavo. La idea básica es que el proceso maestro ejecuta el bucle principal del algoritmo y el esclavo realiza los cómputos asociados a la función de evaluación de forma asíncrona. A diferencia del GA secuencial de estado estacionario, nuestro GrEA realiza diferentes evaluaciones en paralelo; idealmente, deberían ser tantas como procesadores disponibles existieran en el Grid.

Para una mejor descripción del algoritmo, llamaremos GrEA-maestro a la parte del algoritmo correspondiente al proceso maestro, en contraposición de la parte del esclavo, que se denomará GrEA-esclavo. Como comentamos antes, el GrEA-maestro ejecuta el bucle principal como en el GA secuencial, pero cuando un individuo tenga que ser evaluado, se crea una tarea que se envía a un proceso esclavo que se encargará de su realización. Por lo tanto, el GrEA-maestro trabaja de forma reactiva; cuando un individuo ha sido evaluado, lo inserta en la población y se lleva a cabo un nuevo paso del GA secuencial, lo que produce un nuevo individuo que debe ser enviado a un esclavo para ser evaluado. La misión del GrEA-esclavo es recibir un individuo y evaluarlo; opcionalmente, el individuo puede ser mejorado por algún método de mejora, por lo que el individuo devuelto al GrEA-maestro puede ser diferente al original. En el próximo párrafo describiremos el método de mejora usado en nuestro GrEA.

Nuestro método de mejora está basado en el uso de versión simplificada de una estrategia evolutiva (ES). En concreto, usamos un $(\mu+\lambda)$-ES donde la probabilidad de mutación no evoluciona con el individuo, es decir, es fija. El comportamiento global de este mecanismo es como sigue. En cada iteración, el ES genera λ individuos nuevos usando un operador de mutación a partir de μ individuos. Entonces, se seleccionan las mejores μ soluciones entre las μ antiguas y las λ que se acaban de generar, para formar la nueva población. Este proceso se repite hasta que se satisfaga una condición de parada. El operador de mutación utilizado para la generación de nuevos individuos es la mutación por inversión. Este operador selecciona de forma aleatoria dos posiciones de la permutación e invierte el orden de los fragmentos entre ellas.

Ahora analizamos como aborda nuestro GrEA con los requisitos impuestos por los sistemas Grid. Como se discutió en la sección anterior, los fallos en el GrEA-maestro se asume que son resueltos por el sistema Grid; en el caso de los fallos de los procesadores que ejecutan los GrEA-esclavos, adoptamos la estrategia de ignorarlos, es decir, el individuo asignado a ese esclavo se pierde (esto no afecta al comportamiento del algoritmo). Cuando se detecta un nuevo procesador por parte del GrEA, se ejecuta un nuevo paso del GA y se obtiene un nuevo individuo para ser evaluado.

Asumiendo que los esclavos simplemente evalúan individuos, es obvio que los procesadores más rápidos evaluarán más individuos que los más lentos ya que el tiempo de respuesta puede variar. Esto introduce un ligero cambio en el comportamiento del GA "estándar", ya que individuos "viejos" pueden llegar a la población cuando esta ya ha evolucionado varios pasos. El efecto de estos individuos (si son descartados por el mecanismo de selección del algoritmo o producen per-

turbaciones significativas en la población) no es analizado en este trabajo, aunque es un aspecto muy interesante para ser analizado en trabajos futuros.

Otro aspecto en este contexto es que no ajustamos el grano de cómputo de los esclavos: el tiempo requerido para evaluar un individuo depende de ese individuo y el poder de cómputo de la máquina. En esta situación, si el tiempo requerido por la evaluación de un individuo es pequeña, puede producir un desequilibrio entre la computación y la comunicación.

Ahora consideramos la situación de cuando usamos el método ES. Aquí, el grano de cómputo de los esclavos puede ser ajustado de diferente forma; por ejemplo, configurando diferentes valores de λ o modificando el número de iteraciones del ES. En nuestro caso, hemos tomado la decisión de fijar el tiempo máximo que el esclavo puede ejecutar el ES. De esta forma, tanto las máquinas más lentas como las más lentas tardarán aproximadamente igual, con lo que se puede ajustar facilmente el ratio entre computación y comunicación.

5.3.2. Detalles de Implementación

En este apartado daremos una breve introducción a Condor y algunos detalles sobre la biblioteca MW que se usó para implementar nuestro GrEA.

Condor

Condor es un software para sistemas Grid diseñado para manejar conjunto de procesadores distribuidos en un campus u otra organización similar [251]. Se supone que cada máquina tiene un propietario, que puede especificar las condiciones bajo las cuales se permiten ejecutar trabajos de Condor; por defecto, un trabajo de Condor para cuando el propietario de la estación de trabajo empieza a utilizar la computadora. Así, estos trabajos utilizan los ciclos muertos del procesador. Comparado con otros software para el mismo proposito, Condor es fácil de instalar y de administrar, y los programas existentes no necesitan ser modificados o recompilados para ser ejecutados bajo Condor (sólo deben ser re-enlazados con la biblioteca Condor).

Las principales características de Condor incluyen llamadas a sistemas remotos, checkpointing y migración de procesos. Más aún, los *pools* de Condor pueden estar compuestos de máquinas heterogéneas, y se pueden combinar varios usando Globus [101] mediante Condor-G [102]. De esta forma, Condor maneja los aspectos que caracterizan a los sistemas grid que fueron comentados en la Sección 5.3.

La Biblioteca MW

El algoritmo GrEA ha sido implementado usando MW [160], esta biblioteca software permite desarrollar aplicaciones paralelas maestro/esclavo sobre Condor usando C++.

Una aplicación MW consiste principalmente en crear subclases de tres clases bases: MWTask, MWDriver y MWWorkser. MWTask representa la unidad de trabajo que va a ser computada en el esclavo. Esta clase incluye todas las entradas y salidas que van a ser enviadas y recibidas desde los esclavos. En nuestra implementación, la entrada es un array de enteros (la permutación que representa a los individuos) y la salida es un valor real conteniendo el valor de fitness asociado al individuo y otro array de enteros (debido a que el individuo puede ser modificado en el esclavo por un mecanismo de búsqueda local). MWWorker proporciona el contexto para la tarea en el que se va a ejecutar; en concreto, en nuestro GrEA, la subclase que heredamos de MWWorker contiene el código del GrEA-esclavo. Finalmente, la subclase de MWDriver controla el proceso completo: crea las tareas, recibe los resultados de las computaciones, y decide cuando el cómputo ha terminado. En esta clase hemos incluido el código del GrEA-maestro.

Tabla 5.18: Configuración para los experimentos.

Parámetro	**Valor**
Tamaño de la población	512 individuos
Representación	Permutación (773 enteros)
Operador de cruce	OX ($\rho_c = 0{,}8$)
Operador de mutación	Intercambio ($p_m = 0{,}2$)
Método de selección	Torneo binario
Estrategia de reemplazo	Peor individuo

El marco de programación MW trabaja con Condor para encontrar máquinas para las tareas disponibles, maneja la comunicación entre los nodos, reasigna tareas y su máquina actual falla, y controla de forma global todos los aspectos de las computaciones paralelas. MW proporciona mecanismos para guardar el estado del proceso maestro, por lo que si esta máquina falla, el proceso puede ser reiniciado.

Esta herramienta ofrece diferentes posibilidades en la capa de comunicación: PVM, sockets o ficheros compartidos. Hemos elegido esta última opción por ser la más robusta, ya que si por ejemplo, el maestro falla, los esclavos pueden continuar su cómputo, aspecto que no se puede realizar usando sockets o PVM. Aunque el proceso de comuniación usando fichero es lento, nuestra aplicación no realiza un intercambio muy intenso de datos, y el coste de la comunicación es aceptable si el tiempo de cómputo por parte de los esclavos es lo suficientemente alto.

5.3.3. Resultados Experimentales

En esta sección analizaremos el comportamiento del GrEA cuando es ejecutado en un sistema Grid. Nuestro sistema Condor está compuesto de hasta 150 computadoras que pertenecen a diferentes laboratorios del departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga. Este sistema incluye procesadores UltraSPARC 477 Mhz processors que usan Solaris 2.8, y procesadores Intel y AMD D (Pentium III, Pentium IV, Athlon) con diferentes velocidades y ejecutando diferentes distribuciones de Linux (SuSE 8.1, SuSE 9.3 y RedHat 7.1). Hemos usado Condor 6.7.12, y MW 0.9. Todas las máquinas están interconectadas a través de una red Fast Ethernet de 100 Mbps.

Los parámetros utilizados en los experimentos se han detallado en la Tabla 5.18. Debido a la naturaleza no determinista de los GAs, es necesario realizar diferentes ejecuciones independientes (por ejemplo, unas 30) de cada prueba para reunir suficientes datos estadísticos. Sin embargo, como en los sistemas Grid es difícil obtener un conjunto de computadoras estables para realizar las pruebelas durante el tiempo necesario para realizar todas esas ejecuciones independientes (varias semanas de cómputo intensivo), sólo hemos realizado dos tipos diferentes de experimentos y analizamos los resultados teniendo en cuenta los datos obtenidos de repetirlos varias veces.

Experimento 1: 500000 Evaluaciones

El primer experimento consiste en ejecutar nuestro GrEA con un límite de 500000 evaluaciones. En este caso no utilizamos el mecanismo de mejora en los esclavos, por lo que su tarea es simplemente evaluar los individuos. Hemos realizado dos pruebas con esta configuración, utilizando un sistema Grid con diferente número de máquinas. En la Tabla 5.19 resumimos los resultados de estas pruebas. Los primeros cuatro valores de esta tabla son proporcionados por la biblioteca MW. En vez de medir el speedup para medir el rendimiento paralelo (no se dispone de una versión secuencial del algoritmo debido a su complejidad), consideramos la reducción en tiempo, que

Tabla 5.19: Resultados del Experimento 1.

	Ejecución 1	Ejecución 2
Trabajdores totales	67	124
Trabajadores medios	65.5	109.7
Tiempo real	27.04 hours	17.77 hours
Tiempo de CPU total	40.28 days	45.3 days
Reducción de tiempo	35.7	61.2
Eficiencia paralela	0.56	0.56
Mejor fitness obtenido	250808	247361

consiste en dividir el tiempo total de CPU consumido por los esclavos por el tiempo real que ha tardado la aplicación, y también consideramos la eficiencia paralela, que se obtiene de dividir esta reducción de tiempo entre el número médio de trabajadores utilizados.

De la Tabla 5.19 podemos observar que la eficiencia paralela en las dos pruebas está en torno a 0.56. Este valor es relativamente bajo, lo que puede ser explicado debido al grano de computación de los esclavos, que se va volviendo más bajo conforme avanza el mecanismo de búsqueda (este detalle se explica con más detalle posteriormente). Aunque se esperaba una mejor eficiencia, la reducción de tiempo es bastante notable (pasando de unos 40 días a un único dia) y justifica el uso de nuestro GrEA en un sistema Grid.

Respecto a la calidad de las soluciones encontradas, están en el mismo rango de valores para los dos experimentos. Este resultado es consistente, ya que el algoritmo ejecuta un mismo número de evaluaciones y el modelo seguido es el mismo. Además, se ve que el número de esclavos utilizados y nuestra política de ignorar los fallos de los esclavo no afecta numéricamente al mecanismo de búsqueda.

Hemos analizado el comportamiento del algoritmo observado la traza de la primera ejecución. Concretamente, hemos medido los siguientes aspectos al principio de cada hora de ejecución:

- El tiempo de cómputo requerido por una máquina particular para evaluar un individuo.
- La evolución del mejor fitness a lo largo de la búsqueda.
- El número de individuos evaluados por hora.

En la Figura 5.10 se observan los resultados de estas mediciones. Podemos observar que la evaluación de la función del fitness requiere un tiempo de cómputo que difiere según el individuo (Figura 5.10, arriba a la izquierda), variando entre 17 y 30 segundos. Sin embargo, este tiempo desciende gradualmente a lo largo de la búsqueda, llegando a los 2 segundos en las últimas horas de computación, lo que provoca una fuerte reducción del grano de cómputo de los esclavos. La explicación de este comportamiento es debido al modo en el que trabaja la función de evaluación (véase la Sección 8.2). Un individuo en los primeros pasos del GrEA tiene un gran número de contigs, y la función de evaluación intenta reducirlos de forma iterativa mediante un método voraz. A lo largo del proceso evolutivo, este número de contigs se reduce de manera significativa, haciendo que este método voraz sea mucho más rápido y el tiempo de la evaluación disminuya significativamente.

La curva de la evolución del mejor fintess (Figura 5.10, arriba a la derecha) es casi lineal. Esto sugiere que la población todavia no ha convergido y que se puede esperar una mejora en la calidad de las soluciones si se incrementa el número de evaluaciones.

Finalmente, el número de individuos evaluados por hora (Figura 5.10, abajo) se incrementa gradualmente como consecuencia de la reducción del tiempo de cálculo de la función de evaluación. Por lo tanto, el número de evaluaciones que se realizan en las últimas horas es muy alto, lo

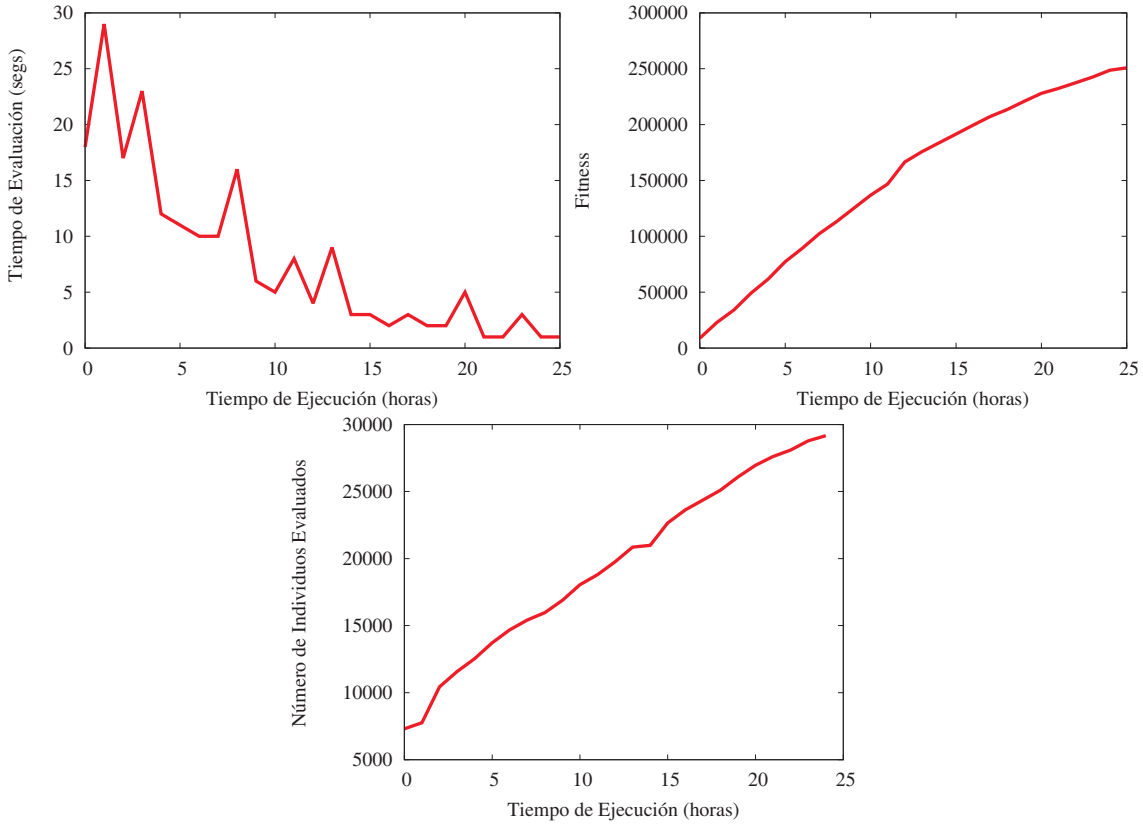


Figura 5.10: Análisis gráfico de la resultados de la ejecución 1 del Experimento 1.

Tabla 5.20: Resultados del Experimento 2.

	Ejecución 1	Ejecución 2
Trabajdores totales	149	146
Trabajadores medios	138.4	138.2
Tiempo total	40.3 hours	41.6 hours
Tiempo de CPU	187.71 days	191.4 days
Redución de tiempo	111.8	110.5
Eficiencia paralela	0.75	0.76
Mejor fitness obtenido	289077	287200

que es un efecto no deseado ya que el ratio entre computación y comunicación se ve claramente desequilibrado.

Como consecuencia de todo este análisis, se ve claramente que se debe intentar ajustar de forma más adecuada el tiempo de cómputo de los esclavos para que no se produzca esta situación. Este aspecto es tratado con nuestro segundo experimento.

Experimento 2: Uso de un Método de Mejora en los Esclavo

De los experimentos previos deducimos que es necesario incrementar el grano de computación de los esclavos cuando la búsqueda progresa. Ahora analizamos el uso de la $(\mu + \lambda)$-ES descrita en la Sección 5.3.1. La idea es establecer un tiempo límite; si tras evaluar el individuo este tiempo no se ha sobrepasado, se ejecuta el ES para mejorar la solución. El ES se ejecuta hasta que este tiempo se haya consumido. Al final de los cálculos realizados el esclavo, se devuelve al maestro el valor de fitness asociado y el individuo mejorado. Por lo tanto, las tareas enviadas al esclavo puede producir varias evaluaciones del individuo.

Una vez establecido el tiempo límite, nos aseguramos que el tiempo de cómputo de todos los esclavos es similar. Como consecuencia, los esclavos más rápidos harán una búsqueda más intensa que los más lentos. Usando esta estrategia, intentamos solucionar el problema del desequilibrio entre computación y comunicación.

Los parámetros usados en estos experimentos son idénticos a los del experimento anterior (Tabla 5.18). Los parámetros para el método ES son $\mu = 1$ y $\lambda = 10$, y el tiempo máximo de cómputo está establecido a 30 segundos. La condición de parada es alcanzar 500000 tareas realizadas. En la Tabla 5.20 se resumen los resultados de estos experimentos.

Comparando con el experimento previo, podemos observar que el tiempo total de CPU se ha incrementado de 40 a 187 días, lo que es bastante lógico, ya que el uso del ES provoca una mayor complejidad. Sin embargo, la eficiencia paralela ha crecido hasta el 0.75 usando más máquinas, lo que indica que esta nueva aproximación mejora el uso de poder de las CPU del Grid. También observamos un incremento importante en el valor de fitness, lo que indica un mayor acercamiento a la solución óptima.

En la Figura 5.11 (arriba a la izquierda), mostramos la traza del tiempo de evaluación de un individuo en una máquina. En esa gráfica se puede ver que en las 10 primeras horas, el tiempo oscila entre 30 y 59 segundos. Este comportamiento es coherente con los resultados del experimento previo (véase la gráfica de arriba a la izquierda en la Figura 5.10), ya que como vimos en las etapas la función de evaluación tardaba en torno a los 30 segundos; así, si este tiempo es menor que los 30 segundos (el tiempo límite), al menos se ejecuta una iteración del ES. De ahí que el tiempo oscile entre 30 y un poco menos de 60 segundos. Pasadas las 10 primeras horas, este tiempo varia entre 30 y 34, lo que indica que nuestro algoritmo permite que el grano del esclavo este en torno a un valor prefijado.

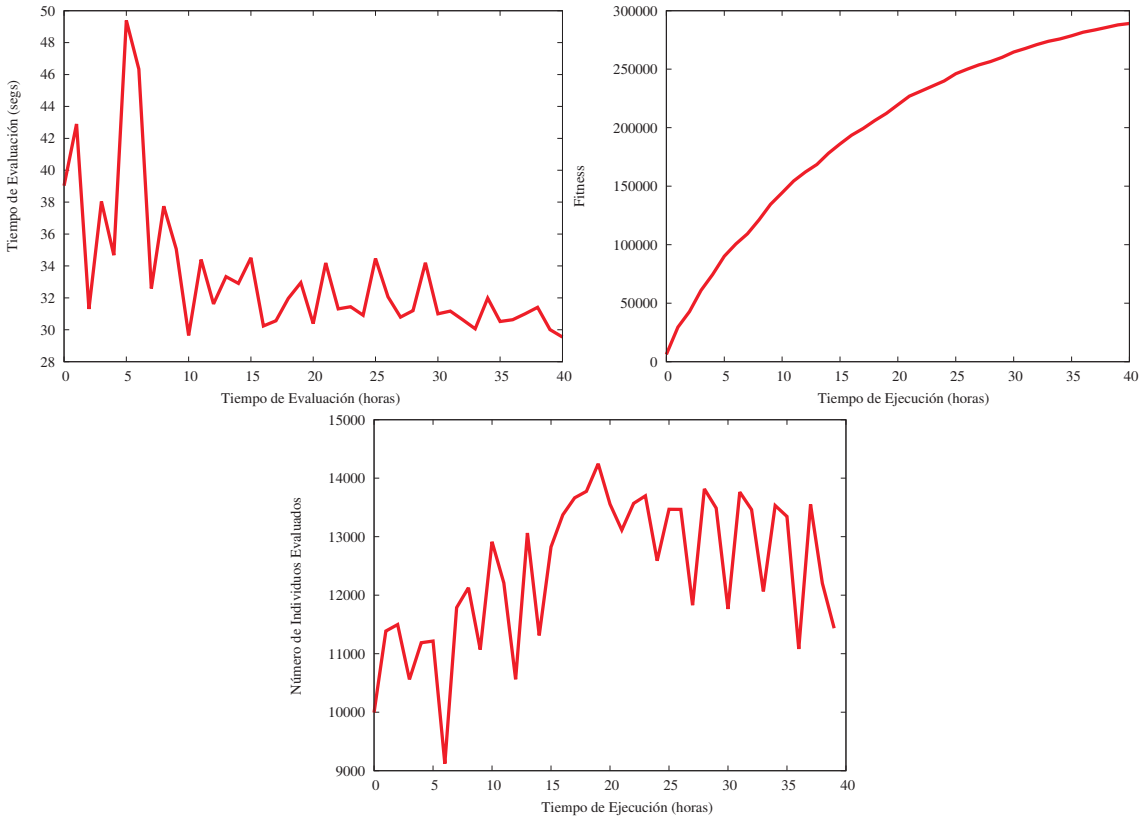


Figura 5.11: Análisis gráfico de la resultados de la ejecución 1 del Experimento 2.

La forma de la curva de la evolución del valor de fitness (Figura 5.11, arriba a la derecha) nos lleva a pensar que el algoritmo está cerca de converge a un óptimo que está cerca del 300000. Finalmente el número de individuos evaluados por hora se mantiene constante entre 9000 y 14000 (Figura 5.11, abajo). Comparada con los resultados previos (véase Figura 5.10), en el que el número de evaluaciones se incrementaba de forma lineal, hemos conseguido mantener este número en un rango en el que el ratio entre computación y comunicación es favorable.

5.3.4. Resumen

En este apartado hemos estudiado el comportamiento de una algoritmo diseñado para ser ejecutado en una plataforma Grid, GrEA. Hemos analizado los requisitos impuestos por el sistema Grid y que debe ser tenido en cuenta en el diseño de nuestro GA. El algoritmo resultante está basado en un modelo panmíctico de acuerdo a un esquema de maestro/esclavo, y su principal característica es que varios individuos son evaluados en paralelo de modo asíncrono. Esta característica simple permite cubrir con los aspectos derivados de la ejecución del algoritmo en un sistema Grid, como la existencia de muchos procesadores, fallos en las máquinas, incorporación de nuevos recursos, o diferentes tiempo de respuesta de los esclavos debido al diferente poder de cómputo de los procesadores o debido a los retrasos en la red. GrEA ha sido implementado en C++ usando la biblioteca MW sobre Condor y ha sido ejecutado sobre un sistema con un número de procesadores variable que oscila entre pocas decenas hasta algo más de 150 procesadores.

Hemos realizado dos experimentos; en el primer experimento no se hacía un ajuste del grano de cómputo del esclavo, lo que producía una pérdida de rendimiento, mostrando una eficiencia paralela de 0.56. Esto es debido a que conforme los individuos van siendo mejorados, se reduce de manera a considerable el tiempo de su evaluación y en las últimas fases del algoritmo, el ratio comunicación/computación se ve muy descompensado. En el segundo experimento incluimos en las tareas realizadas por el esclavo, un mecanismo de mejora, que está fijado para que se ejecute un tiempo máximo, con lo que nos permite fijar el grano de computación de los esclavos. Con esta estrategia se consiguió subir la eficiencia paralela a 0.79 y además usando un mayor número de procesadores.

Como futura línea de investigación se está trabajando en un estudio más profundo de nuestra aproximación, estudiando los efectos principalmente de el diferente tiempo de respuesta y fallos en los esclavos.

5.4. Conclusiones

En este capítulo hemos analizado de forma experimental el comportamiento de la algunas metaheurísticas paralelas en diferentes plataformas paralelas y con diferentes configuraciones.

Empezamos analizando el comportamiento de un algoritmo evolutivo cuando es ejecutado en diferentes redes de área local. En este estudio vimos como la influencia de la plataforma dependía principalmente del acoplamiento entre los algoritmos y por lo tanto, del número y tamaño de los mensajes intercambiados. Vimos que algoritmos con una fase de comunicación muy liviana es donde se consigue mayor eficiencia, ya que la perdida de eficiencia debida al coste de comunicación es escasa. Además, estas configuraciones al no hacer uso intensivo de la red, hace que no exista una diferencia significativa entre las redes. En configuraciones con una comunicación más intensiva, ya se ve que las redes más lentas, como es la Fast Ethernet empiezan a tener mayor problema para absorver el trafico y provoca una fuerte pérdida en el rendimiento con respecto a redes más rápidas y modernas como la Gigabit Ethernet o la Myrinet.

En nuestro segundo conjunto de experimentos, comparamos el rendimiento de diferentes algoritmos sobre una red local y una red de área extensa, para proporcionar un cuerpo de conocimiento en

este ámbito que ha sido en general poco estudiado en la literatura. Las conclusiones que se extraen de estos experimentos se puede analizar desde diferente criterios. Con respecto al comportamiento numérico, los resultados *LAN* y *WAN* parecen ser muy similares, y esa similitud es muy estable. Esto es muy interesante ya que permite reducir la discusión al otro criterio: el tiempo de ejecución. Con este objetivo en mente, hemos encontrado que las versiones *WAN* son consisténtemente más lentas que las *LAN*, como era de esperar. Sin embargo, conforme la dificultad del problem aumenta, la similitud en tiempo entre los algoritmos *LAN* y *WAN* es mayor. En el límite (es decir, en nuestras instancias más difíciles) encontramos que las ejecuciones *WAN* pueden ser incluso más eficientes que las *LAN*. Esto no es cuestión de una ejecución más rápida, ya que *WAN* siempre tiene mayores latencias y retrasos; es un efecto del desacoplamiento. Nuestra hipótesis es que ejecutar los algoritmos en WAN es algo similar a ejecutarlos en LAN pero con un mayor tiempo de aislamiento entre los subalgoritmos. Para problemas más complejos, el mayor desacoplamiento provoca alternativamente fases de exploración y explotación de manera natural, lo cual produce un algoritmo altamente eficiente. También hemos extendido el estudio completo para considerar más procesadores y diferentes conectividades. Hemos observado que la configuraciones donde se reduce el intercambio de paquetes a través de redes más lentas (como Intenet) permite reducir el tiempo de ejecución en entorno WAN y, al mismo tiempo, mantiene el mismo comportamiento numérico que los escenarios *WAN* y *WAN1*.

Finalmente, hemos extendido el trabajo para considerar en nuestro último conjunto de experimentos los sistemas Grid. En estos experimentos iniciales hemos estudiado los requisitos que debe cumplir cualquier algoritmo que quiera ejecutarse en este tipo de sistemas y hemos presentado una propuesta paralela en la que seguimos un modelo de maestro-esclavo. En los resultados hemos observado que mantener un buen equilibrio en el ratio de comunicación/computación es un punto clave para conseguir un algoritmo eficiente y que permita aprovechar las características de este tipo de sistemas.

Capítulo 6

Análisis Teórico del Comportamiento de los Algoritmos Evolutivos Distribuidos

Si en el anterior capítulo examinamos los algoritmos analizando su comportamiento experimentalmente, ahora vamos a analizar esta dinámica desde un punto de vista teórico. En concreto estudiaremos un tipo de metaheurística paralela ampliamente utilizada en la literatura, los algoritmos evolutivos distribuidos (Sección 4.3.1). A la hora de diseñar un algoritmo evolutivo distribuido hay que tomar una serie de decisiones que afectarán de forma importante al comportamiento final del algoritmo. Entre ellas, una decisión clave es determinar la política de migración: topología, ratio de migración (número de individuos que migran en cada intercambio), periodo de migración (número de pasos entre dos migraciones sucesivas), y las políticas de selección/remplazo de emigrantes/inmigrantes. Los valores de esos parámetros decidirán como explora el espacio de búsqueda el algoritmo, modificando su comportamiento según las opciones elegidas. En general, estas decisiones se toman mediante la realización de estudios experimentales. Por lo que es interesante proporcional bases analíticas para esas decisiones.

Podemos definir y estudiar muchos aspectos de los métodos paralelos (speedup, eficiencia, etc.), pero nosotros estamos interesados en analizar el modelo algorítmico distribuido subyacente, que es en realidad el responsable de la búsqueda. Por eso desde el principio de este capítulo debemos distinguir claramente entre el modelo (dEA) y su implementación (PEA). En este trabajo estamos interesados en la dinámica del EA distribuido, en particular en desarrollar una descripción matemática par el takeover time, es decir, el tiempo que tarda la población en converger por completo al mejor individuo. Primero propondremos y analizaremos varios modelos para las curvas de crecimiento, que ya es de por sí una aportación interesante, y entonces discutiremos el cálculo del takeover time.

En la literatura existen trabajos sobre la curvas de crecimiento y el takeover time para EAs estructurados [111, 112, 124, 226, 230, 231, 239]. En general, esos trabajos están orientados al estudio de EAs celulares (con la importante excepción del de Sprave [239] que tiene un impacto mucho más amplio) y en realidad existe un gran hueco en los que los estudios sobre los dEAs pueden aportar conocimiento para la investigación en este tipo de algoritmos paralelos.

En nuestro caso nos enfocamos en la influencia de los tres parámetros más importantes de la

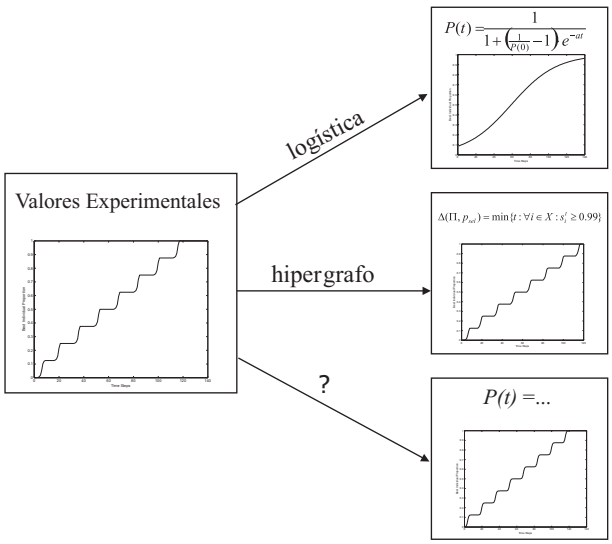


Figura 6.1: Esquema de nuestro acercamiento: estudio del modelo logístico, el de hipergrafos y posiblemente otros modelos más precisos.

política de migración (ratio, periodo y topología) sobre el takeover time y las curvas de crecimiento. Para lograr esta meta fijamos el tipo de selección/reemplazo de los individuos que migran. Los emigrantes se seleccionan por medio de torneo binario mientras que los inmigrantes se incluyen en la población destino solo si son mejores que los peores individuos existentes en la población. En nuestros análisis usamos como mecanismo de selección el muestreo universal estocástico (SUS), aunque sus conclusiones son fácilmente extrapolables a otros mecanismos de selección. Utilizamos un mecanismo de reemplazo elitista que conserva en la población los mejores individuos (en concreto usamos un $(\mu + \mu)$-dEA).

Nuestra primera contribución es poner en funcionamiento el modelo logístico [231] y de hipergrafos [239], que nunca han sido comparados en la práctica. Además, proponemos 3 nuevos modelos para simular la dinámica del modelo distribuido: un modelo de hipergrafos corregido y dos modelos originales basados en el modelo logístico (véase la Figura 6.1). Nuestro propósito es obtener un mejora en la precisión (o sea, un menor error) que los modelos ya propuestos en la literatura a la hora de calcular el takeover time.

La organización seguida en este capítulo es la siguiente. En la Sección 6.1 se dará una introducción sobre los modelos que se han aplicado previamente al cálculo del takeover time. Después en la Sección 6.2 se describirán los modelos que hemos probado en nuestro trabajo. Después continuaremos el estudio, analizando los efectos que tienen los parámetros de la política de migración sobre las curvas de crecimiento (Sección 6.3). Seguidamente analizamos cómo afectan esos mismo parámetros en el takeover time (Sección 6.4). Finalmente resumiremos las principales conclusiones que se extraen de el análisis realizado.

6.1. Modelos Previos

Una aproximación típica para estudiar la presión de selección de un EA es caracterizar su takeover time [119]. Este valor indica el número de generaciones necesario para que partiendo de una población que tiene inicialmente un único individuo óptimo, llegue a una situación donde

toda la población sean copias de ese individuo utilizando únicamente la selección. Las curvas de crecimiento son otro importante mecanismo para analizar la dinámica del modelo seguido por un algoritmo (el dEA en nuestro caso). Esas curvas de crecimiento son funciones que asignan a cada generación del algoritmo la proporción de mejores individuos existentes en la población. Ahora pasaremos a describir brevemente los principales modelos encontrados en la literatura para el análisis del comportamiento de un EA estructurado.

6.1.1. Modelo Logístico

El modelo logístico fue propuesto inicialmente por Sarma y De Jong en 1997 [231]. En ese trabajo, realizaron un detallado análisis empírico sobre los efectos del vecindario y la forma de la rejilla para un EA celular. Los autores propusieron un modelo simple basado en la familia de curvas logísticas que ya habían sido propuesto con éxito para los EAs panmícticos [119]. El modelo propuesto seguía el siguiente esquema:

$$P(t) = \frac{1}{1 + \left(\frac{1}{P(0)} - 1\right) e^{-at}} \tag{6.1}$$

donde a es el coeficientes de crecimiento y $P(t)$ es la proporción del mejor individuo en la paso generacional t. Este modelo obtuvo muy buenos resultados para cEAs con rejilla cuadrada y actualizaciones síncronas de la población. Recientemente, se han propuestos modelos mejorados que no siguen el modelo logístico para el caso de las actualizaciones asíncronas [111]. En cualquier caso, el uso de curvas logísticas es un interesante hito que se han utilizado para EAs no estructurados y para EAs celulares pero en cambio no habían sido validados para dEAs hasta este trabajo que hemos realizado.

6.1.2. Modelo de Hipergrafos

Como ya vimos en el Capítulo 4, Sprave propuso una descripción unificada para cualquier tipo de EA estructurado [239]. Este modelo se basaba en el uso de *hipergrafos*. Como vimos la descripción de un hipergrafo general es:

$$\begin{array}{lcl}
\Pi & = & (X, \mathcal{E}^t, Q) \\
X & = & (0, \ldots, \lambda - 1) \\
Q_i & = & (i\nu, \ldots, i\nu + \nu - 1) \\
\mathcal{E}^t & = & (E_0^t, \ldots, E_{r-1}^t) \\
E_i^t & = & \begin{cases} Q_i \cup \bigcup_{(s,i)\in G} M_{s\to i} & \textit{para } t \equiv 0 \textit{ mod } \eta \\ Q_i & \textit{en otro caso} \end{cases} \\
\text{con } M_{s\to i} \subset Q_i &&
\end{array}$$

Este hipergrafo representa un EA general multipoblacional con un periodo de migración η, donde X denota la población completa, Q_i representa la subpoblación i, $M_{s\to i}$ es el conjunto de individuos que migran desde la subpoblación s a la i, E_i^t son los padres potenciales en el paso generacional t de los individuos de la i-ésima subpoblación, y finalmente $\mathcal{E}^t$ es el número de todos los E_i^t en el paso t.

A partir de este modelo, podemos definir el takeover time sobre un hipergrafo de la siguiente forma: sea $\Pi = (X, \mathcal{E}, Q)$, $|X| = \lambda$ la estructura de la población y $p_{select} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}, (\lambda, k) \longmapsto p_{select}(\lambda, k)$ la función de éxito para el operador de selección usado. Entonces, para todo $i \in X$:

$$s_i^1 = \frac{1}{\lambda} \tag{6.2}$$

$$r_i^t = \sum_{j \in E_v} s_j^t, \quad i \in Q_v \tag{6.3}$$

$$s_i^{(t+1)} = p_{select}(r_i^t, |E_v|), \quad i \in Q_v \tag{6.4}$$

donde s_i^t es la probabilidad de que el individuo i tenga el mejor valor de fitness en el paso t, y r_i^t es el número (probabilístico) de mejores individuos entre los padres del individuo i en el paso t. El takeover time de Π bajo la selección p_{select} es el mínimo paso t^* donde todos los valores de $s_i^{t^*}$ son 1:

$$\Delta_{\mathcal{E}}(\Pi, p_{select}) := min\{t : \forall i \in X : s_i^{(t)} \geq 1 - \varepsilon\} \tag{6.5}$$

El valor $1 - \varepsilon$ es el nivel de precisión, donde normalmente ε es muy próximo a cero.

6.1.3. Otros Modelos

A parte de los modelos logísticos y el de hipergrafos presentados, en la literatura se han presentado otros modelos más específicos que pasaremos a describir en esta sección.

Gorges-Schleuter [124] realizó estudios teóricos sobre el takeover time para un ES celular. En su análisis, estudió la propagación de la información a lo largo de la ejecución del algoritmo para la población completa. Como conclusión ofrece un modelo lineal para la estructura basada en anillo (rejilla $1 \times N$) y un modelo cuadrático para una estructura de la población toroidal.

En otro trabajo, Rudolph [226] llevó acabo un estudio similar al anterior pero poblaciones estructuradas en array y anillo. Este autor consiguió derivar un límite inferior del takeover time para estructuras de población arbitrariamente conectadas, un límite inferior y superior para la topología en array y una fórmula exacta y cerrada para la topología en anillo..

Más tarde, Cantú-Paz [56] estudió el takeover time para un caso extremo de dGAs donde la migración ocurre en cada iteración. Para ello, generalizó el modelo panmíctico presentado por Goldberg y Deb en 1991 [119] mediante la adición de un termino relativo a la política de selección y reemplazo de los individuos inmigrantes y emigrantes.

Giacobini et al. [112] estudió el takeover time en EAs celulares que usa políticas de actualización de las celdas de forma asíncrona. Los autores presentan varios modelos cuantitativos para el takeover time en topología en anillo.

6.2. Modelos utilizados

En este trabajo, nos enfocamos en los dos primeros modelos descritos: el logístico y el del hipergrafos. El resto de los trabajos mencionados [124, 226, 56, 112] no son utilizados de forma directa ya que están ligados a algoritmos especializados (no canónicos) o tienen diferente enfoques (por ejemplo, política de selección). Primero presentamos el modelo logístico utilizado en este análisis:

$$P(t) = \frac{1}{1 + a \cdot e^{-b \cdot t}} \tag{6.6}$$

Para adherirnos estrictamente al trabajo original de Sarma y De Jong, el parámetro a debe ser definida como una constante ($a = \frac{1}{P(0)} - 1$). A este modelo estrictamente similar al de Sarma y De

Jong lo denominamos LOG1. También proponemos una nueva variante que denominamos LOG2. En este último caso, consideramos que a y b son variable libres (en el modelo previo LOG1 a era constante).

Ahora vamos a considerar el modelo de hipergrafos. De hecho, presentamos dos variantes de este modelo: una en la que la distribución de probabilidad p_{select} (Ecuación 6.7) sólo tiene en cuenta la probabilidad de selección (HYP1), y otro en el que esta probabilidad (Ecuación 6.8) tienen en cuenta tanto los efecto de la selección como el reemplazo (HYP2). Hemos introducido esta distinción debido en un trabajo seminal [57] se considera que esta segunda alternativa (combinando la selección y el reemplazo) es más exacta:

$$p_{select1}(i, N) = \frac{i \cdot \gamma}{i \cdot \gamma + N - i} \tag{6.7}$$

$$p_{select2}(i, N) = \frac{i}{N} + \left(1 - \frac{i}{N}\right) p_{select1}(i, N) \tag{6.8}$$

donde N es el tamaño de la población e i denota el número total de mejores individuos en la población. Estas funciones asumen que la población sólo tiene dos tipos de individuos donde el ratio del fitness es $\gamma = f_1/f_0$ [57]. Para nuestros experimentos hemos fijado $\gamma = 2$ ($\gamma = f_{mejor}/f_{avg(no\_mejor)} = 2$).

Para finalizar esta sección, introducimos nuestras nuevas propuestas. En nuestros primeros trabajos [21], presentamos una extensión del modelo logístico más precisa (*per* es el periodo del migración):

$$P(t) = \sum_{i=1}^{i=N} \frac{1/N}{1 + a \cdot e^{-b \cdot (t - per \cdot (i-1))}} \tag{6.9}$$

Pero este modelo no incluía información sobre la topología usada en el EA distribuida, de hecho, este modelo únicamente trabaja de forma apropiada con la topología en anillo. Por lo tanto, proponemos una nueva extensión del modelo anterior pero que tiene en cuenta la topología de migración [23]:

$$P(t) = \sum_{i=1}^{i=d(T)} \frac{1/N}{1 + a \cdot e^{-b \cdot (t - per \cdot (i-1))}} + \frac{N - d(T)/N}{1 + a \cdot e^{-b \cdot (t - per \cdot d(T))}} \tag{6.10}$$

donde $d(T)$ es la longitud del camino más largo entre dos islas cualesquiera (diámetro de la topología). Esta expresión es una combinación del modelo logístico más nuestro previo modelo. De hecho, el caso panmíctico ($d(T) = 0, per = 0$, y $N = 1$), esta ecuación se simplifica en el modelo logístico propuesto por Goldberg y Deb [119]; y si la topología es la de anillo ($d(T) = N - 1$) entonces este modelo se reduce a nuestro modelo previo presentado en la Ecuación 6.9. Debemos tener en cuenta que ya que este modelo es una extensión del logístico, podemos definir dos variantes como hicimos antes con LOG1 y LOG2. El primero (TOP1) definimos a como constante ($a = \frac{1}{P(0)/N} - 1$), y el segundo (TOP2) a y b son parámetros ajustables.

6.3. Efectos de la Política de Migración en las Curvas de Crecimiento

En esta sección analizamos los efectos de la política de migración sobre la curva de crecimiento de la proporción del mejor individuo en dEAs. Nuestro propósito es realizar un conjunto de experimentos con varios valores de periodo, ratio y topología de migración. Primero describimos los parámetros usados en estos experimentos y luego analizaremos los resultados.

6.3.1. Parámetros

Hemos realizado una serie de experimentos para servir como datos base para evaluar los modelos matemáticos. En estos experimentos, hemos usado varias topologías de migración (anillo, estrella y completamente conectada), periodos (1, 2, 4, 8, 16, 32 y 64 generaciones) y ratios (1, 2, 4, 8, 16, 32 y 64 individuos). Hemos elegido esos parámetros por ser los más utilizados en la literatura especializada del tema. Primero, analizamos el efecto de cada parámetro por separado (el resto de los parámetros serán mantenidos constante) y después, estudiaremos el efecto cuando todos esos parámetros actúan simultáneamente. En los experimentos usamos un $(\mu + \mu)$-dEA con 8 islas (512 individuos por islas), y un mecanismo de selección proporcional al fitness.

En todos los experimentos generamos de forma aleatoria los fitness de los individuos con valores entre 0 y 1023. Una vez generados los individuos, introducimos un único individuo con fitness óptimo (fitness = 1024) en una isla seleccionada de forma aleatoria. En los modelos basados en hipergrafos, utilizamos un nivel de precisión de $\varepsilon = 2{,}5 \cdot 10^{-4}$. Los datos para formar las curvas experimentales han sido extraídos de 100 ejecuciones independientes.

$$MSE(modelo) = \frac{1}{k}\sum_{i=1}^{k}(modelo_i - experimental_i)^2 \tag{6.11}$$

Para comparar la precisión de los modelos hemos calculado el error cuadrático medio (Ecuación 6.11) entre los datos reales y los predichos teóricamente (donde k es el número de puntos en la curva predicha). El MSE da el error para cada experimentos. También definimos una segunda métrica que resuma esos errores de los experimentos en un único valor, para permitir una comparación cuantitativo entre los modelos mucho más sencillo. Hemos estudiado diferentes valores estadísticos (media, mediana, desviación estándar, etc.) pero finalmente nos hemos decidido usar la norma 1, $\|\cdot\|_1$, descrita en la Ecuación 6.12 donde E es el número de experimentos. Este valor representa el área debajo de la curva creada por los valores MSE.

$$\|modelo\|_1 = \sum_{i=1}^{E}|MSE(modelo)| \tag{6.12}$$

6.3.2. Topología de Migración

Empezamos el análisis, con el estudio de las curvas de crecimiento, viendo el efecto que tiene sobre ellas, la topología de migración. En la Figura 6.2 contiene las gráficas correspondientes a diferentes topologías: anillo, estrella y completamente conectada (el periodo de migración de esa figura está a 16 generaciones y el ratio a 8 individuos). En esta figura observamos como la topología de migración afecta al crecimiento del número de mejores individuos en la población total a lo largo de la ejecución del algoritmo. El comportamiento observado es el siguiente: primero, todas las topologías tienen un pequeño escalón, indicando la rápida convergencia de la isla donde está el mejor individuos, después tenemos una zona plana, que se corresponde a cuando esta primera isla

ya ha convergido pero todavía no ha migrado la mejor solución a las demás islas. Después, cuando ocurre la migración, se incrementa la proporción del mejor individuo atendiendo al número de vecinos definidos en la topología.

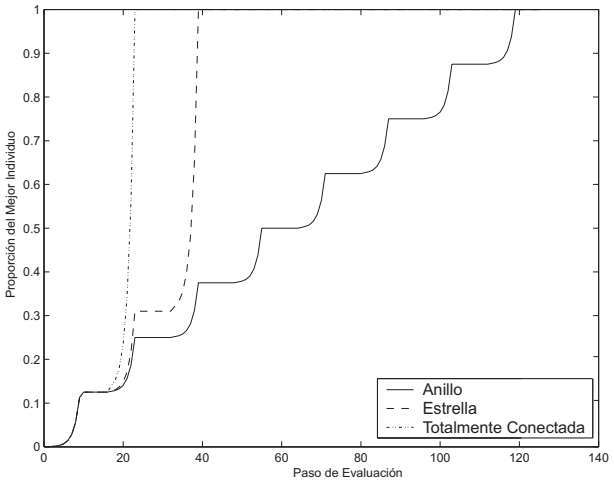


Figura 6.2: Valores experimentales para diferentes topologías (100 ejecuciones independientes).

Una vez entendido las regularidades del comportamiento del modelo, nuestro objetivo es encontrar un modelo matemático que prediga de forma precisa esas curvas.

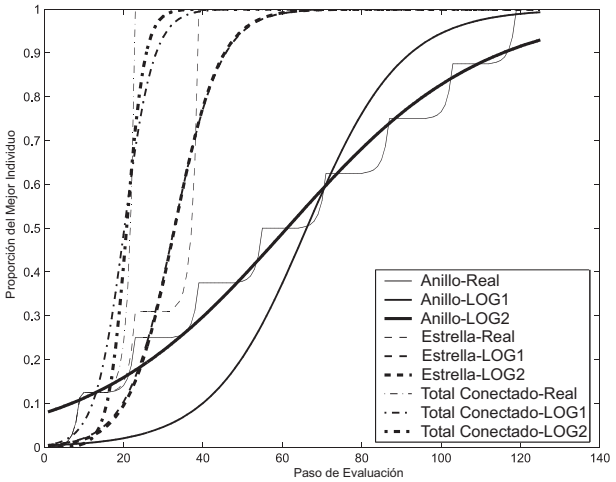


Figura 6.3: Comparación entre los valores reales y predichos con los modelos LOG1 y LOG2.

Empezamos esta tarea intentando usar el modelo logístico y de hipergrafos presentados anteriormente. Primero tratamos el caso logístico. Las curvas predichas por los dos modelos LOG1 y LOG2 están representadas en la Figura 6.3. Podemos puede observar que en el caso de la topología completamente conectada (que es la más parecida al caso panmíctico), el error de las predicciones es pequeño, pero para el resto de las topologías, estos modelos no son capaces de ajustar las curvas, produciendo resultado muy poco preciso. LOG2 parece ajustar mejor las curvas que LOG1, pero

ninguna de las dos es capaz de simular el comportamiento en forma de "escalera" de las curvas reales.

Por lo tanto, nuestra primer conclusión es que este modelos logísticos, no pueden ser utilizados para modelar los dEAs, como ya también ocurría para otros casos de EAs celulares no canónicos [124, 226, 112, 111].

Ahora analizamos el comportamiento exhibido por los modelos basados en hipergrafos (Figura 6.4). En esa figura ponemos observar una clara mejora en el ajuste con respecto a los resultados obtenidos con los modelos logísticos. Como se esperaba, HYP1 obtiene un ajuste ligeramente peor que HYP2, ya que no tiene en cuenta los efectos de la política de reemplazo. En cualquier caso, ambos modelos ofrecen unos resultados bastante precisos.

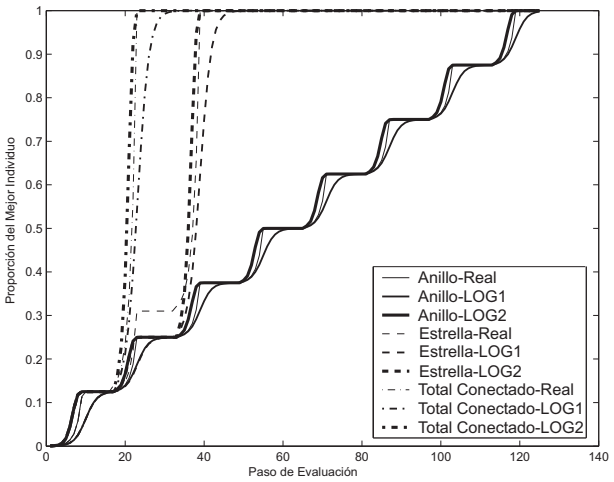


Figura 6.4: Comparación entre los valores reales y predichos con los modelos HYP1 y HYP2.

Para acabar esta sección probamos el rendimiento de nuestro rendimiento. En la Figura 6.5 representa el ajuste logrado por esos modelos. Observamos que ambos modelos, TOP1 y TOP2, son muy precisos y que tienen un comportamiento muy similar. Estos modelos son más precisos o cuanto menos igual que los otros modelos existentes, en particular con respecto a las variantes logísticas básicas y basadas en hipergrafos.

6.3.3. Periodo de Migración

En la sección anterior se fijó el periodo de migración a 16 generaciones. Ahora vamos a analizar el comportamiento cuando variamos esta parámetro con los siguientes valores: 1, 2, 4, 8, 16, 32 y 64 generaciones. La Figura 6.6 contiene las curvas experimentales obtenidos con todos esos valores para el periodo de migración (en este caso la topología ha sido fijada a anillo y el ratio de migración a 8 individuos). Esta figura muestra que para un periodo pequeño, el comportamiento del dEA es muy similar al modelo panmíctico [119]. Esto es debido que existe una alta interacción entre los subalgoritmos. Sin embargo, para periodos grandes (búsqueda desacoplada), se tiene un comportamiento diferente: la islas que tienen el mejor individuo en su población convergen rápidamente, entonces la convergencia global para (parte plana de las líneas) hasta que la migración traslada el mejor individuo a otras islas, dando a la curva un aspecto parecido a una "escalera". El periodo de migración controla el tiempo que la convergencia está parada. Cuanto más largo es el periodo, más largo es el "escalón".

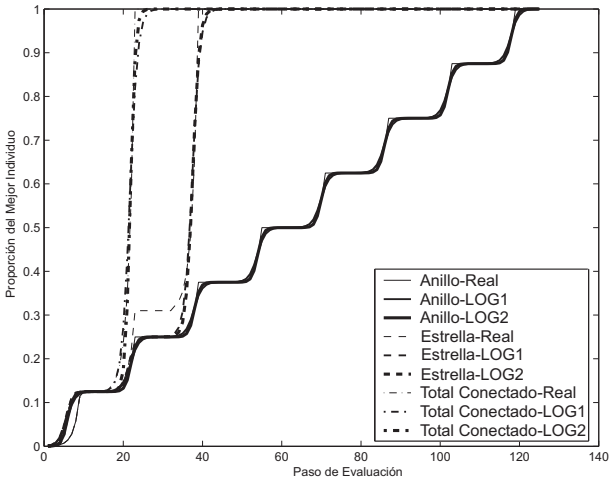


Figura 6.5: Comparación entre los valores reales y predichos con los modelos TOP1 y TOP2.

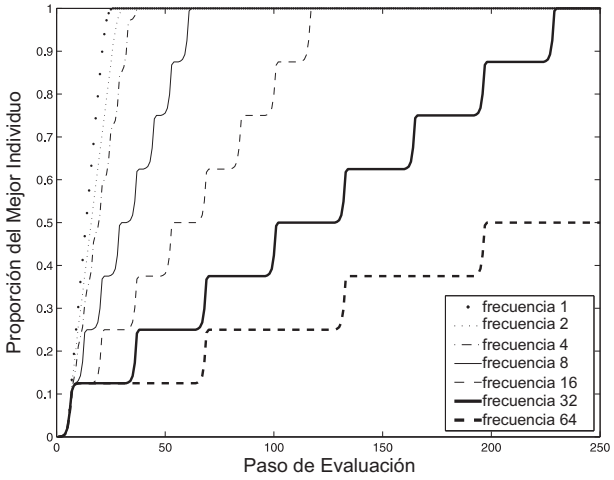


Figura 6.6: Curvas experimentales para los diferentes valores de periodo de migración (100 ejecuciones independientes).

6.3.4. Ratio de Migración

Tras analizar la topología y el periodo de migración, pasamos a estudiar la influencia del ratio de migración sobre las curvas de crecimiento. Como hemos realizado en las secciones anteriores, primero recolectamos datos experimentales usando diferentes valores para el ratio de migración (1, 2, 4, 8, 16, 32 y 64).

En la Figura 6.7 mostramos los resultados de esos experimentos. De esa figura podemos inferir que el valor del ratio de migración determina la pendiente de la curva. La razón de este comportamiento es que cuando se migran muchos individuos, la probabilidad de enviar la mejor solución se incrementa y por tanto la convergencia global es más veloz, en cambio si migramos pocos individuos, debemos esperar varias fases de migración para que finalmente se seleccione el mejor individuo

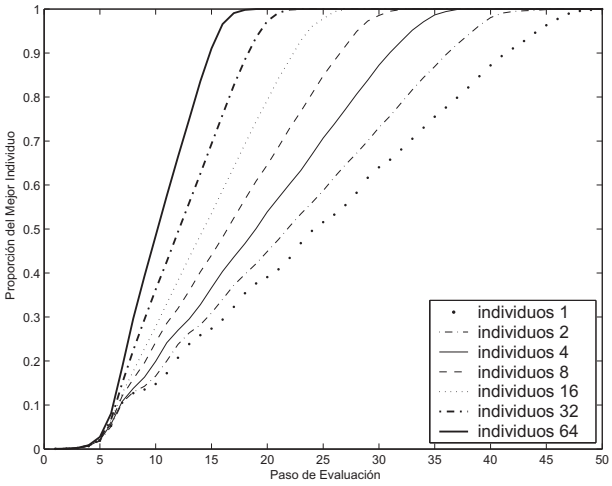


Figura 6.7: Curvas experimentales para los diferentes valores de ratio de migración (100 ejecuciones independientes).

y por tanto la convergencia es más lenta. Obviamente este comportamiento es más acusado con periodos pequeños, ya que si el periodo es lo suficientemente grande para permitir converger a la subpoblación entre dos migraciones consecutivas, el número de individuos que migremos no influye en la probabilidad de seleccionar el mejor individuo (que en cualquier caso sería 1). Por lo tanto la influencia de este parámetro en el comportamiento global está supeditada al valor del periodo utilizado de la política de selección/reemplazao que es la que afecta a la velocidad de convergencia de cada subpoblación.

Un análisis más completo de estos dos últimos parámetros comentados (periodo y ratio) lo hicimos en [21].

6.3.5. Análisis de los Resultados

En las secciones anteriores, analizamos los efectos de los diferentes parámetros por separado (fijando a un valor constante los otros). Ahora, en este sección trataremos de contestar a la pregunta: ¿están estos resultados previos sesgados por la utilización de valores constantes en el resto de los parámetros? Para responder a esta pregunta, extendemos los estudios anteriores analizando el comportamiento cuando se utilizan todos los parámetros de forma conjunta.

En la Figura 6.8 (gráfica de la izquierda) mostramos el error al intentar ajustar el comportamiento real de un dEA con topología de estrella y cualquier combinación de valores de periodo y ratio de migración, con los modelos matemáticos considerados. Para interpretar esta gráfica, se debe considerar que los siete primeros puntos corresponden al error inducido por los modelos en los siete diferentes valores de ratio de migración con un periodo concreto. Cada grupo de siete valores corresponden a un periodo concreto que oscila entre 1 y 64 (leyendo la figura de izquierda a derecha).

Del MSE obtenido por los modelos que se muestran en la Figura 6.8 se pueden extraer diferentes patrones de comportamiento claramente marcados según el modelo matemático usado. Primero, se puede observar que el comportamiento de los modelos logísticos son muy estables y bastante preciso para todas las configuraciones. Ambas variantes exhiben un comportamiento muy similar, aunque LOG2 obtiene un error ligeramente menor que LOG1. En segundo lugar, los modelos

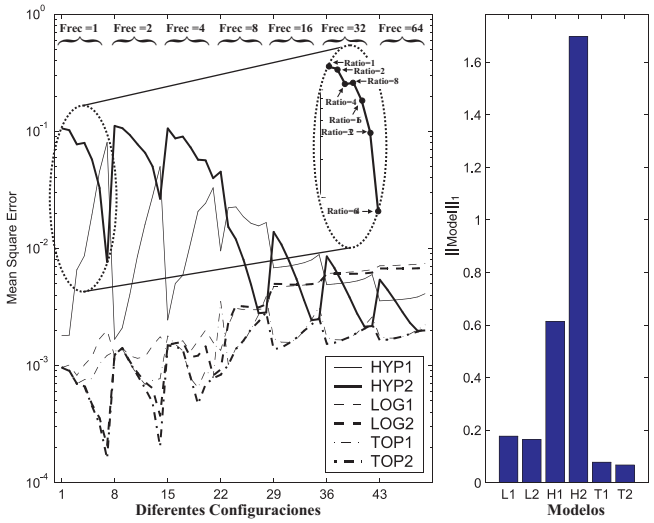


Figura 6.8: Error (MSE) y $\|\cdot\|_1$ entre los resultados experimentales y los predichos por los diferentes modelos para la topología en estrella y cualquier combinación de ratio y periodo de migración.

basados en hipergrafos obtienen un error muy pequeño para las configuraciones que tienen un periodo largo, mientras que para configuraciones muy acopladas (o sea, con un periodo pequeño) el error aumenta de forma considerablemente. También nuestra propuesta (TOPx) es bastante sensible en las configuraciones con periodo pequeño, pero en cambio es muy estable y preciso en el caso de periodos medios/altos. Ambos modelos, HYPx y TOPx, parecen realizar un ciclo de reducción/incremento (respectivamente) del error conforme el ratio de migración se incrementa (para un periodo concreto). El valor obtenido de $\|\cdot\|_1$ resume de forma cuantitativamente los resultados de MSE en un único valor por modelo, que se muestra en la gráfica de la derecha de la Figura 6.8. Aunque los modelos HYPx son muy precisos para los periodos largos, hemos visto que sus errores para pequeños periodos son muy altos, haciendo que la métrica $\|\cdot\|_1$ mucho más grande (peor) que el resto. Los resultados obtenidos por los modelos LOGx son muy precisos y de hecho sólo son peores que nuestras propuestas. Claramente, la segunda versión de nuestra propuesta es la que menor error produce al ajustar las curvas experimentales.

En las Figuras 6.9 y 6.10 hemos mostrado los resultados equivalentes a los anteriores pero con una topología completamente conectada y una topología en anillo, respectivamente. Lo primero que debe ser destacado es que en el caso de topología completamente conectada los modelos ofrecen un comportamiento muy similar al mostrado en el análisis anterior con la topología en estrella (Figura 6.8), es decir, los modelos HYPx son los menos precisos, mientras que LOGx son muy precisos y sólo son superados en precisión por nuestras propuestas. Por otro lado el comportamiento de los modelos mostrados en la Figura 6.10 (topología en anillo), si es bastante diferente que las otras dos. Para periodos pequeños, la mayoría de los modelos obtienen un gran error (con la excepción de LOG2 y TOP2). En general, las segundas variantes de los modelos son más precisas que las variantes originales. Los modelos logísticos muestra un comportamiento muy estable para cualquier configuración, pero mientras que LOG1 es muy poco preciso, LOG2 realiza un ajuste muy bueno, sólo superado por nuestro modelo TOP2. Los modelos HYPx y TOPx también aquí exhiben el ciclo de crecimiento/reducción explicado antes, aunque TOPx siempre es más preciso que HYPx. Como ocurría antes, TOP2 es el modelo más preciso en todas las topologías.

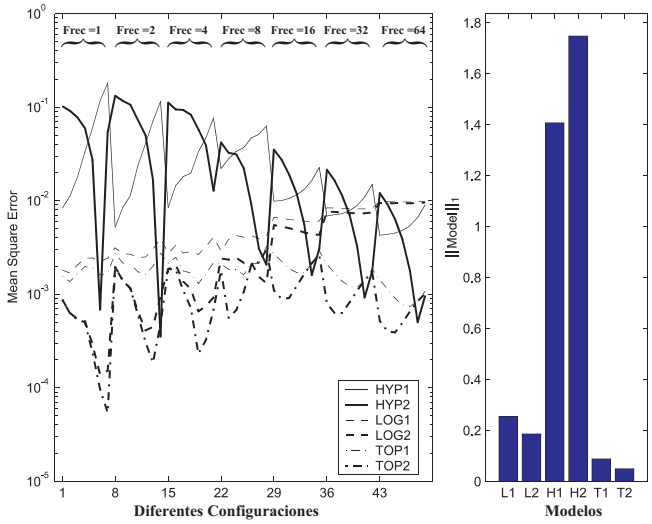


Figura 6.9: Error (MSE) y $\|\cdot\|_1$ entre los resultados experimentales y los predichos por los diferentes modelos para la topología completamente conectada y cualquier combinación de ratio y periodo de migración.

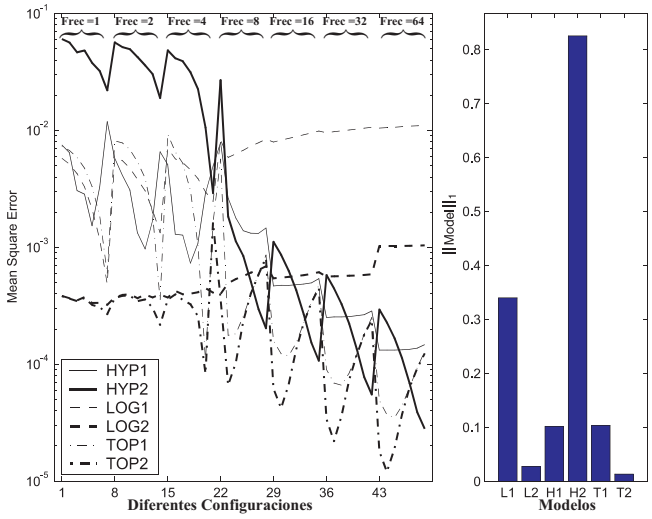


Figura 6.10: Error (MSE) y $\|\cdot\|_1$ entre los resultados experimentales y los predichos por los diferentes modelos para la topología en anillo y cualquier combinación de ratio y periodo de migración.

6.4. Análisis del Takeover Time

En las secciones previas hemos estudiado los efectos de varios parámetros de la política de migración sobre las curvas de crecimiento. Ahora, analizamos esos efectos pero sobre los valores takeover time. La Figura 6.11 contiene los valores de takeover time experimental para los diferentes valores de ratio, periodo y topología de migración. Podemos observar que el valor del takeover time se incrementa conforme aumente el tamaño del periodo. Esto era esperado, ya que es habitual que

algoritmos poco acoplados tarden más en converger que los que tienen mucha interacción. Sin embargo, el efecto del ratio es más suave que el de la frecuencia. También se puede observar como el valor de takeover time también se incrementa con las topologías con diámetro más largo, por ejemplo, la topología en anillo que es la que tiene el mayor diámetro ($d(T) = N - 1$) y su convergencia es la más lenta.

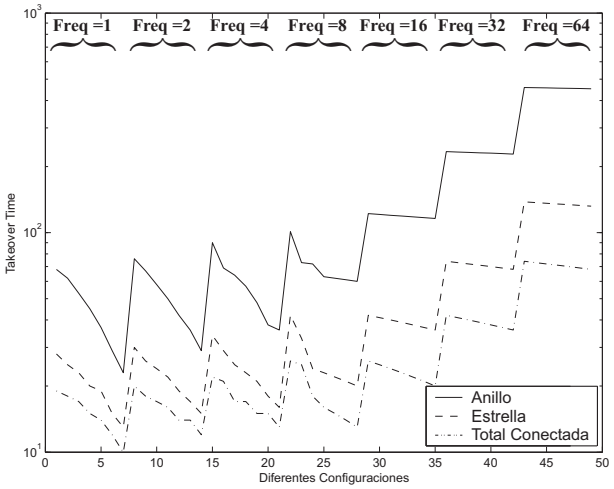


Figura 6.11: Takeover time obtenido experimentalmente para todas las configuraciones.

Una vez observado el efecto de los parámetros en el valor del takeover time, analizaremos los valores predichos por los modelos matemáticos considerados. La Figura 6.12 muestra el error medio de las predicciones. Para encontrar encontrar estos valores de takeover time, simplemente hay que iterar el modelo (sobre el parámetro t) hasta alcanzar el valor 1 (que indica que toda la población ha convergido a la mejor solución). De estos resultados, es muy interesante el caso de los modelos LOGx, que obtenían un ajuste muy bueno en el ajuste de las curvas de crecimiento, pero su error al ajustar el takeover time es muy alto. También tenemos el caso contrario, los modelos HYPx que proporcionan resultados pobres en el ajuste de loas curvas de crecimiento, ahora ofrecen unas predicciones del takeover time es muy precisas. Nuestras propuestas, TOPx, obtienen una precisión ligeramente peor que los modelos HYPx, pero en general esas diferencias son escasas. Otra conclusión que se puede extraer de esas gráficas, es que la predicción de los modelos es bastante sensible al cambio de topología; conforme aumenta la longitud de la topología, el error de los valores predichos se incrementa.

Finalmente, concluimos este estudio mostrando una ecuación cerrada para el takeover time para nuestros modelos (TOPx) y evaluaremos su precisión. La fórmula 6.13 se deriva de la ecuación presentada para el ajusto de la curva de crecimiento (Ecuación 6.10), encontrando el número de generaciones en la que converge la última isla que recibe el mejor individuo, es decir, encontrar el valor de t para el cual es segundo término de la ecuación da el valor $\frac{N-d(T)}{N} - \varepsilon$:

$$t^* = per \cdot d(T) - \frac{1}{b} \cdot Ln\left(\frac{1}{a} \cdot \frac{\varepsilon}{N - d(T) - \varepsilon \cdot N}\right) \tag{6.13}$$

donde t^* es el valor de takeover time, *per* es el periodo de migración, N es el número de islas, $d(T)$ es el diámetro de topología y ε es el nivel de precisión exigido (un valor próximo a cero).

En la Figura 6.13 mostramos el error obtenido cuando se aplica la formula de arriba para estimar el valor del takeover time. Los valores proporcionadas por esta fórmulas son bastante

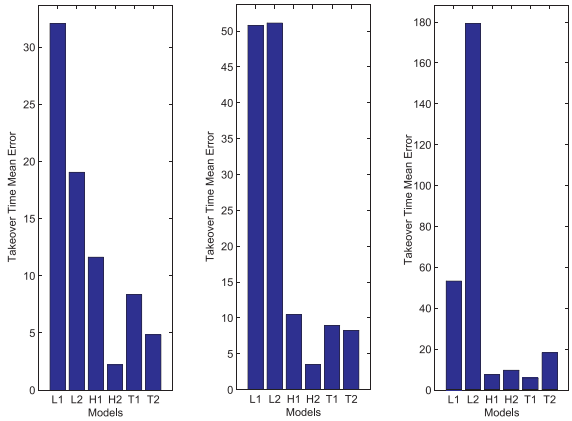


Figura 6.12: Error entre los valore predichos y los experimentales para las topologías: (a) completamente conectada, (b) estrella y (c) anillo.

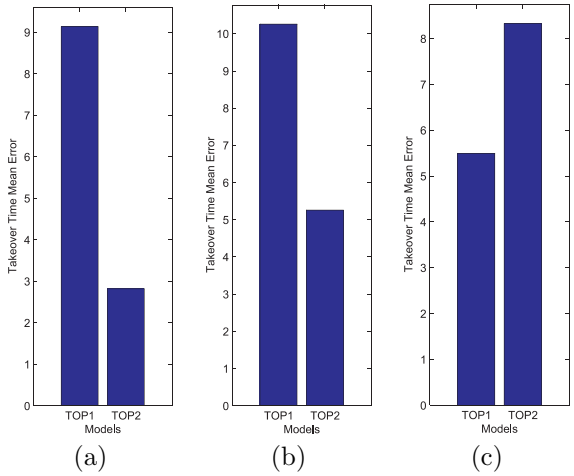


Figura 6.13: Error entre los valores predichos (con la Ecuación 6.13) y los experimentales para las topologías: (a) completamente conectada, (b) estrella y (c) anillo.

precisos y con un error bastante bajo. También hemos observados que los errores obtenidos con la Ecuación 6.13 son significativamente más pequeños que los que producían cuando los modelos TOPx eran iterados.

6.5. Conclusiones

En este capítulo hemos realizado un análisis de curvas de crecimiento y el takeover time de los algoritmos evolutivos distribuidos. Hemos comparados existentes como son el modelo logístico, otro basado en hipergrafos y también hemos propuestos un nuevo modelos. También se ha desarrollado una segunda variante de cada modelo para conseguir más precisión.

Hemos estudiado como estos modelos capturan los efectos de los parámetros más importantes

de la política de migración, es decir, el periodo de migración, el ratio de migración y la topología de migración.

Aunque cada modelo tiene su propia ventaja: ya sea su sencillez (LOGx), su extensibilidad (HYPx) o su precisión (TOP),el modelo TOP2 es que el que mejor ajuste proporciona y su estimación del valor del takeover time es de similar calidad al resto de métodos.

Como trabajo futuro, tenemos pensado aplicar el conocimiento recabada en este trabajo para diseñar nuevos algoritmos que adapten su comportamiento para producir una búsqueda más efectiva.

Parte III

Aplicaciones

Capítulo 7

Problemas de Optimización Combinatoria

Como dijimos en el primer capítulo de esta tesis, la resolución de problemas de optimización es dominio típico dentro de la Informática, ya que muchos aspectos de la vida real se puede equiparar a la resolución del problema de optimización subyacente.

Los problemas de optimización se pueden modelar por medio de un conjunto de variables de decisión con sus dominios y restricciones. Estos problemas se pueden dividir en tres categorías: (1) los que exclusivamente utilizan variables discretas (es decir, el dominio de cada variable consiste en un conjunto finito de valores discretos), (2) las que están formadas exclusivamente por variables continuas, y (3) las que usan ambos tipos de variables. Los problemas tratados principalmente en esta tesis consisten en problemas de la primera clase (denominados problemas de optimización combinatoria), debido a que inicialmente las metaheurísticas fueron definidas para este tipo de aplicaciones y muchos problemas interesantes están definidos de esta manera.

De acuerdo a Papadimitriou y Steigliz [199], un problema de optimización combinatoria $\mathcal{P} = (\mathcal{S}, f)$ es un problema de optimización que está formado por un conjunto finito de objetos $\mathcal{S}$ y por una función objetivo $f : \mathcal{S} \rightarrow R^+$ que asigna un coste positivo a cada objeto $s \in \mathcal{S}$. El objetivo es encontrar el objeto que tenga coste mínimo, es decir el hallar el $s' \in \mathcal{S} | f(s') \leq f(s), \forall s \in \mathcal{S}$ y $s \neq s'$. Se debe hacer notar que minimizar sobre un función objetivo f es lo mismo que maximizar utilizando $-f$, por lo que cualquier problema de optimización combinatorio puede ser descrito como de minimización. Aunque inicialmente en muchos dominios se utilizaban como objetos cadenas de bits, en la actualidad, los objetos típicos incluyen estructuras de datos más complejas, como permutaciones, grafos, árboles, etc.

Una vez descritos de manera formal estos problemas, pasaremos a ofrecer una breve descripción de los problema abordados en esta tesis. El resto del capítulo está dividido en dos partes. En una primera donde se describirán los problemas de carácter más académico, mostrando brevemente algunos de los resultados obtenidos para ellos. La utilización de estos problema no era hacer un estudio exhaustivo de ellos para obtener un algoritmo específico para resolverlo de forma muy eficiente, sino simplemente que sirvieran de banco de prueba para estudiar algún aspecto concreto de los modelos, o hacer comparación de diferentes algoritmos viendo como afectaba la naturaleza del problema a las características de las metaheurísticas. Ya en la segunda parte, describimos los problema de aplicación real que hemos abordado de forma más exhaustiva, pero en este caso no se mostrarán los acercamientos utilizados o los resultados ya que estos detalles se discuten en los próximos capítulos.

7.1. Problemas Académicos

7.1.1. MAXSAT

El problema de satisfatibilidad (SAT) se reconoce como unos de los problemas fundamentales de la inteligencia artificial, razonamiento automatizado, lógica matemática y campos relacionados. El MAXSAT es una variante de este problema genérico.

Formalmente, el problema SAT se define como sigue. Sea $U = \{u_1, \ldots, u_n\}$ un conjunto de n variables booleanas. Una asignación de valores de verdad de U es una función $t : U \rightarrow \{true, false\}$. Cada variable puede tener dos posibles literales u y $\neg u$. Un literal u (resp. $\neg u$) es cierto bajo t si y sólo si $t(u) = true$ (resp. $t(\neg u) = false$). Al conjunto C de literales se le denomina cláusula y representa la disjunción (conectiva lógico o). Un conjunto de cláusulas forman una fórmula. Una fórmula f (considerada como conjunción de cláusulas) es válida bajo t si todas satisface todas las cláusulas C de las que se componen, es decir si para todo C, existe al menos un literal $u \in C$ que es cierto bajo t. El problema SATonsiste en un conjunto de n variables $\{u_1, \ldots, u_n\}$ y un conjunto de m cláusulas $C_1, \ldots, C_m$. El objetivo es determinar si existe alguna asignación de valores de verdad a las variables que hagan que la fórmula $f = C_1 \wedge \cdots \wedge C_m$ sea válida. Entre todas las posibles extensiones del SAT, el problema MAXSAT [109] es el más conocido. En este caso, existe un parámetro K que indica el mínimo número de cláusulas que deben satisfacerse para solucionar el problema. El problema SAT puede considerarse con un caso especial del MAXSAT donde K el igual a m.

En nuestras pruebas utilizamos la primera instancia de De Jong [139]. Esta instancia tiene 100 variables y 430 cláusulas y el valor de fitness asociado a la solución óptima es $f^* = 430$.

En este caso elegimos el problema para examinar el comportamiento de diferentes modelos paralelos del GA. En concreto empleamos un maestro-esclavo (**MS**), un modelo distribuido sin cooperación (**idGA**), otro con cooperación (**dGA**) y un modelo celular distribuido (**cGA**).

Los resultados mostrados en la Tabla 7.1 incluyen el número de ejecuciones en las que se encontró el óptimo (**% hit**), el número de evaluaciones en las que se encontró (**# evals**) y el tiempo en segundo (**tiempo**) para todos los algoritmos paralelos ejecutados en 2, 4, 8 y 16 procesadores. En la Tabla 7.2 se muestra el speedup obtenido por cada configuración, utilizando la definición *débil* del speedup, donde se compara el algoritmo paralelo respecto a la versión canónica.

Tabla 7.1: Resultados medios para todos los algoritmos genéticos paralelos.

Alg.	% hit	# evals	tiempo	**Alg.**	% hit	# evals	tiempo
Sec.	60 %	97671	19.12				
MS2	61 %	95832	16.63	**idGA2**	41 %	92133	9.46
MS4	60 %	101821	14.17	**idGA4**	20 %	89730	5.17
MS8	58 %	99124	13.64	**idGA8**	7 %	91264	2.49
MS16	62 %	96875	12.15	**idGA16**	0 %	-	-
dGA2	72 %	86133	9.87	**cGA2**	85 %	92286	10.40
dGA4	73 %	88200	5.22	**cGA4**	83 %	94187	5.79
dGA8	72 %	85993	2.58	**cGA8**	83 %	92488	2.94
dGA16	68 %	93180	1.30	**cGA16**	84 %	91280	1.64

De estos resultados se pueden extraer diferentes conclusiones. Primero analizamos los resultados por modelo de paralelismo. Como se esperaba, el comportamiento del algoritmo maestro-esclavo es similar al del secuencial, ya que ambos tienen la misma semántica, obteniendo un número de éxitos y un número de evaluaciones similares. El tiempo de cómputo de este modelo paralelo es en general bajo, pero la eficiencia de las versiones paralelas también es muy bajo (véase la Tabla 7.2)

para cualquier número de procesadores. Esto es debido a que el tiempo de cálculo de la función de fitness es muy pequeño y no compensa la sobrecarga producida por las comunicaciones.

Tabla 7.2: Speedup para todos los algoritmos genéticos paralelos.

	Speedup			
Alg.	$n = 2$	$n = 4$	$n = 8$	$n = 16$
MSn	1.14	1.34	1.40	1.57
idGAn	2.02	3.69	7.67	-
dGAn	1.93	3.66	7.41	14.7
cGAn	1.83	3.30	6.50	11.65

El modelo **idGA** permite reducir el tiempo de búsqueda de manera importante y obtener un speedup casi lineal (véase la Tabla 7.2), pero los resultados obtenidos son peores que los del secuencial, e incluso para 16 procesadores no encuentra ninguna solución óptima del problema en ninguna de las 100 ejecuciones independientes. Este comportamiento no es sorprendente, ya que al incrementar el número de procesadores, el tamaño de las subpoblaciones disminuye, y como el algoritmo no realiza ningún tipo de cooperación, el algoritmo no es capaz de mantener la suficiente diversidad para encontrar el óptimo.

El algoritmo distribuido es mejor que el secuencial en ambos aspectos, numéricamente y en término de tiempo de ejecución, ya que permite conseguir un mayor número de éxitos con un número menor de puntos del espacio de búsqueda recorridos y también reduce el tiempo de búsqueda. El speedup es bastante bueno aunque siempre sublineal y va empeorando ligeramente cuando se incrementa el número de procesadores.

Finalmente, el modelos celular es el mejor numéricamente hablando, ya que es el que más veces alcanza al óptimo. Además, sorprendentemente, obtiene un tiempo de ejecución muy bajo, ligeramente superior al del **dGA**, que ejecuta un número de intercambio de mensajes mucho más bajo. Aunque todavía son necesarios más estudios para ver la escalabilidad de este método con un número mucho mayor de procesadores.

7.1.2. Diseño de Códigos Correctores de Errores

El problema del diseño de código corrector de errores [104] (o *error correcting code design problem*) consiste en la construcción de un código binario asignando palabras de código a un alfabeto de tal forma que se minimice la longitud de los mensajes transmitidos, a la vez que se intenta proveer de la máxima capacidad de detección y/o corrección de eventuales errores que pueden aparecer durante la transmisión.

El problema se puede definir como una tupla (n, M, d), donde n es el número de bits de cada palabra, M es el número de palabras y d es la mínima distancia de Hamming entre cualquier par de palabras. Con esta representación, el problema consiste en maximizar d dado un n y M concretos. Con lo que la función a maximizar es:

$$f_{ECC} = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} d_{ij}^{-2}} + \sum_{i=1}^{d_{min}-1} \frac{i^2}{2} \tag{7.1}$$

La resolución del problema entraña una gran dificultad en el sentido de que ambas restricciones (mínima longitud de las palabras y máxima capacidad de detección/corrección de errores) se contraponen, ya que cuanto mayor sea la capacidad de detección/corrección de errores mayor debe ser la redundancia de la información incluida en las palabras del código y, por tanto, mayor será la longitud de dichas palabras. Por tanto, se hace necesario un compromiso entre la longitud de las

palabras del código y la capacidad de detección/corrección de errores. En la Fig. 7.1 se muestra de forma esquemática el problema.

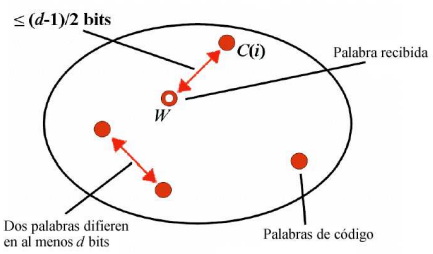


Figura 7.1: Esquema del Problema ECC.

Una instancia muy conocida y utilizada de este problema es diseñar un código con $n = 12$ y $M = 24$, que tiene un espacio de búsqueda muy amplio (aproximadamente 10^{63}).

Este problema fue abordado por un conjunto grande de algoritmos, que incluían, un algoritmo genético estacionario, un CHC, un celular, uno paralelo distribuido y un heurístico específicamente diseñado para el problema (el algoritmo de repulsión). Los detalles de los algoritmos y los resultados concretos se pueden encontrar en [13].

Las principales conclusiones que llegamos es que los algoritmos sin información adicional específica del problema no son capaces de abordar este problema de forma satisfactoria, de hecho, el índice de éxito de los algoritmo puros no superó el 6 % y incluso el método específico que diseñados no conseguía encontrar ninguno. Sin embargo cuando se procedió a hibridizar los métodos generales (el GA estacionario y el GA distribuido) con el algoritmo de repulsión, se consiguió una mejora muy notable, logrando el GA estacionario cerca de un 60 % de éxitos y el distribuido obtuvo cerca de un 90 %, indicando que el modelo paralelelo era muy adecuado para el problema, siempre que se incorporara información que permitiera limitar la búsqueda.

7.1.3. Entrenamiento de Redes Neuronales

Una red neuronal artificial (NN) está formada por un conjunto de unidades de proceso o *neuronas* interconectadas. La topología de la NN puede ser especificada como un gráfico dirigido donde los vértices son neuronas y los enlaces son las interconexiones. Cada interconexión tiene un valor real asociado que se denomina *peso sináptico*. Se pueden distinguir tres tipos de neuronas: las de entrada, las de salidas y las ocultas. Las neuronas de entrada y salida son establecidas externamente por el entorno y constituyen la entra de la NN. La salida de cualquier neurona se calcula mediante la aplicación de una función de activación a la suma de sus entradas ponderadas por el peso sináptico asociado. La salidas de las neuronas de salida es el resultado final ofrecido por la NN. Cuando se presenta un vector de entrada a la red neuronal, este se propaga a traves de las capas ocultas y ofreciendo finalmente un vector de salida en las neuronas de salida.

El par vector de entrada y vector de salida deseado se denomina patrón. Entrenar una NN consiste en ajustar los pesos sinápticos de tal forma que se aprendan esos patrones (es decir, para cada entrada produzca el vector de salida deseado). Este problema puede se especificado como un problema de optimización combinatoria. La meta es minimizar el error entre la salida actual que produce la NN y la salida deseada, computando este valor para todos los vectores de entrada disponibles. Este error puede ser calculado por medio del Error Cuadrático Medio (MSE) cuya expresión es:

$$MSE = \frac{\sum_{p=1}^{P} \sum_{i=1}^{S} (t_i^p - o_i^p)^2}{P \cdot S} \tag{7.2}$$

Otras medidas como el porcentaje de clasificaciones erróneas (CEP) son métricas equivalente que también se usan para valorar la corrección de la red neuronal.

Para la resolución de este problema, se utilizaron tanto algoritmos de búsqueda general (como un GA o CHC) y otros heurísticos específicos. Al igual que ocurrió con el problema anterior, cada técnica por separado ofrecía unos pobres resultados, pero al hibridizarlas y añadir a la técnica general conocimiento del problema a través de la inclusión de los heurísticos específicos como operadores, los resultados mejoraban de forma muy importante, consiguiendo un CEP rondando el 0 % de error para algunas instancias y un error máximo del 25 % para las redes más complejas. Los resultados completos se pueden obtenerse en [14].

7.2. Problemas de Aplicación Real

7.2.1. Ensamblado de Fragmentos de ADN

Encontrar la localización de genes que realizan funciones específicas es un aspecto muy importante dentro de la investigación bioinformática que requiere tener secuencias de ADN muy precisas. Debido a la tecnología actual, en los laboratorios es imposible leer secuencias de más de 600 bases de forma precisa. Por ejemplo, el genoma humano tiene aproximadamente unos 3.2 millones de bases de longitud y obviamente no puede ser extraído de forma completa de una lectura. En este caso, se sigue un proceso denominado *shotgun sequencing*, consistente en leer trozos o fragmentos de menor tamaño y de forma que se tenga la suficiente redundancia para asegurar la reconstrucción y que después sean ordenados automáticamente mediante alguna herramienta software. Esto nos lleva al problema que estamos tratando, el ensamblado de fragmentos de ADN [235].

Este problema es un problema de optimización combinatoria que, incluso en ausencia de ruido, es NP-Completo; si la instancia posee k fragmentos, existen $2^k k!$ posibles ordenamientos de los fragmentos. Desde este punto de vista combinatorio, el proceso entero de construir una secuencia de consenso es muy similar al de construir una ruta en el problema del viajante (TSP). De hecho la forma de resolver ambos problemas también es muy similar, pero existen importantes diferencias entre ambos. La principal diferencia es que en el problema del ensamblado de fragmentos de ADN, el orden final no es circular, a diferencia del TSP donde la primera y la última ciudad están relacionadas. Por lo que muchas soluciones equivalente para el TSP, no lo son en este contexto. Otra diferencia importante es que mientras en el TSP el orden obtenido es el resultado final del problema, en nuestro caso, sólo es un paso intermedio y puede producir que varios ordenaciones diferentes produzcan la misma cadena de ADN resultante. También existen otras pequeñas diferencias debido a ciertos aspectos específicos del problema, como puede ser la cobertura incompleta que impida reconstruir la cadena en un único trozo, o la orientación desconocida de los fragmentos, que obliga a considerar al fragmento en sí, pero también al reverso complementario, etc.

Este problema será tratado en profundidad en el Capítulo 8.

7.2.2. Etiquetado Léxico del Lenguaje Natural

Este problema es uno de los primeros pasos que hay realizar en proceso de reconocimiento del lenguaje natural. Este problema consiste en asignar a cada palabra de un texto, su función dentro del mismo. Para cada palabra existen una gran cantidad de posibles funcionalidades, como se muestra en el ejemplo de la Figura 7.2.

This	the	therapist	may	pursue	in	later	questioning	.
DT	AT	NN	NNP	VB	RP	RP	VB	.
QL			MD	VBP	NNP	RB	NN	
					RB	JJ	JJ	
					NN	JJR		
					FW			
					IN			

Figura 7.2: Ejemplo simple de asignación de etiquetas para el problema del Etiquetado Léxico.

El problema combinatorio subyacente está definido por un conjunto de variables de decisión discretas que asignan a cada palabra del texto, una de las etiquetas disponibles para esa palabra, teniendo en cuanta las etiquetas asignadas a las palabras de su alrededor mediante un modelo probabilístico.

Este problema será tratado en profundidad en el Capítulo 9.

7.2.3. Diseño de Circuitos Combinacionales

Este problema consiste en diseñar un circuito (véase Figura 7.3) que realice una función deseada (especificada por su tabla de verdad), dadas un conjunto específico de puertas lógicas disponibles.

En el diseño de circuitos se pueden utilizar varios criterios para definir expresiones para decidir el coste. Por ejemplo, desde una perspectiva matemática, podríamos minimizar el número de literales o el número de operaciones binarias o el número de símbolos en la expresión. La resolución de este problema mediante estos criterios es difícil. Si se ven el circuitos como redes de puertas lógicas podríamos minimizar el número de puertas sujeto a varias restricciones como el fan-in, fan-out, número de niveles. En general, es difícil encontrar redes mínimas o demostrar la minimalidad de una red concreta. A pesar de esto, es posible resolver varios de estos problema mediante técnicas sistemáticas, suponiendo que nos conformamos con soluciones menos generales.

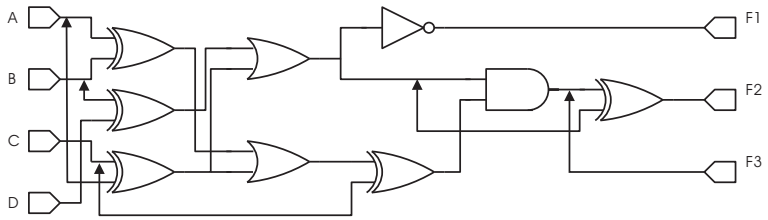


Figura 7.3: Un circuito diseñado al aplicar los algoritmos.

La complejidad de un circuito lógico es determinada por el número de puertas utilizadas en él. A su vez la complejidad de una puerta lógica está asociada al número de entradas que posee. Puesto que una puerta lógica es la realización (implementación) de una función Booleana en hardware, reduciendo el número de literales en la función, reduciríamos el número de entradas de cada puerta y el número de puertas en el circuito, y así se reduciría la complejidad del circuito.

Subyacente a este definición informal, tenemos es un problema de optimización discreto (combinatorio) en el cual las variables de decisión pueden ser o enteras o booleanas (como en este caso), codificando en cada variable de decisión el tipo de puerta que se desea colocar en una determinada

posición y las entradas que recibe. La complejidad de este problem es muy alta ya que el tamaño del espacio de búsqueda crece muy rápidamente al incrementar el número de entradas y/o salidas del circuito que se desea resolver. Además, puesto que se requiere que el circuito resultante produzca todas las salidas de la tabla de verdad correctamente, se puede considerar que este problema tiene un gran número de restricciones de igualdad muy estrictas.

Este problema será tratado en profundidad en el Capítulo 10.

7.2.4. Planificación y Asignación de Trabajadores

La toma de decisiones asociada a la planificación de trabajadores es un problema de optimización muy difícil ya que involucra muchos niveles de complejidad. El problema que nosotros tratamos consta de dos decisiones: selección y asignación. El primer paso consiste en seleccionar un pequeño conjunto de trabajadores de toda la personal disponible, que generalmente es un conjunto mucho mayor. Una vez elegidos los trabajadores, se deben asignar para que realicen las tareas programadas, teniendo en cuenta las restricciones existentes de horas de trabajo, cualificación, etc. El objetivo perseguido es minimizar el coste asociado a los recursos humanos necesarios para completar todos los requisitos de las tareas a completar.

Este problema es un problema combinatorio que utiliza variables de decisión binarias (para decidir si un trabajador es seleccionado o no) y variables enteras para indicar el número de horas que trabaja en cada tarea. Este problema se ha demostrado que es NP-Completo y que es muy difícil encontrar soluciones factibles ya que posee un número muy elevado de restricciones.

En [151] se demuestra que este problema está relacionado con diferentes problemas combinatorios de localización de recursos con capacidad como es el problema de la p-mediana con capacidades [1, 147].

Este problema será tratado en profundidad en el Capítulo 11.

7.3. Conclusiones

En este capítulo hemos dado la definición combinatoria de los problemas abordados en esta tesis. Aunque los problemas de más interés serán tratados en capítulos posteriores, en este capítulo hemos mostrado unos interesantes resultados sobre problemas de prueba que tienen un carácter más académico (aunque han sido extraídos de problemas subyacentes a aplicaciones reales).

En concreto hemos observado una comparativa entre los modelos paralelos existentes más importantes para el GA. En muchos casos los resultados eran esperados, como que el maestro-esclavo tiene un comportamiento similar al secuencial o la pérdida de diversidad de los métodos no cooperativos, pero también hemos tenido resultados bastante sorprendentes. Por ejemplo, hemos visto que el celular distribuido ofrece muy buenos resultados sin tener una gran penalización en tiempo por las comunicaciones, de hecho, su tiempo de ejecución sólo es ligeramente superior al ofrecido por distribuido normal. Este comportamiento fue exhibido en una plataforma con un número de procesadores bajo/medio y habría que hacer más estudios para comprobar su rendimiento con un número más grande de procesadores.

En los otros problemas hemos podido observar que el mecanismo genérico ofrecido por las metaheurísticas no es suficiente para abordar el problema, y al hibridizarlo con algún mecanismo especifico para el problema, el rendimiento mejora de manera considerable, permitiendo encontrar soluciones adecuadas al problema, y en muchos casos la solución óptima.

Capítulo 8

Resolución del Problema de Ensamblado de Fragmentos de ADN

Con el avance de las ciencias de la computación, la bioinformática se ha convertido en un campo muy atractivo para los investigadores del campo de la biología computacional. El uso de aproximaciones computacionales para realizar análisis del material genético también se ha convertido en algo muy popular. La principal meta de cualquier proyecto sobre material genético es determinar la secuencia completa del genoma y su contenido genético. Así, un proyecto de estas características tiene dos pasos, primero secuenciar el genoma y una segunda fase donde se anota y se sacan conclusiones de él. En este capítulo nos centramos en la primera fase, el secuenciamiento del genoma, que también es conocido como el problema de Ensamblado de fragmentos de ADN. Esta fase es uno de los primeros pasos a realizar dentro del proceso completo y el resto de los pasos dependen mucho de su precisión.

8.1. Definición del Problema

El Ensamblado de fragmentos de ADN es una técnica que reconstruye la secuencia original de ADN a partir de un colección de varios fragmentos. Los fragmentos son pequeñas secuencias de ADN, cuya longitud varía entre 200 y 1000 bases [235]. La razón por la que es necesario este proceso es debido a que la tecnología actual no permite secuenciar con precisión moléculas de ADN más grandes de 500 o 700 bases. La técnica llamada electroforesis de ácidos nucleicos en gel es la más usada en los laboratorios para la lectura de secuencias de ADN. Desafortunadamente, este mecanismo sólo puede determinar secuencias de hasta aproximadamente 700 bases en cada vez [206]. Sin embargo, la mayoría de los organismos tienen genomas mas largos que esas 500-700 bases. Por ejemplo, el ADN humano está compuesto por unos 3 billones de nucleótidos que no pueden ser leídos de una vez.

La siguiente técnica fue desarrollada para tratar con este problema: la molécula de ADN es cortada aleatoriamente para obtener fragmentos del tamaño suficientemente pequeño para que puedan ser leídos directamente. Después estos fragmentos son ensamblados para construir la secuencia de ADN original. Esta estrategia es llamada *shotgun sequencing*. Originalmente, el ensamblado de estos fragmentos más era hecho a mano. Esta aproximación manual no es sólo muy ineficiente, sino que también es muy susceptible a errores [144]. Por lo tanto, es crucial encontrar técnicas precisas y

eficientes que automaticen este proceso. En la actualidad existe un número bastante importante de paquetes que intentan resolver esta tarea. Los más populares son PHRAP [126], TIGR assembler [243], STROLL [60], CAP3 [135], Celera assembler [194] y EULER [206]. Cada uno de paquete ensambla de forma automática los fragmentos usando una gran variedad de algoritmos. Los métodos más populares son los métodos voraces y los algoritmos genéticos. El principal inconveniente de estos paquetes es que no son capaces de manejar de manera adecuada instancias de gran tamaño.

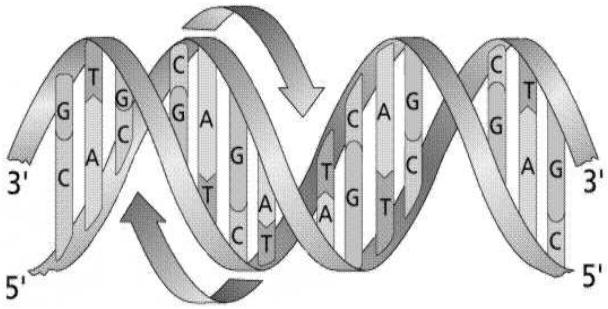


Figura 8.1: Representación en doble hélice del ADN.

La entrada de este problema es un conjunto de fragmentos que han sido cortados de forma aleatoria de una secuencia de ADN. El ADN (Ácido desoxirribonucleico) es un secuencia de nucleótidos que se combinan de forma complementario en una doble cadena formando una doble hélice (véase la Figura 8.1) . Una de las cadenas se lee de 5' a 3' y la otra de 3' a 5'. La secuencia de ADN siempre se lee en dirección 5'-3'. Las cuatro clases de nucleótidos que se encuentran en una secuencia de ADN son: Adenina (A), Timina (T), Guanina (G), y Citosina (C).

Para comprender mejor el problema presentamos la siguiente analogía:

> "Imaginemos tenemos un serie de copias del mismo libro y lo cortamos con las tijeras en miles de trozos, digamos unos 10 millones. Cada copia es cortada de tal forma que un trozo de una copia pueda solaparse con los trozos de otra copia. Digamos que en torno a un millón de trozos se pierde y que algunos del resto se manchan de tinta. El objetivo sería recomponer el texto original." [206]

Podemos identificar la secuencia de ADN destino como el texto original y los fragmentos de ADN con los trozos cortados de los libros.

El proceso empieza con la rotura de la secuencia de ADN en trozos. Esto se hace mediante la duplicación de la cadena original y el posterior corte aleatorio de cada copia. Entonces este material biológico se "convierte" a datos comprensible por la computadora. Estos pasos se realizan en los laboratorios y corresponden con los primeros pasos indicados en la Figura 8.2. Después de obtener los fragmentos se aplica el proceso tradicional de ensamblado: calcular solapamiento, calcular orden de los fragmentos y finalmente calcular la secuencia de consenso. Estos pasos son los últimos que se muestran en la Figura 8.2. A continuación describimos con un poco más de detalle estos últimos pasos.

Cálculo de Solapamiento Esta fase consiste en encontrar el mejor o más largo solapamiento común entre el sufijo de una secuencia y el prefijo de otra. Para ello, se comparan todos los pares de fragmentos para determinar su similitud. Para realizar este proceso se suele aplicar un algoritmo de programación dinámica. La idea intuitiva detrás de este cálculo, es que fragmentos que tengan un solapamiento muy grande entre ellos, tienden a estar juntos o muy cerca en la secuencia destino.

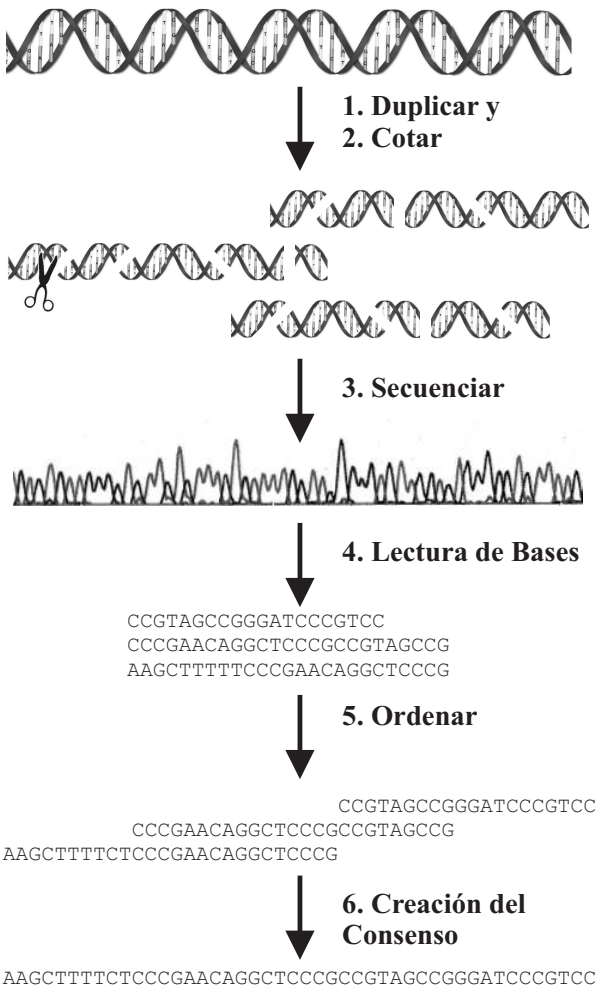


Figura 8.2: Representación gráfica del proceso de ensamblado y secuenciación de ADN [33].

Ordenación de los Fragmentos En este paso se procede a ordenar los fragmentos según el solapamiento existente entre ellos que se calculó en el paso anterior. Este paso es el más complicado, ya que es complicado decidir cual es el solapamiento real entre los fragmentos debido a los siguientes retos:

- Orientación Desconocida: Tras romper la secuencia original en trozos se pierde la orientación. Aunque un fragmento no tenga solapamiento con otro, también debemos probar si existe solapamiento utilizando el complementario reverso, ya que no conocemos la orientación.
- Errores en las Bases: Hay tres tipos de errores: sustitución, inserción y errores de borrado. Estos errores se producen durante la aplicación del proceso de electroforesis.
- Cobertura Incompleta: Esto ocurre cuando hay regiones perdidas y el algoritmo no es capaz de ensamblar todos los fragmentos en un único contig. Con el término *contig* nos referimos

al número de cadenas producidas por el algoritmo tras ensamblar los fragmentos. Idealmente este valor debería ser uno, indicando que se ha reconstruido la cadena original. Pero ese objetivo es muy complicado debido a la dificultad intrínseca del problema (es NP-Duro) y los errores que puedan haber en los datos.

- Regiones Repetidas: En la cadena de ADN puede tener diferentes regiones que se repitan en la secuencia. Estas regiones son muy difíciles de controlar por los algoritmos de ensamblado.

- Contaminación y Quimeras: Las quimeras surgen cuando dos fragmentos que no son adyacentes en la molécula destino se unen en un único fragmento. La contaminación se debe a la purificación incompleta de los fragmentos extraídos de la secuencia de ADN.

Construcción del Consenso Tras obtener el orden anterior, ahora se debe reconstruir la cadena de ADN consenso final. Para ello se sigue la regla del mayor consenso para decidir que base poner en cada posición de la secuencia final

Ejemplo Para dar una idea más completa del proceso anterior, vamos a ilustrarlo con un ejemplo simple, donde se vea con claridad el procedimiento completo.

Partimos de un conjunto de fragmentos {`F1=GTCAG, F2=TCGGA, F3=ATGTC, F4=CGGATG`}. Primero se debe calcular el solapamiento entre todos los fragmentos. Por ejemplo, entre `F1` y `F2` el solapamiento es 0, mientras que entre `F3` y `F1` el solapamiento es 3. Tras calcular estos valores, se determina el orden basado en los valores de solapamiento. Por ejemplo imaginemos que en este caso llegamos a encontrar el siguiente orden: `F2`→`F4`→`F3`→`F1`. A partir de este ordenamiento construimos la cadena de consenso como sigue:

```
    F2 ->    TCGGA
    F4 ->     CGGATG
    F3 ->        ATGTC
    F1 ->          GTCAG
------------------------
Consenso  -> TCGGATGTCAG
```

En este ejemplo, el orden resultante permite construir una secuencia que tiene un único contig. Puesto que encontrar el orden exacto consume muchos recursos tanto en tiempo como en memoria, en la literatura se recurre mucho a heurísticos para resolver este paso [201, 196, 202]. Nuestra propuesta también aplica varias metaheurísticas para resolver este paso y en las siguientes secciones se indicará como se ha abordado el problema.

Para medir la calidad de un consenso, se puede mirar la distribución de la cobertura. La cobertura en una posición se define como el número de fragmentos en esa posición. Es una medida que indica la redundancia en los datos. Indica el número de fragmentos, de media, en que se espera que aparezca cada nucleótido. Se puede calcular como el número de bases en los fragmentos entre la longitud total de la secuencia destino [235].

$$Cobertura = \frac{\sum_{i=1}^{n} longitud\ del\ fragmento\ i}{longitud\ de\ la\ secuencia\ destino} \tag{8.1}$$

donde n es el número de fragmentos. TIGR usa la cobertura para asegurarse la corrección de su ensamblado resultante. La cobertura normalmente oscila entre 6 y 10 [144]. Cuanto más alto es este valor mejor es el resultado obtenido.

8.2. Aproximación Algorítmica para su Resolución

En esta sección se comentará las aproximaciones elegidas para tratar con este problema mediante diferentes algoritmos.

8.2.1. Detalles Comunes

Primero comentamos los detalles comunes a todos los heurísticos utilizados.

Representación de la Solución

En este caso, usamos una representación basada en una permutación de enteros para codificar una solución. Esta permutación representa una secuencia de fragmentos, donde los fragmentos adyacentes se solapan entre sí. Una solución mediante esta representación requiere una lista de fragmentos asignados con un único identificador entero. Por ejemplo, con ocho fragmentos necesitaremos ocho identificadores: 0, 1, 2, 3, 4, 5, 6, 7. La permutación requiere de operadores especiales para asegurarse que producen siempre soluciones legales. Para que una solución sea legal debe satisfacer que (1) todos los fragmentos deben estar presentes en el orden y (2) no hay fragmentos duplicados dentro del orden. Por ejemplo, una posible ordenación para 4 fragmentos es $3 \rightarrow 0 \rightarrow 2 \rightarrow 1$. Con este orden nos referimos que el fragmento 3 está en primera posición, seguido del 0 en la segunda y así sucesivamente.

Función de Fitness

La función de fitness nos permite evaluar como de buena es una solución. Esta función se aplica a cada individuo en la población y permite guiar la búsqueda de la metaheurística hacia la solución óptima. En este problema existen varias posibilidades. Las más usadas se basan en el solapamiento que existen entre los fragmentos del orden. Parsons, Forrest y Burks definen dos posibles funciones de fitness [201].

La primera función (Ecuación 8.2) se basa en la suma de los valores de solapamiento entre fragmentos adyacentes en una solución dada. Cuando se usa esta función, el objetivo que se persigue es maximizar esta suma.

$$F1(l) = \sum_{i=0}^{n-2} w(f[i]f[i+1]) \tag{8.2}$$

La segunda función (Ecuación 8.3) con la que trabajan no sólo suma el valor de solapamiento entre fragmentos adyacentes sino que también suma el valor de solapamiento entre cualquier par de fragmentos. Haciendo esa suma, la función penaliza las soluciones en las que se producen un fuerte solapamiento entre fragmentos muy alejados entre sí. Entonces cuando se use esta función, el objetivo del problema debe ser minimizar este valor.

$$F2(l) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |i-j| \times w(f[i]f[j]) \tag{8.3}$$

En ambos casos, el solapamiento que existe entre cada par de fragmentos se ha calculado previamente. Para realizar este cálculo se utiliza habitualmente un mecanismo de programación dinámica.

El problema de estas fórmulas es que no capturan de forma adecuada la naturaleza del problema, que tiene como objetivo final construir la cadena original o al menos producir un número de

contigs lo más pequeño posible. Además en casos extremos puede llevar a un mejor valor de fitness (mediante las dos anteriores ecuaciones) pero un número de contigs peor. Añadir directamente el número de contigs como función de fitness no es posible, ya que su cálculo es bastante complejo y además es una medida que no da mucha información, ya que no permite distinguir entre soluciones con el mismo número de contigs (y en general existen muchas soluciones con el mismo número de contigs). Por ello en nuestros últimos trabajos en este problema, propusimos el uso de una función que tuviese en cuenta tanto el solapamiento como alguna información sobre el número de contigs. Para ello calculamos una aproximación del valor final de contigs (una cota superior, c) quedando la función final como sigue:

$$F3(l) = F1(l) \cdot \left(1 + \frac{1}{c(l)}\right) \tag{8.4}$$

Con esa fórmula se intenta de forma simultánea maximizar el solapamiento entre fragmentos adyacentes y minimizar el número de contigs. El problema de esta formulación del problema es que su cálculo requiere una gran cantidad de tiempo (incluso llegar al minuto). Este acercamiento es el usado en Capítulo 5 para evaluar el comportamiento del algoritmo en un sistema Grid, mientras que en estos trabajos (que fueron realizados con anterioridad al otro) se utilizan las dos primeras funciones de fitness. Otro posible acercamiento a esta función sería plantearla como multiobjetivo, pero no está clara que los dos objetivos sean contrapuestos, en algunos casos es claro, pero no siempre.

Condición de Parada

Aunque se sabe que la solución óptima debe tener un único contig, no se conoce el valor de fitness asociada a esa solución. Al no conocer el óptimo se considera que la condición de parada se fija como un número de evaluaciones máximo.

8.2.2. Detalles Específicos con GA

Tamaño de la Población

Para este problema utilizamos una población de tamaño fijo.

Operador de Recombinación

Este operador combina el material genético de dos o más soluciones para producir otra solución, que se espera que reúna lo mejor de los padres y sea una solución más próxima al óptimo. Este operador depende de un parámetro ρ_c que representa la probabilidad de que dos individuos sean recombinados. Un ratio de cruce $\rho_c = 1$ indica que todos los individuos seleccionados se cruzan. Sin embargo, hemos realizado estudios previos que indican que los mejores valores para estos parámetros oscilan entre 0.65 y 0.85.

Para nuestros experimentos, hemos usado el operador de cruce basado en el orden o *order-based crossover* (OX). También probamos otros operadores como el ER (*edge-recombination crossover*) pero el que mejor resultado ofreció en las pruebas iniciales fue el OX.

Este operador copia los fragmentos entre dos posiciones elegidas aleatoriamente del primer padre y las incluye en el hijo. Después el resto de fragmentos son elegidos de tal modo que sigan el orden relativo indicado por el segundo padre, teniendo en cuenta que no se pueden repetir fragmentos. Este método está expresamente diseñado para una representación basada en permutaciones y crean soluciones válidas.

Operador de Mutación

Este operador es usado para la modificación del contenido de una única solución. La razón por la que es necesario este tipo de operador es para mantener diversidad en la población. Al igual que ocurría en el otro operador, la mutación está controlada por un parámetro ρ_m que representa la probabilidad que de un individuo se vea afectado por este operador.

En los experimentos, hemos utilizado el operador de intercambio. Este operador, selecciona dos posiciones aleatoriamente e intercambia los fragmentos que están situados en esas posiciones. Puesto que este operador no duplica números en la permutación ni elimina ningún elemento, la solución generada es válida y no necesita ningún proceso de reparación.

Operadores de Selección y Reemplazo

El propósito de estos operadores es seleccionar los individuos que van a sufrir el proceso de reproducción y también decidir que soluciones formarán la siguiente población. En general estos procesos dependerán del valor de fitness que tengan las soluciones. Estos operadores definen la presión selectiva del algoritmo, que es un aspecto muy importante. Si es demasiado baja, la convergencia será muy lenta, en cambio si es demasiado alta, el algoritmo convergerá de forma prematura en un óptimo local.

En este capítulo, hemos usado un mecanismo de reemplazo por ranking [262], en el cual el GA primero ordena la población de acuerdo con el valor de fitness y la nueva población será seleccionada entre los mejores. Para la selección usamos torneo binario [119], que elige aleatoriamente dos soluciones y selecciona la mejor de las dos. Este proceso es repetido hasta que todos los padres sean seleccionados.

Versión Paralela

A la hora de paralelizar el GA, hemos seguido un modelo distribuido (Sección 4.3.1). En este modelo un se ejecutan en paralelo pequeño número de GAs independientes, que periódicamente intercambian individuos para cooperar en la búsqueda global. Puesto que entre nuestros objetivos está comparar el comportamiento de este algoritmo con respecto a la versión secuencial, se ha configurado para que la población total del algoritmo paralela sea del mismo tamaño que la del algoritmo secuencial. Es decir, si el algoritmo secuencial usa una población de K individuos, cada una de las islas en el método paralelo usa K/num_islas.

Para terminar de caracterizar nuestra propuesta de algoritmo genético distribuido para este problema, debemos indicar como se lleva a cabo la migración. Nuestra implementación usa como topología un anillo unidireccional, donde cada GA recibe información del GA que le precede y la envía a su sucesor. La migración se produce cada 20 generaciones, enviando una única solución seleccionada por torneo binario. La solución que llega sustituye a la peor solución existente en la población si es mejor que ella.

8.2.3. Detalles Específicos con CHC

Prevención del Incesto

Como se comentó en el Capítulo 2, este método tiene un mecanismo de *prevención de incesto* para evitar la recombinación de soluciones muy similares. Normalmente para medir la similaridad entre las soluciones se utiliza la distancia de Hamming, pero esta métrica no puede usarse en permutaciones. En nuestros experimentos, consideramos que la distancia entre dos soluciones es el número total de ejes menos los ejes comunes.

Operador de Recombinación

El operador de recombinación que usamos en nuestra implementación de CHC, crea un único descendiente que conserva los ejes comunes a ambos padres y que asigna de forma aleatoria el resto de los ejes para generar una permutación legal.

Reinicio de la Población

Cuando la población converge, la población es reiniciada parcialmente. Para este propósito usamos la mejor solución como plantilla para reiniciar la población, creando nuevos individuos mediante el intercambio de ejes hasta que el número de ejes diferente a la plantilla supere cierto umbral.

Versión Paralela

En el caso del CHC no hay demasiados estudios indicando los posibles modelos de paralelismo y cómo afectan al comportamiento del algoritmo. Al ser una metaheurística basada en población, se le puede aplicar los mecanismos de paralelismo generales presentados en la Sección 2.2.2. En particular, el esquema propuesto para resolver el problema del ensamblado de fragmentos de ADN, sigue el modelo distribuido similar al propuesto al GA. Aquí tenemos un conjunto de CHC secuenciales que se ejecutan de forma aislada, hasta que cada cierto número de iteraciones, pasan una solución elegida por torneo binario a su vecino siguiendo una topología en anillo unidireccional.

8.2.4. Detalles Específicos con SS

Creación de la Población Inicial

Existen diferentes métodos para obtener una población inicial que esté formada tanto por buenas soluciones como diversas. Para este problema, las soluciones de la población se generan de forma aleatoria para obtener cierto nivel de diversidad. Entonces, aplicamos el *Método de Mejora* (que será explicado en la próxima sección) para obtener soluciones de mejor calidad.

Generación de Subconjuntos y Operador de Recombinación

En esta aplicación generamos todos los subconjuntos de dos elementos. El operador de recombinación de soluciones utilizado es el OX, que ya explicamos en la sección anterior.

Método de Mejora

Para este problema utilizamos un método de escalada (*hillclimber*) para mejorar la calidad de las soluciones. Nuestro procedimiento es una variante de 2-opt de Lin [158]. En este método se eligen aleatoriamente dos posiciones, y entonces se invierte la subpermutación existente entre esos dos ejes. Cuando el movimiento mejora la solución, se actualiza la solución actual de este método y se continúa hasta que se alcance un número predeterminado de operación de intercambio.

Versión Paralela

Como vimos en el Capítulo 2 se han propuesto varias alternativas para la paralelización de la SS. Nuestro objetivo es conseguir una mejora en la calidad de las soluciones (a parte del obvio ahorro en tiempo de cómputo). Por lo tanto, modelos como el maestro-esclavo no han sido considerados.

Finalmente nuestra implementación está basada en el modelo distribuido, donde tenemos varios SS secuenciales ejecutándose en paralelo y cada cierto número de iteraciones se intercambian una

solución del RefSet. La topología de comunicación es la misma que la utilizada en el GA paralelo. Las soluciones emigrantes son elegidas por torneo binario, permitiendo así enviar tanto soluciones de buena calidad (que saldrían del RefSet_1) como soluciones que permitan aumentar la diversidad (RefSet_2). Las soluciones emigrantes son introducidas en la población son introducidas en la población siguiendo el esquema general de actualización del conjunto de referencia.

Para que el coste computacional entre el SS paralelo y el secuencial sean similares, se ha decidido reducir el número de combinaciones consideradas por el método paralelo. En concreto, el número de combinaciones de soluciones analizadas en cada paso, es el mismo que el producido en la versión secuencial pero dividido entre el número de islas. En este caso, los subconjuntos de soluciones son generados de forma aleatoria, pero sin permitir que el mismo subconjunto sea seleccionado más de una vez.

8.2.5. Detalles Específicos con SA

Esquema de Enfriamiento

Como vimos anteriormente el esquema de enfriado controla los valores de la temperatura. Este método especifica de valor inicial y cómo se actualiza la temperatura en cada paso del algoritmo. En el apartado correspondiente al SA del Capítulo 2, vimos que existían diferentes métodos. En concreto en este problema, nosotros usamos el esquema proporcional ($T \leftarrow \alpha \cdot T$).

Operador de Movimiento

Este operador genera un nuevo vecino a partir de la solución actual. Para este problema hemos usado el operador de intercambio. Este operador selecciona de forma aleatoria dos posiciones de la permutación e intercambio los fragmentos de estas posiciones

Versión Paralela

Para la paralelización del SA, hemos seguido el modelo de múltiples ejecuciones con cooperación (Sección 4.3.1). En concreto el SA paralelo (PSA) consta de múltiples SAs que se ejecutan de manera asíncrona. Cada uno de estos SAs empiezan de una solución aleatoria diferente, y cada cierto número de evaluaciones (*fase de cooperación*) intercambian la mejor solución encontrada hasta el momento. El intercambio de información se produce en una topología de anillo unidireccional. Para el proceso de aceptación de la nueva solución inmigrante, se aplica el mecanismo típico del SA, es decir, si es mejor, se acepta inmediatamente y si es peor, puede ser aceptada dependiendo de cierta distribución de probabilidad que depende del fitness y la temperatura actual.

8.3. Análisis de los Resultados

Ahora pasamos a comentar los resultados obtenidos al aplicar nuestras propuestas algorítmicas al problema de ensamblado de fragmentos de ADN. Aquí se muestra un resumen de los más interesantes, los resultados completos se pueden consultar en [16], [26] y [168]. Primero presentaremos las instancias usadas y la configuración de los algoritmos aplicada. Entonces, analizamos el comportamiento de los algoritmos tanto respecto a su capacidad de producir buenas soluciones como su capacidad para reducir el tiempo de ejecución.

Los algoritmos en este trabajo han sido implementado bajo las especificaciones de Mallba en C++ y han sido ejecutados en un clúster de Pentium 4 a 2.8 GHz con 512 MB de memoria y SuSE Linux 8.1 (kernel 2.4.19-4GB). La red de comunicación es una Fast-Ethernet a 100 Mbps.

Tabla 8.1: Parámetros para la resolución del problema de ensamblado de fragmentos de ADN.

Parámetros Comunes	
Ejecuciones independientes	30
Función de Evaluación	F1
Umbral de corte (*Cutoff*)	30
Número de Evaluaciones	512000
Frecuencia de migración	20 iteraciones
Ratio de migración	1
Algoritmo Genético	
Tamaño de la población	1024
Recombinación	OX ($\rho_c = 0{,}8$)
Mutación	Intercambio ($\rho_m = 0{,}2$)
Selección	Ranking
CHC	
Tamaño de la población	50
Recombinación	específico ($\rho_c = 1{,}0$)
Reinicio	Intercambio (Diferencia con la original 30 %)
Búsqueda Dispersa	
Población inicial	15
Conjunto de referencia	8 (5 + 3)
Generación de subconjuntos	Todos los subconjuntos de 2 elementos (28)
Recombinación	OX ($\rho_c = 1{,}0$)
Operador de mejora	Intercambio (100 iteraciones)
Enfriamiento Simulado	
Operador de movimiento	Intercambio
Long. de la cadena de Markov	$\frac{maximo\ evaluaciones}{100}$
Esquema de enfriamiento	Proporcional ($\alpha = 0{,}99$)
Fase de comunicación	500 iteraciones

8.3.1. Instancias y Parámetros

En este trabajo hemos usado la secuencia destino que tiene como número de acceso BX842596 (GI 38524243). Se puede obtener del sitio web del NCBI[1]. Esta secuencia corresponde con Neurospora crassa BAC, y tiene 77.292 bases. Para analizar el rendimiento de nuestros acercamientos algorítmicos, hemos generado dos instancias con el GenFrag [87]. Este software toma una secuencia de ADN conocida y la usa para generar aleatoriamente fragmentos de la misma que siguen algún criterio aportado por el usuario (por ejemplo, una determinada cobertura y un tamaño medio del fragmentos especificado). La primera instancia, 842596_4, contiene 442 fragmento con un tamaño medio de 708 bases y una cobertura de 4. La segunda instancia, 842596_7, contiene 773 fragmentos de un tamaño medio de 703 bases y una cobertura de 7.

Para poder realizar una análisis justo de los resultados de los diferentes algoritmos secuenciales y paralelos, los hemos configurado para que realicen un esfuerzo computacional similar todos ellos (un máximo número de 512000 evaluaciones). La búsqueda dispersa es la única excepción, puesto que su proceso más pesado es el método de búsqueda local que utiliza para mejorar las soluciones. Este método no realiza evaluaciones completas de los individuos, sino que sólo necesita saber si la nueva solución es mejor o peor que la actual. Por lo tanto, hemos configurado el SS para que

[1] `http://www.ncbi.nlm.nih.gov`

su tiempo de cómputo sea similar al del GA. Puesto que los resultados pueden variar mucho de acuerdo de su configuración, previamente hemos realizado un completo análisis de como afectan los diferentes parámetros al rendimiento de los algoritmos. De esos estudios hemos llegado a la conclusión que la mejor configuración para las instancias usadas es la indicada en la Tabla 8.1.

8.3.2. Resultados Secuenciales

A continuación empezaremos a analizar los resultados ofrecidos por las versiones secuenciales, antes de pasar a estudiar los resultados de las versiones paralelas y su comparación. Los valores analizados serán: el mejor valor de fitness obtenido (b), el fitness medio (f), el número medio de evaluaciones (e), y el tiempo medio en segundos (t). Los mejores resultados se indican en negrita. Sólo indicamos la media, porque la desviación estándar u otras métricas no son muy indicativas, ya que las fluctuaciones en la precisión de las diferentes ejecuciones son pequeñas, lo que prueba la robustez de los algoritmos probados. Estos valores también se ven avalada su significancia estadística por unos test ANOVA realizados [168].

Tabla 8.2: Resultados para las versiones secuenciales.

	38524243_4				**38524243_7**			
Algoritmo	b	f	e	t	b	f	e	t
GA	92772	88996	508471	32.62	108297	104330	499421	85.77
CHC	61423	54973	487698	65.33	86239	81943	490815	162.29
SS	94127	90341	**51142**	27.83	262317	254842	**52916**	66.21
SA	**225744**	**223994**	504850	**7.92**	**416838**	**411818**	501731	**12.52**

La Tabla 8.2 muestra los resultados con las versiones secuenciales de los algoritmos probados. La primera conclusión es que la versión secuencial del SA es el que mejor rendimiento ofrece. Por un lado que el algoritmo SA supera al resto de los algoritmos en tanto en calidad de la solución encontrada como tiempo. La razón por la que el SA requiera mucho menos tiempo para encontrar sus mejores soluciones, es debido que este método opera sobre una única solución, mientras que el resto de algoritmos trabajan sobre una población y además ejecutan operadores que consumen mucho tiempo (especialmente el operador de cruce).

El CHC es el peor algoritmo tanto en calidad de soluciones encontradas como en tiempo de ejecución. El tiempo adicional que necesita este algoritmo respecto al resto de métodos basados en población es debido a las operaciones para detectar la convergencia de la población y la detección del incesto. El CHC no es capaz de resolver de forma adecuada el problema que tratamos en este capítulo y quizás para mejorar su rendimiento, necesite un mecanismo de búsqueda local (como recomienda el autor del método [89]) para reducir el espacio de búsqueda.

El SS obtiene mejores resultados que el GA, y también logra esas soluciones en un tiempo menor. Esto significa que la aplicación estructurada de un operador de mejora llevada acabo por el SS es una buena idea a la hora de resolver este problema.

Tras analizar la calidad de soluciones y el tiempo de ejecución (el cual analizaremos en más detalle cuando tratemos las versiones paralelas), ahora nos vamos a fijar en el esfuerzo computacional (número de evaluaciones) realizado por los algoritmos a la hora de resolver el problema. Se puede observar en la Tabla 8.2 que en general no existen grandes diferencias entre los algoritmos e incluso en algunos casos esas diferencias no son significativas estadísticamente. Como se comentó al comienzo de esta sección, la excepción es la búsqueda dispersa, ya que su operación más costosa es el método de mejora que no realizar evaluaciones completas. Este resultado indica que todos los algoritmos examinan un número similar de puntos del espacio de búsqueda y que la diferencia en la calidad de las soluciones encontradas es debida a cómo hacen esa exploración. En este problema,

los métodos basados en trayectoria como es el caso del enfriamiento simulado es más efectiva que los métodos basados en población. Así, una ordenación de los métodos de mejor a peor podría ser: SA, SS, GA y por último el CHC.

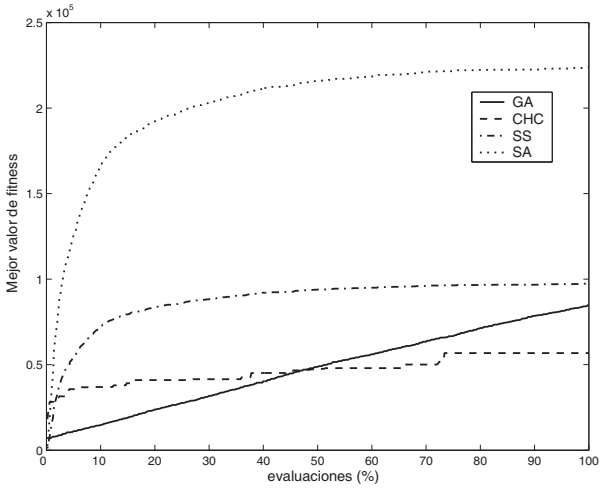


Figura 8.3: Evolución del mejor fitness para la instancia 38524243_4.

En la Figura 8.3 se muestra la traza de la evolución de la mejor solución durante la ejecución de todos los heurísticos para la primera instancia. Analicemos el comportamiento de cada algoritmo. El GA va mejorando de forma continua y estable su mejor solución, y posiblemente con un número mayor de evaluaciones podría obtener mejores soluciones, aunque su convergencia es muy lenta. De hecho, se hicieron pruebas con un número mucho mayor de evaluaciones y aunque es cierto que se producía una mejora, esta mejora era pequeña y ni siquiera alcanzaba la calidad de las soluciones del SS. En el comportamiento del CHC, observamos que converge demasiado rápido a una solución subóptima y que le resulta muy difícil escapar de ella, pese a los mecanismos que implementa para estos casos (la reinicialización de la población). Finalmente, la búsqueda dispersa y el enfriamiento simulado mejoran de forma la calidad de las soluciones en el inicio de la ejecución, pero ya después la búsqueda avanza más lentamente. En esa figura podemos observar como la pendiente de crecimiento del SA es bastante más rápida que la del resto de algoritmos.

Tabla 8.3: Mejor número de contigs final.

Algoritmo	**38524243_4**	**38524243_7**
GA	6	4
CHC	7	5
SS	6	4
SA	**4**	**2**

Acabamos el análisis de los resultados secuenciales estudiando el número de contigs calculados en cada caso (Tabla 8.3). Los valores de la tabla confirman otra vez que el SA supera claramente al resto de los algoritmos, mientras que el CHC obtiene los peores resultados y el SS y GA obtienen un número muy similar de contigs.

8.4. Resultados Paralelos

A continuación mostramos los resultados para ambas instancias utilizando las versiones paralelas de los algoritmos utilizando 8 procesadores. En la Tabla 8.4 se muestra un resumen de los resultados obtenidos. En este caso no se muestran los test estadísticos, porque las diferencias siempre son significativos y la inclusión de una tabla con los test no aporta nada. Estos valores pueden ser examinados según diferentes criterios. Si se examinan por tiempo se puede observar como en todos los casos se obtiene una importante reducción del tiempo total de ejecución.

Tabla 8.4: Resultados de las versiones paralelas para ambas instancias.

	38524243_4				**38524243_7**			
Algoritmo	b	f	e	t	b	f	e	t
GA	107148	90154	733909	4.7	156969	116605	611694	13.42
CHC	80184	68653	527059	8.1	87429	824518	481694	24.51
SS	149315	134842	**60141**	4.2	305234	297425	**61624**	8.8
SA	**176157**	**141684**	713932	1.2	**383676**	**361238**	576515	1.8

Si analizamos de acuerdo a la calidad de las soluciones encontradas, encontramos resultados dispares; en general, las versiones paralelas mejoran las soluciones que se obtenían con las versiones secuenciales. Pero existe un importante excepción, el paralelizar el SA se produce un bajada en la calidad de las soluciones encontradas. Nuestra hipótesis es que al tener varios SAs ejecutándose simultáneamente, para el que esfuerzo sea similar al secuencial, se ha tenido que reducir el número de evaluaciones de cada componente, lo cual pese a la cooperación entre los subalgoritmos ha sido muy perjudicial para el comportamiento global del algoritmo. El esfuerzo computacional en general no sufre una gran variación al paralelizar los métodos.

Tabla 8.5: Speedup.

	38524243_4			**38524243_7**		
Algoritmo	8 CPUs	1 CPU	Speedup	8 CPUs	1 CPU	Speedup
GA	4.7	32.8	6.98	13.42	85.77	6.39
CHC	8.1	63.5	7.83	24.51	162.29	6.62
SS	4.2	28.4	6.76	8.8	66.21	7.52
SA	1.2	8.0	6.67	1.8	12.52	6.95

En la Tabla 8.5 muestra el speedup para los algoritmos paralelos. En este caso utilizamos la variante de speedup débil, que compara la versión paralela ejecutada en una plataforma paralela con el mismo algoritmo ejecutada en un sistema monoprocesador. En estos resultados podemos observar como en todos los casos el speedup es sublineal, aunque la pérdida de eficiencia que se produce es de carácter moderado. Cabe destacar el caso del CHC para la primera instancia y la del SS en la segunda instancia, donde el speedup es casi lineal (con 7.83 y 7.52, respectivamente). No se observan grandes diferencias entre los diferentes algoritmos, oscilando el valor obtenido entre el 6.67 y el 7.83.

Finalmente en la Tabla 8.6 mostramos el número de contigs producidos por las mejores soluciones halladas por cada algoritmo. Usamos este valor como criterio de alto nivel para juzgar la calidad completa de los resultados, ya que como dijimos antes, es difícil capturar la dinámica del problema con una función matemática. Estos valores se calculan, mediante la aplicación de un último paso de refinamiento con un método ávido (*greedy*) muy popular en este aplicación [156]. El problema de esta medida, es que existe un gran salto entre el fitness calculado y el número de contigs resultante. Esto puede llevar a situaciones extremas donde una solución con un fitness mejor

Tabla 8.6: Mejor número de contigs final.

Algoritmo	**38524243_4**	**38524243_7**
GA	5	3
CHC	7	5
SS	**4**	**2**
SA	**4**	**2**

que otra, produzca un número de contigs más altos que esta segunda que inicialmente parecía peor. Esto es un claro indicativo de que se necesita una mayor investigación para conseguir un función de fitness más precisa. De hecho, esta situación nos llevó en una fase posterior a estos resultados a desarrollar una nueva función de fitness (Ecuación 8.4), cuyos resultados se dicusten en el Capítulo 5. Observamos que pese a la bajada de la calidad de las soluciones del SA respecto a las versiones secuenciales, el número de contigs no ha empeorado, indicando que un amplio rango del valor de fitness producen soluciones equivalentes, por lo que se hace necesaria que en próximos estudios se utilice una ecuación más adecuada al problema. En general se observa una mejora en el número de contigs encontrada respecto a los secuenciales, en especial, en el caso del SS, que consigue alcanzar los mejores resultados obtenidos por el SA, que es el que mejores soluciones produce.

8.5. Conclusiones

El problema de ensamblado de fragmentos de ADN es un problema muy complejo del campo de la biología computacional. Puesto que un problema NP-Duro, encontrar la solución óptima mediante algoritmos exactos es imposible para la mayoría casos reales, a excepción de instancias muy pequeñas. Por lo tanto, se necesitan técnicas con una complejidad menor que permitan tratar con este problema.

En nuestro estudio del problema hemos trabajado con cuatro técnicas metaheurísticas: tres basadas en población (un algoritmo genético, una búsqueda dispersa y un CHC) y un método basado en trayectoria (el enfriamiento simulado). Los resultados obtenidos han sido muy interesantes. Por un lado hemos visto que el paralelismo ha permitido reducir de forma importante el tiempo de cómputo en todos los métodos y en la mayoría a contribuido a aumentar la calidad de las soluciones. Pero también se ha mostrado que el paralelismo no siempre el útil, como se demuestra en el caso del enfriamiento simulado; aunque es cierto que la versión paralela permite una reducción en el tiempo total de cálculo, la calidad de las soluciones conseguidas son peores que las del algoritmo secuencial. Pensamos que esta situación es debida a la reducción en el número de evaluaciones realizadas por cada componente que se debe realizar para que la comparativa con el algoritmo secuencial sea justa. Esta reducción de soluciones exploradas por cada componente no se compensa con la comunicación entre las diversas componente, y provocan una caída en el rendimiento global del método. Por otro lado, hemos observado que el CHC tanto en secuencial como en paralelo obtiene unos pobres resultados debido a una convergencia muy rápida a una solución subóptima en los pasos iniciales de la búsqueda.

Finalmente, comentar que nuestras aproximaciones algorítmicas han permitido la resolución de una manera bastante satisfactoria una instancia de 77k bases, lo cual es un logro muy importante, ya que hasta ahora el tamaño de las instancias aparecidas en la literatura del problema no superaban la longitud de 30-50k bases.

Capítulo 9

Resolución del Problema del Etiquetado Léxico del Lenguaje Natural

En este capítulo analizaremos el comportamiento de diferentes metaheurísticas y estudiamos sus ventajas relativas para resolver el problema del etiquetado del lenguaje natural. Este problema consiste en asignar a cada palabra de un texto su cometido léxico de acuerdo con el contexto en el que esté utilizada. Especialmente, hemos desarrollado versiones específicas del GA, el CHC y el SA para tratar con este problema, tanto en versión secuencial como paralelo. Tras describir el problema y como lo hemos abordado con los métodos antes mencionados, haremos dos fases de prueba. Una primera donde estudiamos los algoritmos y la influencia de ciertos aspectos, como el tamaño de la población o el tipo de codificación usada. Una vez con un conocimiento más profundo, abordaremos otros corpus más complejos y compararemos nuestros resultados con otros algoritmos típicos dentro del campo del procesamiento del lenguaje natural.

9.1. Definición del Problema

El etiquetado léxico o simplemente "etiquetado" es una de las primeras tareas que hay que realizar en el procesamiento del lenguaje natural (NLP). El etiquetado cosiste en determinar cual etiqueta léxica hay que aplicar a cada palabra dentro de un texto. Por ejemplo, la palabra *can*[1] puede ser un nombre, un verbo auxiliar o un verbo transitivo. La categoría que se asigne a una palabra determina la estructura de la frase donde aparece así como su significado. Por ejemplo, consideremos el siguiente ejemplo, extraído de un titular de un periódicos cuyo significado puede cambiar dramáticamente según las diferentes etiquetas que se les asigne a las palabras:

Teacher *strikes idle* kids

Podemos observar que según la etiqueta que asignemos a las palabras enfatizadas, las frase adquiere diferentes sentidos: "El profesor pega a los niños ociosos" (considerando *strikes* como verbo e *idle* como adjetivo) o "Los golpes del profesor hacen ociosos a los niños" (considerando *strikes* como nombre e *idle* como verbo). De hecho, el etiquetado léxico es un paso necesario para

[1]Los ejemplos de este problema aparecerán en inglés, ya que este problema es muy dependiente del idioma y su traducción cambia totalmente el sentido.

el análisis sintáctico [208], los sistemas de recuperación de información [41], reconocimiento del lenguaje, etc [170]. Más aún, el etiquetado en sí mismo es un problema difícil debido a que muchas palabras puede pertenecer a más clase léxica. Para dar una idea de su complejidad decir que de acuerdo a [77], sobre el 40 % de las palabras que aparecen en el corpus de Brown (etiquetado a mano) [195] son ambiguas.

Debido a la importancia de esta tarea, se han desarrollado muchos trabajos para producir etiquetadores automáticos. Los etiquetados automáticos [59, 180, 234], normalmente están basado en modelos de Markov ocultos, que confían en la información estadística para establecer las probabilidades de cada escenario. Los datos estadísticos que son extraídos de textos previamente etiquetados se les denomina *corpus*. Estos etiquetadores estocásticos no requieren conocer las reglas del lenguaje ni intentan deducirlas, y así ellos pueden ser aplicados a textos de cualquier lenguaje, siempre que se proporcione para ser entrenado de un corpus en ese lenguaje previamente.

El contexto en el cual aparece una palabra ayuda a decidir cual es la etiqueta más apropiada, siendo esta la idea básica en la que se sustentan la mayoría de los etiquetadores. Por ejemplo, consideremos la frase de la Figura 9.1, que está extraída del corpus de Brown. La palabra *questioning* puede ser etiquetada como un nombre común si la etiqueta precedente es un adjetivo. Pero puede ocurrir que la palabra precedente también fuese ambigua, y así sucesivamente, con lo que habría que resolver varias dependencias simultáneamente.

This	the	therapist	may	pursue	in	later	questioning	.
DT	AT	NN	NNP	VB	RP	RP	VB	.
QL			MD	VBP	NNP	RB	NN	
					RB	JJ	JJ	
					NN	JJR		
					FW			
					IN			

Figura 9.1: Etiquetas posibles para una sentencia extraída del corpus de Brown. Las etiquetas subrayadas se corresponden con las correctas de acuerdo con el corpus de Brown. En este caso: DT es el acrónimo para determinante/pronomble, AT para artículo, N para nombre común, MD para verbo modal auxiliar, IN para preposición, etc
.

El modelo estadístico considerado en este capítulo asignar la etiqueta de aquellos contextos con mayor probabilidad. Asignar la etiqueta con mayor probabilidad no siempre es posible, ya que al fijar una etiqueta, limitas las posibilidades de las de alrededor y quizás impida asignar la etiqueta correspondiente al contexto con mayor probabilidad. Por lo tanto necesitamos un método que maximice de forma global esta medida de probabilidad.

Con esas intenciones en este capítulo proponemos diferentes metaheurísticas para realizar esa búsqueda. En concreto los algoritmo que hemos desarrollado para resolver este problema son un algoritmo genético, un CHC y un enfriamiento simulado. Una de las ventajas de usar los algoritmos evolutivos como algoritmo de búsqueda para el etiquetado léxico es que estos algoritmos pueden aplicarse a cualquier modelo estadísticos, incluso a aquellos que no aseguran las hipótesis de Markov (como por ejemplo, que la etiqueta de una palabra depende de las palabras previas), como son requeridos por los algoritmos clásicos para el etiquetado léxico, siendo su mayor exponente el algoritmo Viterbi [59]. Los algoritmos genéticos han sido aplicados con éxito al problema previamente [37, 38].

Debido que el proceso de búsqueda es muy costoso en tiempo, nos ha llevado a desarrollar versiones de los algoritmos antes descritos para aligerar el cómputo. En una primera fase estudiaremos

el comportamiento de estos algoritmos en el corpus de Brown, probando diferentes alternativas de codificación de las soluciones y otros parámetros como el tamaño de la población. Una vez tengamos un conocimiento más exhaustivo del comportamiento de los algoritmos, probaremos sus mejores configuraciones en otro corpus (el de Susanne) y la validaremos los resultados comparándolos con un método de etiquetado específico y que ampliamente utilizado en el campo del lenguaje natural, el método de Viterbi.

Antes de describir cómo hemos aplicado estos algoritmos al problema, describiremos los tipos de modelos estadísticos a los cuales se les pueden aplicar.

9.1.1. Aproximación Estadística al Etiquetado Léxico

El etiquetado estadístico es probablemente la aproximación más extendida actualmente, se basa en un modelo estadístico definidos de acuerdo a unos parámetros que pueden ser extraídos de los textos etiquetados.

La meta de estos modelos es asignar a cada palabra la etiqueta léxica más apropiada de acuerdo al *contexto* de la palabra, es decir, de las palabras próximas. Por lo tanto, debemos recolectar de estadísticas de para esa palabra cuales son los contextos más habituales y a partir de esa información asignarle a la etiqueta correcta. Pero las palabras cercanas también pueden ser ambiguas por lo que se necesita alguna clase de modelo estadístico para seleccionar el "mejor" etiquetado para la secuencia global de acuerdo con el modelo. Más formalmente, el problema del etiquetado léxico puede ser definido como:

$$t_{1,n} = \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n})$$

donde $\arg\max_x f(x)$ es el valor de x que maximiza $f(x)$, y $t_{1,n}$ es la secuencia de etiquetas de las palabras $w_{1,n}$ que componen el texto que se desea etiquetar.

Si asumimos que la etiqueta de una palabra sólo depende de la etiqueta previa, y que esta dependencia no cambia a través del tiempo, podemos adoptar un modelo de Markov para el etiquetado. Sea w_i la palabra en la posición i en el texto, t_i la etiqueta de w_i, $w_{i,j}$ las palabras que aparecen desde la posición i hasta la j, y $t_{i,j}$ las etiquetas de las palabras $w_{i,j}$. Entonces el modelo establece que:

$$P(t_{i+1}|t_{1,i}) = P(t_{i+1}|t_i)$$

Si también se asume que la probabilid de una palabra que aparece en una posición particular sólo depende en la etiqueta asignada a esa posición, la secuencia óptima de etiquetas se puede estimar como:

$$\begin{aligned} t_{1,n} &= \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \\ &= \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \end{aligned}$$

Como consecuencia, los parámetros de etiquetado basado en el modelo de Markov pueden ser computados desde el corpus de entrenamiento. Esto se puede hacer mediante el almacenamiento en una tabla (*tabla de entrenamiento*) los diferentes contextos de cada etiqueta. Esta tabla puede ser calculada recorriendo el texto de entrenamiento y almacenando los diferentes contextos y el número de ocurrencias para cada uno de ellos.

El modelo de Markov descrito arriba es conocido como *bigram* ya que solo hace predicciones basándose en la etiqueta precedente, es decir, la unidad básicas considerada está compuesta de dos etiquetas: la etiqueta precedente y la actual. Este modelo puede extenderse de tal forma que las predicciones dependan de más de una etiqueta. Por ejemplo, el modelo *trigram* hace sus predicciones dependiendo en las dos etiquetas precedentes. En un principio, se podrían esperar predicciones más precisas cuando el contexto se agranda. Sin embargo, en la práctica, un etiquetador que sigue un modelo trigram suele hacer peores predicciones que uno que siga el bigram debido a problema de

datos dispersos, ya que almacenar estadísticas significativas para contextos trigram requiere un conjunto de entrenamiento mucho mayor.

Una vez que el modelo estadístico ya ha sido definido, la mayoría de los etiquetadores usan el algoritmo de Viterbi [98] (un algoritmo de programación dinámica) para encontrar la secuencia de etiquetas que maximice la probabilidad de acuerdo al modelo de Markov elegido.

En este capítulo ofrecemos una alternativa al etiquetado que puede ser usado en vez del algoritmo de Viterbi. Esta aproximación se basa en la utilización de algoritmos evolutivos. Estos métodos proporcionan un mecanismo general que puede ser aplicado a cualquier modelo estadístico. Por ejemplo, puede ser aplicado para realizar el etiquetado de acuerdo al modelo de Markov definido arriba o cualquier otro modelo. Por ejemplo, podríamos aplicarlos a un modelo que ha probado ser mejor que el de Markov [229], en el cual el contexto de la palabra es compuesto tanto de palabras precedentes como de las que siguen a la actual. Por ejemplo, si consideramos contextos con dos etiquetetas a la derecha y otras dos a la izquierda, las entradas en la tabla de entrenamiento de la etiqueta *JJ* podría tener la siguiente estructura:

```
JJ 4557 9519

VBD AT JJ NN IN 37
IN PP\$ JJ NNS NULL 20
PPS BEZ JJ TO VB 18
       ...
```

observa que *JJ* tiene 4557 contextos diferentes y aparece 9519 veces en el texto, y que el primer contexto aparece 37 veces, el segundo 20 y así sucesivamente hasta los 4557 contextos diferentes.

9.2. Aproximación algorítmica para su resolución

En esta sección se comentará las aproximaciones elegidas para tratar con este problema mediante diferentes algoritmos.

9.2.1. Detalles comunes

Primero comentamos los detalles comunes a todos los heurísticos utilizados.

Representación

Oración	This	the	therapist	may	pursue	in	later	questioning	.
Ind. 1:	DT	AT	NN	NNP	VBP	IN	JJ	VB	.
Ind. 2:	DT	AT	NN	MD	VB	RB	RB	NN	.
Ind. 3:	QL	AT	NN	NNP	VB	FW	JJ	JJ	.

Figura 9.2: Posibles individuos para la oración de la Figura 9.1.

Las soluciones tentativas son secuencias de genes que corresponden a cada palabra en la sentencia que debe ser etiquetada. La Figura 9.2 muestra algunos posibles individuos para la oración de la Figura 9.1. Cada gen representa una etiqueta y cualquier otra información útil para la evaluación del cromosoma, como el número de contextos para esta palabra en la tabla de entrenamiento. Cada etiqueta de un gen representa el índice al vector que contiene todas las posibles etiquetas para la

palabra. La composición de los genes dependen de la codificación elegida como se muestra en la Figura 9.3. En la codificación entera, el gen solo contiene el entero correspondiente al índice. En la codificación binaria contiene la representación binaria de ese índice. Nosotros utilizamos diferentes longitudes de bits para codificar cada índice, como mínimo es 3 que es el valor que permite codificar los 6 posibles índices que puede tener como máximo una palabra.

palabra	**Índice Etiqueta**							**Int**	**Bin(7)**	**Bin(3)**
	0	1	2	3	4	5	···			
This	DT	QL						0	0000000	000
the	AT							0	0000000	000
therapist	NN							0	0000000	000
may	NNP	MD						1	0000001	001
pursue	VB	VBP						0	0000000	000
in	RP	NNP	RB	NN	FW	IN		5	0000101	101
later	RP	RB	JJ	JJR				3	0000011	011
questioning	VB	NN	JJ					1	0000001	001

Figura 9.3: Codificaciones enteras y binarias para cada gen (palabra) indicando la etiqueta elegida (subrayada) entre todas las posibles.

Función de Evaluación

La función de evaluación de un individuo es una medida de la probabilidad de que sea correcta en la secuencia de etiquetas, de acuerdo con los datos almacenados en la tabla de entrenamiento. Este valor es calculado como la suma del fitness de sus genes, $\sum_i f(g_i)$. el fitness de un gen se define como

$$f(g) = \log P(T|LC, RC)$$

donde $P(T|LC, RC)$ es la probabilidad de que la etiqueta del gen g sea T, dados los contextos izquierdo LC y el RC. Esta probabilidad se estimada estima con los datos de la tabla de entrenamiento siguiendo la siguiente fórmula

$$P(T|LC, RC) \approx \frac{occ(LC, T, RC)}{\sum_{T' \in \mathcal{T}} occ(LC, T', RC)}$$

donde $occ(LC, T, RC)$ es el número de ocurrencias de la lista de etiquetas LC, T, RC en la tabla de entrenamiento, y $\mathcal{T}$ es el conjunto de todas las posibles etiquetas del gen g_i.

9.2.2. Detalles Específicos con GA

Una vez definido la representación, los principales detalles que ha que decidir son los operadores utilizados. Como operador de recombinación usamos el cruce de un punto, es decir, se elige de forma aleatoria un punto aleatorio y la primera parte de un padre es combinada con la segunda del otro para así producir dos hijos. Como operador de mutación, reemplazamos la etiqueta actual por otra etiqueta válida elegida de acuerdo a su probabilidad (la frecuencia en que aparece en el corpus).

A la hora de paralelizar el GA, hemos seguido un modelo distribuido (Sección 4.3.1). En concreto, tenemos un conjunto de islas ejecutando un GA secuencial en cada una, y cooperan entre sí, enviando una solución elegida por torneo binario cada 10 iteraciones a su vecino. La vecindad entre las islas se define por medio de una topología en anillo unidireccional.

9.2.3. Detalles Específicos con CHC

En este caso utilizamos el CHC tal como está descrito en la Sección 2.1.2.

Para poder realizar una comparación justa con respecto a nuestro algoritmo de referencia, el GA, hemos paralelizado este algoritmo siguiendo exactamente el mismo esquema explicado en la sección anterior.

9.2.4. Detalles Específicos con SA

Para instancias este algoritmo con este problema en concreto se debe definir la estrategia de actualización utilizada y el mecanismo para explorar el vecindario.

Como mecanismo de actualización de la temperatura usamos el esquema proporcional ($T = \alpha \cdot T$) que ya presentamos en el Capítulo 2. Para crear un vecino a partir del actual utilizamos el mismo mecanismo que utiliza el GA para mutar las soluciones, es decir, cambiar la etiqueta actual de la palabra por otra válida de acuerdo con su probabilidad.

Para la paralelización del SA, hemos seguido el modelo de múltiples ejecuciones con cooperación (Capítulo 2). En concreto el SA paralelo (PSA) consta de múltiples SAs que se ejecutan de manera asíncrona. Cada uno de estos SAs empiezan de una solución aleatoria diferente, y cada 100 evaluaciones (*fase de cooperación*) intercambian la mejor solución encontrada hasta el momento. El intercambio de información se produce en una topología de anillo unidireccional. Para el proceso de aceptación de la nueva solución inmigrante, se aplica el mecanismo típico del SA, es decir, si es mejor, se acepta inmediatamente y si es peor, puede ser aceptada dependiendo de cierta distribución de probabilidad que depende del fitness y la temperatura actual.

9.3. Análisis de los Resultados

Ahora pasamos a describir y analizar los experimentos realizados. Hemos realizado dos tandas de experimentos. La primera era para estudiar el comportamiento de los algoritmos a la hora de abordar el problema y estudiar la influencia de ciertos parámetros como la codificación, el tamaño de la población o el comportamiento del modelo paralelo. Una vez concluidos hemos experimentos y aprovechando el conocimiento extraído de ellos, haciendo una segunda tanda de experimentos, donde probamos nuestras propuestas en varios corpus diferentes comparando sus resultados con un algoritmo muy establecido en el campo del lenguaje natural como es el algoritmo Viterbi.

9.3.1. Experimento 1: Análisis del Comportamiento

En esta primera fase experimental hemos utilizado como texto de entrenamiento para nuestros etiquetadores el corpus de Brown [195], uno de los más extendidos en lingüistica. El conjunto de etiquedas de este corpus nos es muy grande, que favorece la precisión del sistema. Además, también se ha reducido el conjunto de etiquetas mediante el agrupamiento de algunas relacionadas en una única etiqueta, lo que que mejora las estadísticas. Por ejemplo, las diferentes clases de adjetivos (JJ, $JJ + JJ$, JJR, $JJR + CS$, JJS, JJT) que estaban distinguidos en el corpus han sido agrupadas bajo la etiqueta común JJ.

Los parámetros usados en estos experimentos son los siguientes: hemos utilizado un ratio de cruce de 50 % ($rho_c = 0{,}5$)para el CHC. El GA aplica el operador de recombinación también con una probabilidad de 0.5, y la mutación con 0.05. En las versiones paralelas, la migración se produce cada 10 generaciones.

En las Tablas 9.1 y 9.2 muestran los resultados obtenidos con los algoritmos CHC y GA, utilizando diferentes representaciones y tamaños de población. Analizamos el impacto de la repre-

Tabla 9.1: Precisión del etiquetado obtenido con el CHC para el texto de prueba con 2500 palabras. PS indica tamaño de la población.

Cont.	**CHC-Int**				**CHC-Bin(7)**				**CHC-Bin(4)**			
	PS = 20		PS = 56		PS = 20		PS = 56		PS = 20		PS = 56	
	Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**
1-0	89.96	90.15	89.34	89.34	92.08	92.17	91.04	90.95	91.94	**92.53**	91.18	91.35
2-0	91.41	91.68	90.91	91.32	93.34	93.43	92.35	91.90	93.38	**93.52**	92.04	92.40
3-0	92.58	92.89	91.68	91.72	93.74	93.97	92.98	93.07	93.92	**93.97**	93.16	93.25
1-1	93.12	93.48	92.39	92.58	94.78	94.97	93.88	94.06	**95.14**	94.82	94.06	94.10
2-1	93.56	93.70	93.07	93.21	94.51	94.51	93.83	94.06	94.47	**94.65**	93.83	94.15
2-2	94.51	94.11	94.29	93.98	94.61	94.73	94.78	94.78	95.01	**95.23**	94.06	94.87

sentación a dos niveles: por un lado comparamos una representación entera con otra binaria, y por otro lado estudiamos el efecto de la longitud en bits utilizados para la codificación binaria (7 o 4 bits). En cada columna de las tablas consideramos una clase de contexto: 1-0 (también denominado bigram) es un contexto que considera únicamente una etiqueta al lado izquierda de la palabra actual, 1-1 que considera tanto la etiqueta precedente como la siguiente, y así sucesivamente. Los datos mostrados muestran los mejores resultados de 20 ejecuciones independientes. Los mejores valores para cada contexto aparece en negrita. **Int** representa la representación entera, **Bin(7)** para la representación binaria con 7 bits, y **Bin(4)** para el caso binario con 4 bits. Estas medidas han sido tomadas para dos tamaños diferentes de población, una de 56 individuos y otra de 20. Además, también hemos analizado tanto las versiones secuenciales como paralelas con 4 islas. En el caso paralelo, el tamaño de la población de cada isla, es el tamaño de la población secuencial dividido entre el número de islas.

Analizando la Tabla 9.1, podemos extraer diferente conclusiones. En primer lugar se puede observar que la codificación binaria logra una mejor precisión que cuando se utiliza la entera. Además cuanto más corta es la representación de la etiqueta, mejores resultado ofrece. Esto sugiere que la representación entera no es apropiada para el CHC, ya que produce un número muy pequeño de genes lo que dificulta la aplicación del mecanismo de prevención de incesto. Respecto a las ejecuciones paralelas, queda bastante claro que el paralelismo permite una clara mejora en la precisión, obteniendo los mejores resultados en 4 de los 5 contextos probados. Además esos resultados son logrados utilizando la población más pequeña (20 individuos), ya que en poblaciones tan pequeñas, el paralelismo permite mantener la diversidad.

La Tabla 9.2 muestra los resultados obtenidos por el GA. En este caso, el comportamiento del GA es diferente del que acabamos de observar en los resultados del CHC. El GA obtiene un mejor rendimiento cuando utiliza una representación entera y con una pequeña población. En el caso de las versiones paralelas, en este caso no ofrecen una mejora respecto al secuencial en el caso del tamaño de población 20, debido al pequeño tamaño de las islas resultantes. En cambio para el otro tamaño de población mayor (56 individuos), la versión paralelo si es capaz de producir soluciones más precisas que la versión secuencial con el mismo tamaño de población.

Comparando ambos algoritmos, 9.1 y 9.2, podemos ver que el GA obtiene las mejores soluciones en la mayoría de los contextos (1-0, 2-0, 3-0, 2-1 y 2-2), aunque muchas veces las diferencias entre las soluciones encontradas por el GA y el por CHC son escasas. Esto parece indicar que la exploración del espacio de búsqueda proporcionada por los operadores clásicos de cruce y mutación es suficiente para esta instancia. Otro detalle a destacar es que conforme se aumenta el tamaño de contexto y el algoritmo utiliza más de una etiquetas de alrededor de la palabra actual para decidir su etiqueta, la precisión obtenida por ambos algoritmos aumenta considerablemente.

La Tabla 9.3 presenta los resultados obtenidos al aplicar el SA a este problema. Para este texto de prueba, el SA ejecuta 5656 iteraciones usando una cadena de Markov de 800 y factor de decrecimiento de $\alpha = 0{,}99$. En la versión paralela, la fase de comunicación se realiza cada 100

Tabla 9.2: Precisión obtenida por el GA para el texto de prueba. Con PS nos referimos al tamaño de población.

Cont.	**GA-Int**				**GA-Bin(7)**				**GA-Bin(4)**			
	PS = 20		PS = 56		PS = 20		PS = 56		PS = 20		PS = 56	
	Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**
1-0	**93.30**	93.12	92.94	92.67	92.84	92.48	92.53	92.39	93.07	92.44	92.35	92.21
2-0	**93.97**	93.70	93.43	93.88	93.83	93.56	93.61	93.16	93.43	92.89	93.61	93.07
3-0	**94.47**	94.38	93.88	93.79	94.19	94.01	94.28	93.65	93.97	93.52	93.83	93.70
1-1	94.78	94.87	94.92	94.42	**95.14**	94.37	94.64	94.42	94.64	94.01	94.64	94.06
2-1	94.69	**95.19**	94.69	94.87	94.87	94.69	94.55	94.78	94.64	94.37	94.69	94.28
2-2	95.19	95.23	95.14	94.83	**95.54**	94.91	94.96	95.14	95.41	94.64	94.73	95.00

Tabla 9.3: Precisión obtenida por el SA para el texto de prueba.

	Contexto					
	1-0	2-0	3-0	1-1	2-1	2-2
Sec.	91.32	92.40	92.98	94.24	94.06	94.60
Par.	91.00	92.53	92.67	93.47	94.15	94.33

iteraciones. Como ocurría con los algoritmos anterior también se produce un incremento de la precisión conforme el tamaño del contexto es mayor, aunque la precisión del SA nunca alcanza a la de los algoritmos evolutivos anteriores. Esto es un claro indicativo de que la aproximación basada en población es más efectiva que la basada en trayectoria seguida por este algoritmo. Respecto a las versiones paralelas, no ofrecen una conclusión clara, ya que en la mitad de los contexto mejora la solución encontrada por la versión secuencial y en la otra mitad no, aunque en cualquier caso las diferencias son escasas.

La Tabla 9.4 presenta la media y la desviación estándar obtenida de los algoritmos evolutivos usando la codificación con la que mejores resultados obtiene (entera para el GA y binaria de 4 bits para el CHC), tanto para la versión secuencial como paralela. Podemos ver como la fluctuación de la precisión en las diferentes ejecuciones está siempre por debajo del 1 %, lo que indica que el comportamiento de los algoritmos es muy robusto.

Tabla 9.4: Media y desviación estándar para la mejor configuración de los algoritmo (PS = 20).

Contexto	**GA-Int**		**CHC-Bin(4)**	
	Sec.	Par.	Sec.	Par.
1-0	$93{,}07 \pm 0{,}24$	$92{,}81 \pm 0{,}23$	$91{,}79 \pm 0{,}16$	$91{,}96 \pm 0{,}30$
2-0	$93{,}68 \pm 0{,}23$	$93{,}37 \pm 0{,}18$	$92{,}91 \pm 0{,}28$	$93{,}31 \pm 0{,}16$
3-0	$94{,}15 \pm 0{,}32$	$94{,}07 \pm 0{,}20$	$93{,}66 \pm 0{,}16$	$93{,}76 \pm 0{,}18$
1-1	$94{,}54 \pm 0{,}13$	$94{,}41 \pm 0{,}23$	$94{,}61 \pm 0{,}31$	$94{,}49 \pm 0{,}24$
2-1	$94{,}31 \pm 0{,}26$	$94{,}27 \pm 0{,}25$	$94{,}23 \pm 0{,}21$	$94{,}44 \pm 0{,}17$
2-2	$94{,}91 \pm 0{,}26$	$94{,}72 \pm 0{,}35$	$94{,}76 \pm 0{,}15$	$94{,}82 \pm 0{,}32$

Otra característica de los resultados que es interesante mencionar es que la precisión obtenida está siempre en torno al 95 %, lo cual es un gran resultado [59] de acuerdo con el modelo estadístico usado. Se debe tener en cuenta que la precisión está limitada por los datos estadísticos proporcionados al algoritmo de búsqueda. Además, la meta del modelo es maximizar la probabilidad de cada contexto que aparece en la sentencia. Sin embargo en determinados textos, la etiqueta correcta no es siempre (aunque la mayoría de las veces sí es así) la más probable, haciendo que el etiquetador falle.

Tabla 9.5: Ratios del tiempo de ejecución de los algoritmos con respecto a la ejecución del GA secuencial con representación entera aplicado al contexto 1-0 (17.2805 s.).

Contexto	**GA-Int**		**GA-Bin(4)**		**CHC-Int**		**CHC-Bin(4)**	
	Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**
1-0	1	0.899	1.999	1.349	1.002	0.603	1.849	1.308
2-0	4.085	2.160	11.043	6.258	3.749	1.832	10.027	5.436
3-0	18.695	6.138	55.394	18.998	17.003	5.384	47.328	16.490
1-1	4.493	1.866	13.152	5.502	4.450	1.857	11.850	5.480
2-1	21.250	7.206	67.692	23.467	21.014	5.566	60.878	20.655
2-2	96.434	25.344	268.559	52.860	95.742	26.349	247.645	68.255

Finalmente, en la Tabla 9.5 mostramos el tiempo medio de ejecución para las codificaciones entera y binaria de 4 bits para el GA y el CHC. Se puede observar que el tiempo de ejecución se incrementa con el contexto, ya que para analizar cada palabra hay que analizar más datos, aunque como vimos antes, esto también conduce a una mejor calidad en las soluciones. También es destacable que el CHC es ligeramente más rápido que el GA (usando la misma codificación). Esto se puede explicar debido a que la falta de mutación del CHC compensa los cálculos adicionales que realiza (prevención de incesto o detección de la convergencia). También es interesante resaltar que las codificaciones binarias son más lentas que las enteras, ya que requieren cómputos adicionales (para pasarla a entera) antes de poder aplicar la función de evaluación a la solución tratada. Finalmente, observamos como el uso del paralelismo permite reducir el tiempo de cómputo total en todos los casos, aunque esta mejora es especialmente importante para tamaños de contexto grandes, ya que en contexto pequeños el grano de paralelismo es demasiado fino para ser aprovechado de forma adecuada.

9.3.2. Experimento 2: EAs vs. Viterbi

Ahora realizamos una segunda fase experimental donde basándonos en los resultados anteriores proponemos nuevas configuraciones de los algoritmos que serán aplicados a un conjunto de prueba más amplio que utiliza dos corpus. En concreto en estos experimentos utilizamos los corpus de Brown [195] y Susanne [228]. Además hemos comprobado la precisión de nuestros algoritmos, comparándolo con un algoritmo clásico dentro del campo, el algoritmo de Viterbi. La configuración de los algoritmos es la misma que la que se empleó en los experimentos anteriores.

De los experimentos previos, concluimos no había una codificación que fuese óptima para todos los métodos, sino que ésta dependía del método, por eso en estos experimentos, hemos mantenido ambas codificaciones, aunque se ha eliminado la codificación binaria con 7 bits, por que era la que peores resultados ofrecía siempre. En cuanto a los tamaños de la población ocurre algo similar, pero en este caso hemos aumentado ligeramente el tamaño, ya que los anteriores vimos que eran demasiado pequeños que las implementaciones paralelas tuviesen un comportamiento adecuado.

En primer lugar analizamos los resultados ofrecidos por el CHC de la Tabla 9.6. La primera conclusión que se obtiene es que la versión binaria permite obtener una mayor precisión con respecto a la codificación entera. Esto es consistente con los resultados anteriores en los que se observaba un comportamiento similar. El papel del paralelismo en este algoritmo es muy importante, ya que los mejores resultados son obtenidos por las versiones paralelas. Este comportamiento indica que el uso del paralelismo es muy beneficioso para este problema cuando se aborda con el CHC. En general, la precisión obtenida para el corpus de Brown es mejor que para el texto de Susanne,

Tabla 9.6: Precisión usando el CHC para ambos textos de prueba.

Contexto		**Entera**				**Binaria**			
		PS = 32		PS = 64		PS = 32		PS = 64	
		Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**
Brown	1-0	91.02	91.74	91.35	91.55	94.98	**95.32**	94.67	94.62
	2-0	91.31	91.31	91.83	92.18	**95.35**	**95.35**	95.18	95.03
Susanne	1-0	91.43	91.19	92.32	92.68	94.35	**95.01**	93.61	94.41
	2-0	93.42	93.75	93.53	93.56	94.37	**94.82**	93.96	94.31

Tabla 9.7: Precisión usando el GA para ambos textos de prueba.

Contexto		**Entera**				**Binaria**			
		PS = 32		PS = 64		PS = 32		PS = 64	
		Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**
Brown	1-0	95.83	**96.01**	94.34	94.95	93.15	93.02	92.97	93.02
	2-0	96.13	**96.41**	95.14	95.42	94.86	94.82	94.54	94.89
Susanne	1-0	96.41	**96.74**	95.46	96.25	95.12	95.32	94.96	94.54
	2-0	**97.32**	**97.32**	96.91	97.01	95.39	95.43	95.34	95.39

probablemente debido a que el conjunto de etiquetas del corpus de Susanne es muchoa más amplio que el de Brown, incluyendo etiquetas muy especificas que reducen la ambigüedad léxica de las palabras y limita los mecanismos de diversidad del CHC.

Ahora pasamos a examinar los resultados obtenidos por el GA y que son presentados en la Tabla 9.7. A diferencia del CHC, el GA obtiene los mejores resultados utilizando una representación entera. En lo que si coincide con el CHC, es en que el paralelismo permite obtener soluciones más precisas. De hecho en ambos algoritmos las mejores soluciones son alcanzadas cuando se utilizan versiones paralelas con una población de 32 individuos (aunque con diferente codificación dependiendo del algoritmo). También se puede observar como la precisión del algoritmo se incrementa cuando el contexto aumenta y por tanto el algoritmo dispone de más información a la hora de elegir las etiquetas. Esta tendencia también se observó antes en el CHC, pero no era tan conclusiva como lo es en el caso del GA. Otro detalle en el que varía el comportamiento del GA respecto al del CHC, es que en el GA los mejores resultados son obtenidos con el corpus de Susanne (en el caso del CHC, era con el corpus de Brown).

La Tabla 9.8 presenta los resultados de aplicar el SA a los dos textos de prueba. Confirmando lo que vimos en los experimentos de la sección anterior, observamos que el SA siempre proporciona peores resultados que los de los dos algoritmos evolutivos anteriores. La versión paralela del algoritmo ayuda a mejorar la calidad de las soluciones, pero pese a esa mejora, el comportamiento del algoritmo es bastante pobre, indicando que este mecanismo no es adecuado para la resolución de este problema.

En la Tabla 9.9 presentamos el mejor valor de fitness encontrado y la media de este valor para la configuración que proporciona mejor rendimiento en cada algoritmo, es decir, la versión paralela con codificación entera en el caso de GA y SA, y con codificación binaria para el CHC. También hemos includo los resultado del algoritmo Viterbi para realizar una comparativa entre nuestra aproximación y un método clásico de etiquetado. No indicamos la desviación estándar, ya que siempre está dentro del intervalo del 1 % y su inclusión no aporta demasiada información.

Inicialmente, estos resultados vuelven a confirmar lo estudiado en los experimentos anteriores y es que el GA permite realizar una exploración más eficiente del espacio de búsqueda que el CHC

Tabla 9.8: Precisión obtenida por SA para los dos textos de prueba.

Contexto		**Entera**		**Binaria**	
		Sec.	**Par.**	**Sec.**	**Par.**
Brown	1-0	91.41	**91.83**	91.25	91.58
	2-0	91.92	**92.28**	91.68	91.72
Susanne	1-00	91.03	**91.87**	89.79	90.53
	2-0	**92.31**	**92.31**	91.31	91.74

y el SA, permitiendo encontrar soluciones más precisas. El algorito Viterbi obtiene los mejores resultados en el caso del contexto bigram (1-0), aunque la diferencia con respecto al GA (que es el que mejores resultados ofrece de nuestros métodos) es bastante pequeña, por debajo del 1 %. En cambio, en el caso del modelo trigram (2-0), los resultados del GA superan a los de Viterbi, obteniendo una mejora de entre 2.5 % y 3 %. Estos resultados demuestran que algoritmos de búsqueda general como es el caso del algoritmo genético pueden ser competitivos con algoritmos específicos para el problema del etiquetado, siempre que se apliquen de forma adecuada.

Tabla 9.9: Comparación de todos los algoritmos para los dos textos.

Contexto		**GA-Int**		**CHC-Bin**		**SA-Int**		**Viterbi**
		Mejor	**Media**	**Mejor**	**Media**	**Mejor**	**Media**	**Mejor**
Brown	1-0	96.01	95.26	95.32	94.91	91.83	90.95	**96.85**
	2-0	**96.41**	96.23	95.35	94.80	92.28	92.14	93.88
Susanne	1-0	96.74	96.51	95.01	94.72	91.87	91.43	**98.59**
	2-0	**97.32**	96.84	94.82	94.38	92.31	91.45	93.96

Tabla 9.10: Tiempo de ejecución de los algoritmos (en segundos).

Contexto		**GA-Int**		**CHC-Bin**		**SA-Int**		**Viterbi**
		Sec.	**Par.**	**Sec.**	**Par.**	**Sec.**	**Par.**	**Sec.**
Brown	1-0	12.31	7.02	20.48	9.60	5.84	2.98	0.60
	2-0	47.93	21.19	60.92	26.44	17.32	7.93	39.29
Susanne	1-0	10.86	6.32	17.28	8.03	4.37	1.85	0.43
	2-0	75.12	24.31	123.94	58.31	32.12	10.42	92.11

Ahora analizaremos la Tabla 9.10, donde se muestran los tiempos medios para las mejores configuraciones del GA, CHC y SA, incluyendo también los tiempos del método Viterbi. Las conclusión sobre el tipo de codificación que obtuvimos en la sección anterior se sigue manteniendo en estos resultados, es decir, la codificación binaria (debido a los cómputos adicionales) es más lenta que la entera. De nuestros algoritmos, el SA es el más rápido. La razón de esto es que mientras este método trabaja sobre un única solución, los otros dos utilizan una población y además operadores más complejos (como el de cruce). Como esperábamos, las versiones paralelas permiten reducir de manera considerable el tiempo de ejecución (entre un 42 % y un 78 %), y esta reducción es mayor

confnforme el tamaño del contexto aumenta.

Comparando nuestros métodos con el algoritmo de Viterbi, observamos que esta técnica clásica para el etiquetado es bastante más rápida que nuestras propuestas en el modelo bigram. Sin embargo, cuando nos movemos a modelos más complejos, como es el trigram, esa ventaja se reduce e incluso se invierte, haciendo que nuestras propuestas sean más rápidas que Viterbi, lo que prueba la buenas características de escalados de estos métodos.

Tabla 9.11: Precisión y tiempo de ejecución (en segundos) del GA para ambos textos de pruebas usando dos nuevos contextos.

Contexto		Sec.			Par.		
		Mejor	**Media**	**Tiempo**	**Mejor**	**Media**	**Tiempo**
Brown	1-1	96.72	96.41	56.92	**96.78**	96.56	19.98
	2-1	**96.43**	96.22	210.36	**96.43**	96.27	67.28
Susanne	1-1	98.36	98.11	77.14	**98.59**	98.39	21.25
	2-1	97.78	97.31	283.49	**98.01**	97.64	78.17

Finalmente, hemos extendido este análisis y hemos estudiado nuestro mejor algoritmo (el GA con codificación entera) con dos nuevos contextos más complejos (1-1 y 2-1). La Tabla 9.11 muestra los resultados de estos resultados. No es aplicado el método de Viterbi debido a que este algoritmo está diseñado siguiendo un modelo de Markov, y por lo tanto sólo se puede aplicar con contextos que tengan en cuenta las etiquetas precedentes y no las que siguen a la actual. Si consideramos etiquetas de la derecha a la palabra considerada para etiqueta, no se sigue un proceso basado en modelo de Markov. Esto por si mismo es una razón importante para seguir con la investigación de metaheurísticas. Como vimos antes, el paralelismo permite mejorar la calidad de las soluciones y, al mismo tiempo, reducir el tiempo de cómputo de forma considerable (véase Tabla 9.10). En este caso, se observa como el incremente de la longitud del contexto (de 1-1 a 2-1) provoca una ejecución much más larga, pero sin producir ninguna mejora en la calidad de las soluciones. De hecho, la precisión usando el contexto 2-1 es peor que en el caso del contexto 1-1.

9.4. Conclusiones

En este capítulo hemos probado diferentes métodos de optimización para resolver una tarea muy importante dentro del lenguaje natural: la categarorización de cada palabra en un texto. Hemos probado diferentes método evolutivos como son un algoritmo genético y un CHC, y un método basado en trayectoria, el enfriamiento simulado. También hemos comparado esos resultados con un algoritmo clásico para el etiquetado, el algoritmo de Viterbi.

De los resultados obtenidos se pueden extraer diferentes conclusiones. La primera es que a codificación entera produce una mejor rendimiento de los algoritmos GA y SA, mientras que para el CHC es más adecuada la binaria. Se ha mostrado que el paralelismo es una herramienta muy útil tanto para mejorar la calidad de las soluciones como para reducir el tiempo de ejecución. También se ha observado que el GA obtiene mejores resultados que el CHC, indicando que la exploración del espacio de búsqueda llevada a cabo por los operadores tradicionales es suficiente para este problema. En cualquier caso, ambos métodos evolutivos tienen un rendimiento mejor que el SA. También, hemos observado que nuestra aproximación evolutiva es capaz de superar al método específico Viterbi en contextos grandes. Además nuestros mecanismos pueden resolver escenarios a los que no se pueden aplicar estos métodos de etiquetado que siguen modelos de Markov.

Capítulo 10

Resolución del Problema del Diseño de Circuitos Combinacionales

El diseño de circuitos combinacionales consiste en encontrar una fórmula booleana que produzca las soluciones requeridas dado un conjunto de entradas (tabla de verdad). Visto como problema de optimización, el diseño de circuitos tiene muchas características muy interesantes:

- Es un problema de optimización discreto en el cual las variables de decisión pueden ser o enteras o booleanas (como en este caso). Las soluciones producidas son expresiones booleanas que pueden ser descritas gráficamente.
- El tamaño del espacio de búsqueda crece muy rápidamente al incrementar el número de entradas y/o salidas del circuito.
- Puesto que se requiere que el circuito resultante produzca todas las salidas de la tabla de verdad correctamente, se puede considerar que este problema tiene un gran número de restricciones de igualdad muy estrictas.
- Muchos parámetros del problema se pueden modificar con el fin de producir diferentes grados de dificultad, pudiendo producir variantes más complejas que el problema original. Por ejemplo, se puede variar el tipo de puertas lógicas disponibles y el número de entradas que disponen cada tipo.

Hay muchos métodos basados en gráficos ampliamente utilizados por los diseñadores de este tipo de circuitos lógicos (por ejemplo, los mapas de Karnaugh [141, 258], y el método de Quine-McCluskey [172, 212]). A pesar de sus ventajas y sencillez, esos métodos no garantizan la obtención del circuito óptimo para una tabla de verdad arbitraria. Además, algunos de esos métodos como el de los mapas de Karnaugh tienen problemas de escalabilidad muy conocidos, y sólo pueden ser utilizados con circuitos con pocas entradas (normalmente no más de cinco o seis). Debido a esos inconvenientes de esos métodos y a su complejidad intrínseca, el diseño de circuitos se ha tratado con una gran variedad de heurísticos en los últimos años [66, 187]. A pesar de los buenos resultados ofrecidos para circuitos de tamaño pequeño y medio, los heurísticos también se ven fuertemente afectados por problemas de escalabilidad. Sin embargos, últimamente, se está siguiendo un diferente enfoque. La nueva meta consiste en optimizar circuitos pequeños/medianos tal que ese diseño

novedoso (ya que no requiere la intervención humana) sirvan como bloques constructivos para fabricar circuitos más complejos pero basado en estos circuitos muy optimizados. Este esquema se ha seguido ya previamente para el diseño de circuitos [66, 204, 186, 187].

En este capítulo presentamos un estudio comparativo entre un algoritmo genético tradicional, uno no tradicional como es el CHC, un enfriamiento simulado, y dos algoritmos híbridos. La razón de estos acercamientos es determinar si el diseño de circuitos combinacionales pueden beneficiarse de tales estrategias de búsqueda no incluidas en el algoritmo genético tradicional. Para este estudio se han considerado tanto las versiones secuenciales como las paralelas.

10.1. Definición del Problema

El problema consiste en diseñar un circuito que realice una función deseada (especificada a través de su tabla de verdad), dado un conjunto específico de puertas lógica. A pesar de esta definición el problema es tratado como un problema de optimización discreta.

En el diseño de circuitos, es posible usar diferentes criterios a la hora de crearlos. Por ejemplo, desde una perspectiva matemática, se podría querer minimizar el número total de literales, o el número total de operadores binarios, o el número de símbolos en una expresión.

La complejidad del circuito lógico está relacionada con el número de puertas en él. La complejidad de la puerta está a su vez relacionada con el número de entradas que tenga. Debido a que un circuito lógico es la implementación en hardware de una función booleana, reducir el número de literales en la función debería reducir el número de entradas en cada puerta y el número de puertas en el circuito y por lo tanto se reduciría la complejidad del circuito.

Z	**W**	**X**	**Y**	**F**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Tabla 10.1: Tabla de verdad del primer ejemplo.

Así, nuestra medida de la optimalidad de un circuito hemos considerado que es el número de puertas utilizadas (sean del tipo que sean). Obviamente, este criterio de minimización se puede aplicar a circuitos completamente funcionales (es decir, aquellos que realmente produzcan todas las salidas de la misma forma que venían indicadas en la tabla de verdad). Por lo tanto, primero definimos un circuito factible es un que produce *correctamente* todas las salidas como se indica en

la correspondiente tabla de verdad. Para ejemplificar esto, consideremos la Tabla 10.1. En este caso , tenemos como solución la siguiente expresión booleana: $\mathbf{F} = (WX + (Y \oplus W)) \oplus ((X + Y)' + Z)$. Así, para comprobar la factibilidad del circuitos, debemos reemplazar cada valor de las variable (**Z**, **W**, **X** e **Y**) por cada uno de los conjuntos de valores mostrados en la Tabla 10.1. Por ejemplo, en la fila 1, tenemos **Z**=0, **W**=0, **X**=0, **Y**=0. Reemplazando esos valores en **F** (tal como la definimos antes), obtenemos que **F**=1, que es precisamente lo indicado en la fila 1. Por lo tanto, nuestro circuito es correcto para la primera salida. Este mismo proceso se debe repetir para todas las filas. Si el circuito no da la salida correcta para todo ellos, consideramos que el circuito es *no factible* y no puede ser una solución válida a este problema.

Dos de los métodos más populares usados por los ingenieros electrónicos son los *Mapas de Karnaugh* [141], el cual se basa en una representación gráfica de las funciones booleanas, y el procedimiento de *Quine-McCluskey* [212, 172], el cual es un método tabular. Ambos métodos son automatizables por naturaleza de los mismos. Los mapas de Karnaugh son útiles para minimizar el número de literales hasta cinco o seis variables. El método de Quine-McCluskey es útil para funciones con cualquier número de variables y puede ser fácilmente programados para ser ejecutado en una computadora. En general, a partir de una tabla de verdad se pueden generar diferentes funciones booleanas con un mínimo número de literales basado en las elecciones hechas durante el mecanismo de minimización. Todas estas funciones con el mismo número de literales conducen a circuitos de complejidad similar entre ellos; por lo tanto, cualquiera de ellas puede ser seleccionadas para implementarse en hardware.

Se debe observar que el proceso de simplificación algebraico depende enteramente de la familiaridad personal con los postulados y teoremas y también de la habilidad personal para reconocer como aplicarlos. Por supuesto, esta habilidad varía de una persona a otra. Dependiendo de la secuencia en que los teoremas y los postulados sean aplicados, se puede obtener más de una simplificación de una expresión general. Normalmente todas estas simplificaciones son válidas y aceptables. Por lo tanto, normalmente no hay una única forma de minimizar una expresión booleana. Las soluciones que serán mostradas más tarde en la Sección 10.3 denominadas como diseñadores humanos, son en realidad la mejor solución (basada en nuestro criterio de evaluación) elegidas de un conjunto producidas por individuos que pueden ser considerados como "diseñadores expertos" de un circuito lógico. Sin embargo, esto no quiere decir que un humano no pueda mejorara algunas de esas soluciones que proporcionaremos, principalmente si consideramos que la solución óptima es desconocida para la mayoría de los problemas abordados.

Antes de continuar con la presentación de como hemos abordado nosotros este problema, pasaremos a describir brevemente los trabajos más significativos sobre este problema.

10.1.1. Trabajos Previos

A pesar de la gran cantidad de trabajo actual disponible en el uso de algoritmos genéticos, programación genética y estrategias evolutivas para diseñar circuitos lógicos combinacionales en los últimos años (véase por ejemplo [66, 162, 187]), ha habido pocos intentos de compararlos. Por eso una de nuestras principales motivaciones (obviamente a parte de producir soluciones eficientes) es realizar un estudio comparativo de diferentes tipos de metaheurísticas para averiguar que tipo de heurístico puede ser más adecuado para este problema que los algoritmos genéticos, ampliamente probados.

Los trabajos previos han demostrado,entre otras cosas, que este problema es altamente sensible a la codificación usada [10, 162, 236], y al grado de interconexión permitida entre las puertas [257]. Ha habido muchos estudios sobre la forma del espacio de búsqueda y su dificultad a la hora de ser explorada por algoritmos evolutivos [188, 256]. Sin embargo, este tipo de análisis han sido dirigidos a un único tipo de heurísticos (por ejemplo, los algoritmos genéticos [66], las estrategias evolutivas

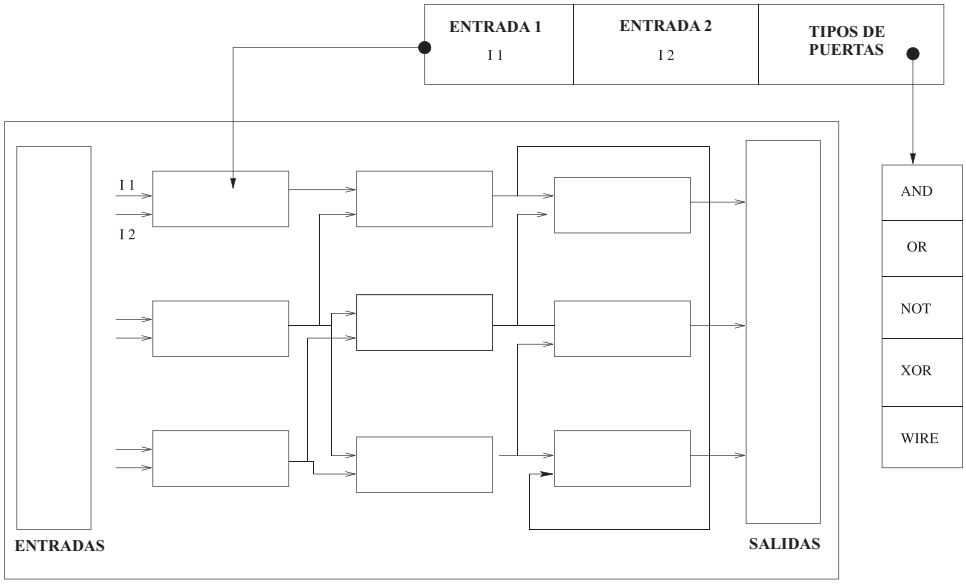


Figura 10.1: Matriz usada para representar un circuito. Cada puerta tiene sus entradas de cualquiera de las puertas de la columna previa. También se describe la codificación adoptada de cada elemento de la matriz.

[187], evolución simulada [10], colonias de hormigas [2, 67] o enjambre de partículas [68, 129]). Además, dado los problemas de escalabilidad en el diseño de circuitos usando algoritmos evolutivos, el uso del paralelismo puede ser un factor muy importante [122]. Pero sin embargo es remarcable que apenas existen estudios sobre este tema en la literatura del problema. Por lo tanto también es una importante contribución el uso de versiones paralelos de nuestros algoritmos para estudiar como el paralelismo afecta a la exploración del espacio de búsqueda.

10.2. Aproximación Algorítmica para su Resolución

A continuación describiremos la aproximación que hemos seguido a la hora de tratar este problema con los diferentes algoritmos. Para poder realizar una comparación justa, todos los heurísticos utilizados tendrán la misma representación y función de evaluación, que serán descritos en el siguiente apartado. Los detalles propios de cada algoritmo serán descritos a continuación.

10.2.1. Detalles Comunes

Representación

La representación en forma de matriz utilizada en este capítulo es similar a la presentada en la literatura en trabajos previos [65, 66] (véase la Figura 10.1).

Más formalmente, podemos decir que cualquier circuito puede ser representado como un array bidimensional de puertas $S_{i,j}$, donde j indica el *nivel* de la puerta, de forma que las puertas más cercanas a las entradas originales tienen un valor más bajo de j (El nivel se incrementa de izquierda a derecha en la Figura 10.1). Para un valor fijo de j, el índice i varía con respecto a las puertas que están "próximas" en el circuito, pero sin estar necesariamente conectadas. Cada elemento de

la matriz es una puerta (hay 5 tipos de puertas: AND, NOT, OR, XOR y WIRE[1]) que reciben dos entradas de cualquiera de las puertas de la columna previa como se muestra en la Figura 10.1.

En la parte superior de la Figura 10.1 también se indica la codificación utilizada para cada elemento de la matriz: una tripleta, donde los dos primeros campos se refieren a la entradas de la puerta y el tercero el tipo de puerta. Un individuo será un conjunto de tripletas de esta clase. Al tener cada valor de la tripleta un valor máximo, esta codificación puede ser representada como cadenas de bits de tamaño fijo, que es la representación real utilizada por nuestros métodos.

Esta representación en forma de matriz fue proporcionada originalmente por Louis [162, 163, 164]. El aplicó este acercamiento a el sumador de dos bits y al problema de comprobar la paridad de n bits (para $n = 4, 5, 6$). Esta representación también fue utilizada por Miller et al. [187, 189].

Es útil mencionar que el uso de este tipo de codificación es particularmente útil para el diseño de circuitos, ya que no permiten efectos negativos como el *bloat* (el crecimiento no controlado de los árboles asociados tradicionalmente a la programación genética [42]) [66, 187].

Función de Fitness

La siguiente fórmula es usada para calcular el fitness asociado a un individuo $\mathbf{x}$ para todos las técnicas utilizadas en este capítulo:

$$\text{fitness}(\mathbf{x}) = \begin{cases} \sum_{j=1}^{p} f_j(\mathbf{x}) & \text{si } f(\mathbf{x}) \text{ no es factible} \\ \sum_{j=1}^{p} f_j(\mathbf{x}) + w(\mathbf{x}) & \text{otro caso} \end{cases} \tag{10.1}$$

donde p es el número de entradas en la tabla de verdad (normalmente $p = 2^n$, siendo n el número de entradas en la tabla de verdad, pero a p se le puede también asignar un valor directamente, y que los casos que no aparecen en la tabla de verdad, no importa el valor ofrecido por el circuito), y el valor de $f_j(\mathbf{x})$ depende de la salida producida por el circuito codificado $\mathbf{x}$ (devuelve 1 si la salida es correcta para esa entrada y 0 en caso contrario). La función $w(\mathbf{x})$ devuelve un entero igual al número de puertas tipo WIRE presentes en el circuito $\mathbf{x}$. La solución producida son expresiones booleanas que están formadas por operaciones booleanas (AND, OR, NOT, XOR) y de variables binarias. Estas soluciones puede representarse de dos formas (que son equivalentes): (1) a través de su expresión booleana y (2) mostrando su representación gráfica.

En otras palabras, podemos decir que nuestra función de fitness trabaja a dos niveles [66]: primero, el maximiza el número de salidas que se calculan correctamente. Una vez que el circuito es factible se maximiza el número de WIREs en el circuito. Haciendo esto, se consigue optimizar el circuito para que tenga el menor número de puertas posibles. Resumiendo, nuestra meta es producir diseños completamente funcionales que maximice el número de cables (WIREs).

10.2.2. Detalles Específicos con GA

En este caso trabajamos con un algoritmo genético totalmente clásico que utiliza la representación binaria comentada anteriormente. Para completar nuestro diseño únicamente hay que definir que operadores utilizaremos en la fase de reproducción del algoritmo y describir la forma en la que ha sido paralelizado el algoritmo

En concreto utilizamos la recombinación uniforme (UX), que toma dos soluciones y crea dos soluciones. Cada uno de los alelos de cada nuevo descendiente se toma de forma aleatoria de uno de los padres. Para el cruce se utiliza la inversión de bits (*bit-flip*) que va cambiando el valor de cada valor a su complementario de forma probabilística.

[1]WIRE (cable) indica básicamente la no operación, o en otras palabras, la ausencia de puerta, y se utiliza para mantener la regularidad en la representación utilizada. En otro caso, tendríamos que utilizar cadenas de longitud variable.

A la hora de paralelizar el GA, hemos seguido un modelo distribuido (Sección 4.3.1). En concreto, tenemos un conjunto de islas ejecutando un GA secuencial en cada una, y cooperan entre sí, enviando una solución elegida por torneo binario cada 20 iteraciones a su vecino. La vecindad entre las islas se define por medio de una topología en anillo unidireccional.

10.2.3. Detalles Específicos con CHC

En este caso utilizamos el CHC tal como está descrito en la Sección 2.1.2, ya que la codificación en cadenas de bits es la misma que la que se ofrece en el trabajo original [89].

Para poder realizar una comparación justa con respecto a nuestro algoritmo de referencia, el GA, hemos paralelizado este algoritmo siguiendo exactamente el mismo esquema explicado en la sección anterior.

10.2.4. Detalles Específicos con SA

Para instancias este algoritmo con este problema en concreto se debe definir la estrategia de actualización utilizada y el mecanismo para explorar el vecindario.

Tras estudios preliminares, vimos que para este problema conviene utilizar un mecanismo de actualización de la temperatura que la disminuya con bastante rapidez, por lo que se decidió utilizar el esquema denominado FSA ($T_k = T_0/(1+k)$) que ya presentamos en el Capítulo 2. Para crear un vecino a partir del actual utilizamos el mismo mecanismo que utiliza el GA para mutar las soluciones, es decir, el operación de inversión de bits.

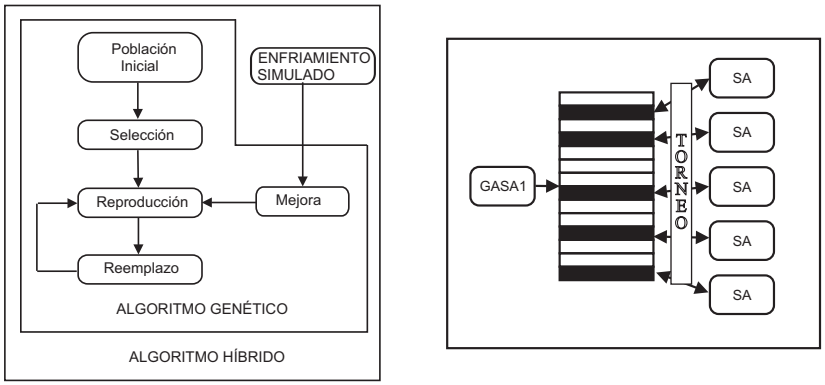
Para la paralelización del SA, hemos seguido el modelo de múltiples ejecuciones con cooperación (Capítulo 2). En concreto el SA paralelo (PSA) consta de múltiples SAs que se ejecutan de manera asíncrona. Cada uno de estos SAs empiezan de una solución aleatoria diferente, y cada cierto número de evaluaciones (*fase de cooperación*) intercambian la mejor solución encontrada hasta el momento. El intercambio de información se produce en una topología de anillo unidireccional. Para el proceso de aceptación de la nueva solución inmigrante, se aplica el mecanismo típico del SA, es decir, si es mejor, se acepta inmediatamente y si es peor, puede ser aceptada dependiendo de cierta distribución de probabilidad que depende del fitness y la temperatura actual.

10.2.5. Detalles de los Algoritmos Híbridos

Finalmente, hemos definido dos algoritmos híbridos. Como ya comentamos antes la hibridación es incluir conocimiento del problema en el algoritmo. Existe dos formas: la *hibridación fuerte* donde el conocimiento del problema se introduce mediante la utilización de representaciones o/y operadores específicos del problema y la *hibridación débil*, donde varios algoritmos son combinados para producir un nuevo esquema de búsqueda.

Nuestros dos algoritmos híbridos siguen este segundo esquema. El primero, al que denominamos GASA1, sigue el comportamiento básico de un GA, pero con la particularidad de que usa un SA como operador evolutivo. Es decir, en el bucle tradicional de un GA, tras aplicar los procesos de recombinación y mutación, se seleccionan varias soluciones (con baja probabilidad) que serán mejoradas usando el método SA. El esquema seguido por ese algoritmo se muestra en la Figura 10.2(a). La razón para esta clase de hibridación es que mientras que el GA localiza "buenas" regiones del espacio de búsqueda, el SA permite la explotación de las mejores regiones encontradas por el GA. Evidentemente, la motivación en este caso era ver si el nuevo heurístico híbrido toma lo mejor de ambas estrategias, produciendo una exploración del espacio de búsqueda más eficiente que cada una de ellas por separado.

En segundo esquema híbrido desarrollado, al que denominamos GASA2, ejecuta el GASA1 hasta que el algoritmo termine. Entonces este segundo híbrido selecciona (mediante torneo) algunas



(a) (b)

Figura 10.2: Modelos de Hibridización: (a) GASA1 y (b) GASA2.

soluciones de la población final y utiliza estas soluciones como soluciones iniciales de un SA. El principal motivo para el uso de esta aproximación es ver si el enfriamiento simulado puede usarse para mejorar las soluciones generadas por el otro algoritmo, y como consecuencia obtener soluciones más próximas al óptimo global. Este esquema está muestra en la Figura 10.2(b).

Ya que el esquema básico de los algoritmos híbridos es un algoritmo genético, al paralelizar hemos seguido el esquema utilizado en el algoritmo genético descrito anteriormente.

10.3. Análisis de los Resultados

Ahora pasamos a comentar los resultados obtenidos al aplicar nuestras propuestas algorítmicas al problema de diseño de circuitos lógicos combinacionales. Los resultados completos se han publicado en [64] y [25]. En concreto en [25] se muestran las tablas de verdad de todas las instancias y la representación gráfica de los circuitos. Primero presentaremos las instancias usadas y la configuración de los algoritmos aplicada. Entonces, analizamos el comportamiento de los algoritmos tanto respecto a su capacidad de producir buenas soluciones como su capacidad para reducir el tiempo de ejecución.

Los algoritmos en este trabajo han sido implementado bajo las especificaciones de MALLBA en C++ y han sido ejecutados en un clúster de Pentium 4 a 2.8 GHz con 512 MB de memoria y SuSE Linux 8.1 (kernel 2.4.19-4GB). La red de comunicación es una Fast-Ethernet a 100 Mbps.

10.3.1. Parámetros e Instancias

En la Tabla 10.2 resumimos las características de las instancias usada en nuestros experimentos.

Puesto que nuestra meta principal era analizar el comportamiento de los diferentes heurísticos y el impacto del paralelismo, no se hicieron pruebas exhaustivas para ajustar los parámetros, únicamente se realizaron pruebas aisladas para determinar a un alto nivel cual eran los operadores y rangos en los parámetros que mejores resultados ofrecían, y también basándonos en los resultados aportados por la literatura [65]. En todos esquemas englobados bajo la etiqueta de algoritmo evolutivos (es decir todos menos el SA), usan una población de 320 individuos para el primer ejemplo y de 600 para el resto. Todos los experimentos usan un ratio de cruce de un 60 % ($\rho_c = 0{,}6$) y de mutación el 50 % de la longitud del individuo. Cuando la población converge en el método

Tabla 10.2: Características de los circuitos. **matriz** = tamaño de matriz en filas × columnas, **tamaño** = longitud de la cadena binario, **BKS** = mejor solución reportada en la literatura.

nombre	**entradas**	**salidas**	**matriz**	**tamaño**	**BKS**
Sasao	4	1	5×5	225	34 [63]
Catherine	5	1	6×7	278	67 [205]
Katz 1	4	3	6×7	278	81 [63]
multiplicador de 2-bit	4	4	5×5	225	82 [63]
Katz 2	5	3	5×5	225	114 [166]

CHC, el algoritmo reinicia el 35 % de la población mediante la aplicación de una mutación uniforme ($\rho_m = 0{,}7$). En SA que utiliza como operador el GASA1, realiza 100 iteraciones en el primer y tercer ejemplo y 500 iteraciones para el resto. La probabilidad de aplicación de este operador es 0,01, es decir, este operador de mejora sólo se aplica a una de cada 100 soluciones. El SA que ejecuta el segundo híbrido (GASA2) tras la ejecución del GASA1 normal ejecuta 3000 iteraciones en la primera instancia y 10000 en el resto. De cada algoritmo se han realizado 20 ejecuciones independientes.

Los aspectos más relevantes que hemos medido en esta comparación son los siguientes: mejor fitness obtenido (**opt**), el número de veces que se encuentra ese valor entre todas las ejecuciones independientes (**hits**), el valor del fitness medio (**avg**) y el valor medio del número de evaluaciones (**#evals**).

10.3.2. Ejemplo 1: Circuito Sasao

Nuestro primer ejemplo tiene 4 entradas y una única salida. Los resultados para este ejemplo están resumidos en la Tabla 10.3. En este caso, los dos híbridos, GASA1 y GASA2, son capaces de converger a la mejor solución conocida para este circuito (la cual posee 7 puertas y tiene un fitness de 24) [63]. La expresión booleana asociada a este circuito es: $F = (WX+(Y \oplus W)) \oplus ((X+Y)'+Z)$. Se debe hacer notar que el número de evaluaciones requerido por GASA1 y GASA2 es el más alto de entre todos los algoritmos, aunque también hay que destacar que son los que obtienen mejores valores. También podemos observar como paralelizar los algoritmos híbridos permite incrementar la calidad de las soluciones, tanto en el número de veces que encuentra el óptimo como el valor medio obtenido de todas las ejecuciones. Sin embargo, el número de evaluaciones necesarias para encontrar el óptimo no disminuye en las versiones paralelas, como si que ocurre en las versiones paralelas del resto de algoritmos. Finalmente, debemos observar que el valor de fitness medio obtenido por la versión paralela del SA es ligeramente peor que en la versión secuencial, lo cual es un indicio que el comportamiento debido al esquema paralelo no es adecuado para esta instancia. Aunque se debe entender que esto es un caso aislado, ya que en el resto de los algoritmos si se obtiene un incremento en la calidad de las soluciones al ser ejecutadas en paralelo.

Otro aspecto que es útil analizar es el porcentaje de soluciones factibles que cada algoritmo mantiene a lo largo del proceso de búsqueda. Este porcentaje da una idea de cómo de difícil es para cada algoritmo alcanzar la región factible y mantenerse en ella. La Figura 10.3 muestra el porcentaje medio de soluciones factibles presentes en la población a lo largo del tiempo de búsqueda (es decir, generaciones) para todos los algoritmos utilizados. De esta figura se puede destacar el comportamiento del GASA2 paralelo que incrementa de forma muy rápido el número de soluciones factibles en la población, alcanzando el 100 % de soluciones factibles en menos de 100 iteraciones. En el lado contrario está el GA nunca alcanza el 100 % de soluciones factible (en cualquiera de

Tabla 10.3: Comparación de los resultados para el primer ejemplo.

Algoritmo	**secuencial**				**paralelo**			
	opt	**hits**	**avg**	**#evals**	**opt**	**hits**	**avg**	**#evals**
GA	31	10 %	15.8	96806	33	5 %	18.1	79107
CHC	27	5 %	15.1	107680	32	5 %	16.4	75804
SA	30	35 %	15.6	70724	31	5 %	15.2	69652
GASA1	34	10 %	23.2	145121	34	20 %	25.5	151327
GASA2	34	10 %	24.2	147381	34	30 %	27.8	155293

sus dos versiones). Los otros algoritmos si son capaces de alcanzar el 100 %, pero de una forma mucho más lenta que el GASA2. A partir de estos resultados podemos concluir que el GASA2 es el algoritmo que mejor rendimiento ha ofrecido en este algoritmo en sus dos versiones. GASA2 por un lado es el que mejores resultados ofrece (tanto el calidad de las soluciones como el número de éxitos) y también es el más rápido en alcanzar la región factible.

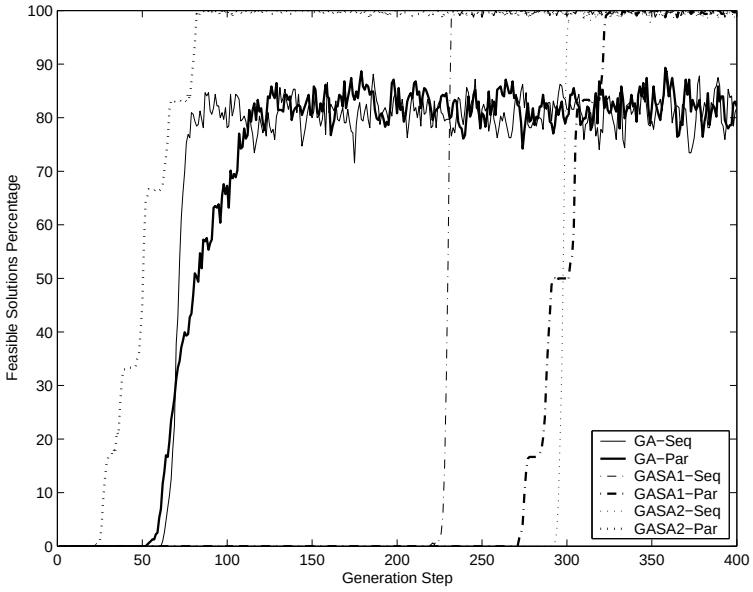


Figura 10.3: Porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos para el primer ejemplo.

Tabla 10.4: Comparación para el primer ejemplo entre nuestro mejor algoritmo (GASA2), el n-cardinality GA (**NGA**) [65], un diseñador humano (**HD 1**) que usa los mapas de Karnaugh, y **Sasao** [232].

GASA2	**NGA**	**HD 1**	**Sasao**
7 puertas	10 puertas	11 puertas	12 puertas

Para dar una idea de cómo de buena es la solución encontrada por GASA2, mostramos en

la Tabla 10.4 un comparativa entre esta solución y las ofrecidas por otros algoritmos existentes para este circuito. Esta segunda comparación es realizada sólo considerando la expresión booleana encontrada. Podemos ver como el circuito encontrado por nuestro algoritmo híbrido mejora al resto de los métodos existentes hasta el momento y que se habían aplicado a esta instancia.

10.3.3. Ejemplo 2: Circuito Catherine

Nuestro segundo ejemplo denominado Catherine tiene 5 entradas y una única salida. Los resultados obtenidos para este circuito se muestra en la Tabla 10.3.3. Otra vez, el algoritmo híbrido GASA2 es el que mejor solución encuentra, aunque en este caso, la versión paralela produce un resultado ligeramente mejor (columna **opt**) que su versión secuencial. La mejor solución encontrado en esta caso tiene la siguiente expresión booleana: $F = ((A_4)'(A_2A_0+A_1)(A_2+A_0))'((A_2A_0+A_1)(A_2+A_0)+A_3)$. De esa tabla se pueden extraer algunas conclusiones interesantes. En primer lugar, podemos observar que el fitness medio de los algoritmos híbridos se ve mejorado cuando se paraleliza el algoritmo. Más aún, además de mejorar la calidad de las soluciones también permite reducir el número medio de evaluaciones necesarias para encontrar la solución óptima en todos los algoritmos. A excepción del CHC, todos los algoritmos mejoran el fitness medio de las soluciones encontradas cuando son ejecutadas en paralelo respecto a su versión tradicional monoprocesadora. Otro detalle importante es que el SA supera al GA en relación a la mejor solución encontrada y con un número significativamente menor de evaluaciones, pero en cambio su fitness medio es peor que el del GA.

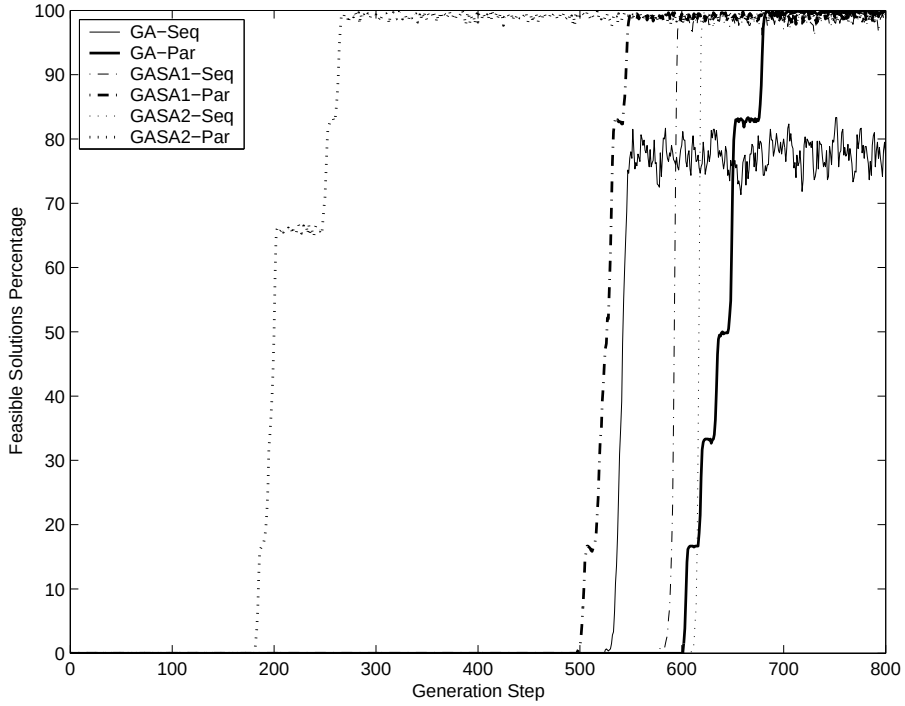


Figura 10.4: Porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos para el segundo ejemplo.

Al igual que hicimos en la instancia anterior, en este problema también mostramos la evoluación de la soluciones factibles a lo largo del proceso proceso de búsqueda (Figura 10.4). Como ocurría en

Tabla 10.5: Comparación de los resultados para el segundo ejemplo.

Algoritmo	**secuencial**				**paralelo**			
	opt	**hits**	**avg**	**#evals**	**opt**	**hits**	**avg**	**#evals**
GA	60	5 %	36.5	432170	62	10 %	41.0	345578
CHC	58	15 %	29.8	312482	61	5 %	28.9	246090
SA	61	5 %	33.1	175633	62	5 %	34.2	154064
GASA1	63	40 %	45.1	694897	65	5 %	50.6	593517
GASA2	64	10 %	47.3	720106	65	10 %	52.9	609485

el ejemplo anterior, el GASA2 paralelo es el que más rápido llega a obtener el 100 % de soluciones factibles (en menos de 300 iteraciones). El segundo que mejor rendimiento ofrece es la versión paralela del GASA1 y como ocurrió antes el único que no llega a alcanzar el 100 % es el algoritmo genético.

Tabla 10.6: Comparación para el segundo ejemplo entre nuestro mejor algoritmo (GASA2), el n-cardinality GA (**NGA**) [65] y un diseñador humano (**HD 1**).

GASA2	**NGA**	**HD 1**
9 puertas	10 puertas	12 puertas

Finalmente también en este ejemplo, comparamos nuestra mejor solución (obtenida por el GASA2) con otros algoritmos existentes en la literatura (Tabla 10.3.3). GASA2 mejora a las dos soluciones con las que se compara. También se debe mencionar que la solución obtenida por el GASA2, con 9 puertas, no es el mejor circuito posible, ya que en [205] reportan una solución 7 puertas ($F = (A + BC)(D \oplus E)(B + C) \oplus D)$) obtenido mediante el uso de un método que sigue el esquema de la programación genética, aunque también hay que aclarar que los resultados no son directamente comparables ya que no se usa exactamente criterio de optimización.

10.3.4. Ejemplo 3: Circuito Katz 1

Nuestro tercer ejemplo tiene 4 entradas y 3 salidas y los resultados de resolverlo se muestran en la Tabla 10.3.4. En este caso, también son los algoritmos híbridos los que son capaces de encontrar los mejores resultados reportados en la literatura para este circuito [63], con 9 puertas y un fitness de 81. Aunque hay que destacar que el GASA2 encuentra la solución en un número mayor de ocasiones (en la versión paralela). La mejor solución encontrada tiene la siguiente expresión booleana: $F_1 = ((D \oplus B) + (A \oplus C))'$, $F_2 = ((D \oplus B) + (A \oplus C))(C \oplus ((A \oplus C) + (A \oplus B))) \oplus ((D \oplus B) + (A \oplus C))$, $F_3 = (C \oplus ((A \oplus C) + (A \oplus B)))((D \oplus B) + (A \oplus C))$. En este caso, el uso del paralelismo produce un notable incremento del fitness medio de GASA1 y GASA2, aunque el número de veces que encuentra el óptimo es bastante escaso. Este incremento de la calidad de las soluciones por parte de los algoritmos híbridos tiene como punto negativo su alto coste computacional, casi el doble que el algoritmo genético tradicional. También se debe destacar que como ocurría en el ejemplo anterior, el comportamiento seguido los algoritmos paralelos exploran de manera más eficiente el espacio de búsqueda, lo que se traduce en un menor número de evaluaciones para encontrar su mejor solución que en general también es mejor que la que encontraban sus versiones secuenciales. El SA tiene un comportamiento ligeramente diferente al resto para esta instancia, siempre muestra un número muy bajo de evaluaciones, pero el coste de ese bajo esfuerzo computacional es que el número de veces que encuentra el óptimo es bajo.

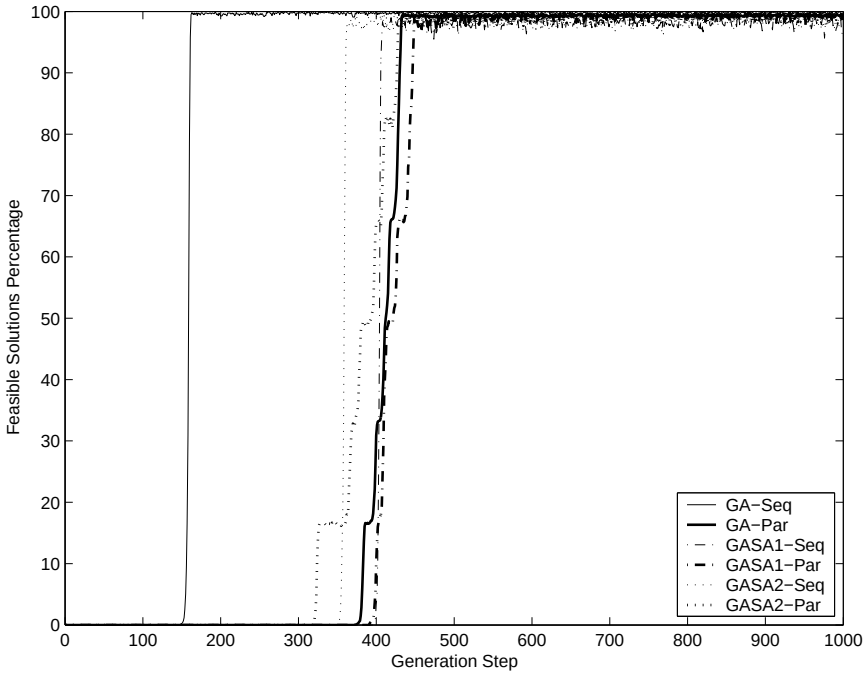


Figura 10.5: Porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos para el tercer ejemplo.

Tabla 10.7: Comparación de los resultados para el tercer ejemplo.

Algoritmo	**secuencial**				**paralelo**			
	opt	**hits**	**avg**	**#evals**	**opt**	**hits**	**avg**	**#evals**
GA	71	10 % 10 %	51.2	552486	76	15 %	54.5	498512
CHC	64	20 %	47.3	362745	70	5 %	49.3	252969
SA	67	15 %	46.3	194573	71	5 %	51.3	197315
GASA1	78	35 %	70.0	1090472	81	5 %	76.1	963482
GASA2	78	5 %	69.3	1143853	81	10 %	77.9	1009713

Ahora pasamos a ver el comportamiento de los algoritmos respecto a la factibilidad de las soluciones (Figura 10.5). Quizás el resultado más sorprendente sea el buen comportamiento del GA en esta instancia. Si en los ejemplos previos nunca alcanzaba el 100 % en este caso, es el más rápido en alcanzar dicho valor (en menos de 200 generaciones). Después del GA, observamos que el GASA2 también obtiene el 100 % bastante rápido. De hecho, podemos observar como en este caso, todos los algoritmos alcanzan la región factible para todas sus soluciones (100 %) antes de las 500 generaciones. Esta situación es indicativa que el espacio de búsqueda para este ejemplo, no es tan "accidentado" como el de los ejemplos previos. Si se debe decidir un ganador entre nuestros algoritmos, el GASA2 puede considerarse que tiene el mejor comportamiento de forma global, ya que es el que encuentra la mejor solución de forma más consistente.

La Tabla 10.3.4 muestra que el GASA2 supera claramente a todos los algoritmos existentes en la literatura para esta instancia.

Tabla 10.8: Comparación para el tercer ejemplo entre nuestro mejor algoritmo (GASA2), el n-cardinality GA (**NGA**) [65], un diseñador humano (**HD 1**) que usa los mapas de Karnaugh, y un segundo diseñador humano (**HD 2**) que usa el método de Quine-McCluskey.

GASA2	**NGA**	**HD 1**	**HD 2**
9 puertas	12 puertas	19 puertas	13 puertas

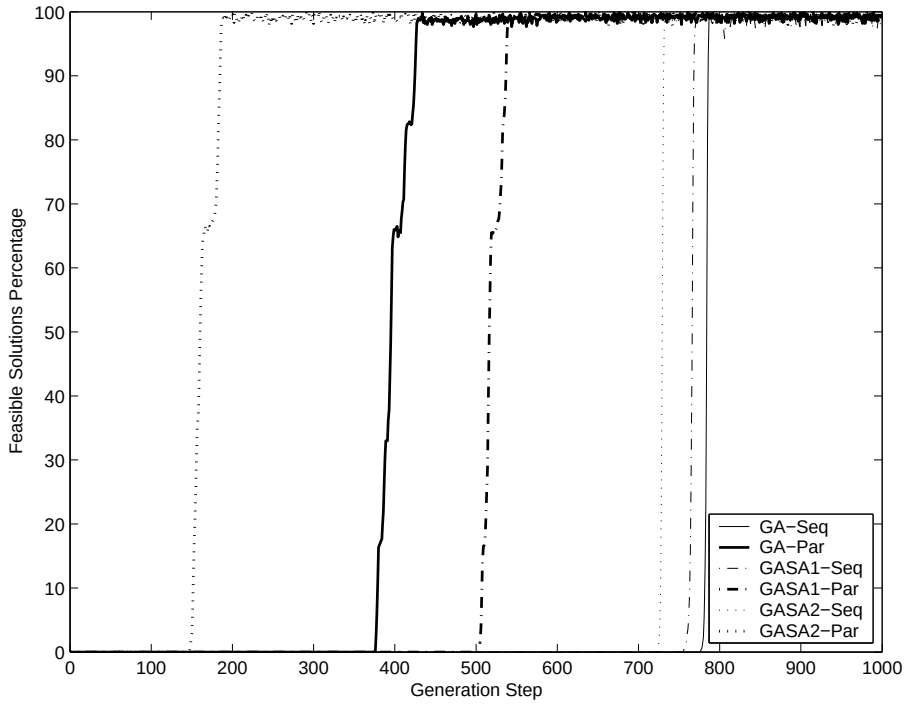


Figura 10.6: Porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos para el cuarto ejemplo.

10.3.5. Ejemplo 4: Circuito Multiplicador de 2 bits

La cuarta instancia que vamos a probar tiene 4 entradas y 4 salidas y se corresponde con un multiplicador de 2 bits. Los resultados obtenidos para este circuito están resumidos en la Tabla 10.3.5. En este caso (y como viene ocurriendo hasta ahora), el GASA2 es el método que mejores resultados obtiene y coincide con la mejor solución encontrada en la literatura que es un circuito de 7 puertas y un fitness de 82 [63]. Las expresiones booleanas asociadas a esta solución es: $C_3 = (B_0A_1)(B_1A_0)$, $C_2 = (A_1B_1) \oplus (B_0A_1)(B_1A_0)$, $C_1 = (B_0A_1) \oplus (B_1A_0)$, $C_0 = A_0B_0$. El uso del paralelismo sólo produce una ligera mejora en la calidad de las soluciones encontradas. Pero esa pequeña mejora, es lo que necesitaba el algoritmo GASA2 para pasar de una solución subóptima en su versión secuencial a encontrar la mejor solución conocida en la literatura en la versión paralela. También es destacable que el GA paralelo es capaz de encontrar una solución final que las que encuentra el GASA1, pero es un caso particular como demuestra que la media de fitness del GASA1 supere a la del GA. El GA también produce mejores resultados (columnas **opt** y **avg**) que los otros algoritmos "puros", SA y CHC, pero a costa de incrementar el número

de evaluaciones necesarias para encontrar esas soluciones.

Tabla 10.9: Comparación de los resultados para el cuarto ejemplo.

Algoritmo	**secuencial**				**paralelo**			
	opt	**hits**	**avg**	**#evals**	**opt**	**hits**	**avg**	**#evals**
GA	78	15 %	71.8	528390	81	5 %	76.3	425100
CHC	76	5 %	72.7	417930	80	10 %	74.2	246090
SA	77	5 %	68.6	268954	77	10 %	69.3	234562
GASA1	78	25 %	74.1	711675	80	20 %	76.9	852120
GASA2	80	10 %	75.4	817245	82	20 %	78.7	927845

La Figura 10.6 muestra el porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos. Otra vez más, el algoritmo GASA2 en su versión paralela es el más rápido tanto en alcanzar la región factible como el lograr que toda la población lo sea (hecho que alcanza antes de las 200 generaciones). El segundo mejor algoritmos en cuestión de factibilidad es la versión paralela del GA, el cual logra el 100 % de la población factible en menos de 400 generaciones. Sin embargo, su versión secuencial es el más lento en este aspecto. Como ocurría en el ejemplo anterior, todos los algoritmos llegan al tener toda la población factible, pero después no suelen ser capaces de explotar de forma adecuada la región factible para encontrar la solución óptima y sólo la versión paralela del GASA2 es capaz de encontrarla.

Finalmente en la Tabla 10.3.5 comparamos nuestra mejor solución con las obtenidas por otro métodos que previamente habían sido aplicados a esta instancia, donde se ve como la solución encontrada por el GASA2 es mejor que la del resto de métodos. También debe aclararse que Miller et al. [189] encuentran una solución con 7 puertas pero esa discrepancia con nuestra tabla es debido a la codificación que utilizan (permiten la puerta NAND, que según nuestra representación son dos puertas: AND + NOT). Además esa solución se encuentra en ejecuciones que tardan más de 3 millones de evaluaciones.

10.3.6. Ejemplo 5: Circuito Katz 2

Nuestro último ejemplo es el más complejo y posee 5 entradas y 3 salidas. Los resultados de los algoritmos para esta instancia se muestran en Tabla 10.3.6. Como ha venido ocurriendo en todos los ejemplos anteriores, el GASA2 (y en este caso también el GASA1) encuentran la mejor solución de la que tenemos constancia en la literatura para este circuito [166]. Esta solución tiene 7 puertas y un fitness final de 114 y la expresión booleana correspondientes e: $S_0 = E' + DC$, $S_1 = A' + BC$, $S_2 = C \oplus BC$. El comportamiento de los algoritmos en este ejemplo, también similar al de las instancias anteriores, es decir, el paralelismo ayuda a mejorar ligeramente las soluciones y el número de veces que se encuentra la mejor solución, siendo el GASA2 paralelo el método que mejor comportamiento global ofrece. El GA sigue la tendencia de los dos últimos

Tabla 10.10: Comparación para el cuarto ejemplo entre nuestro mejor algoritmo (GASA2), el n-cardinality GA (**NGA**) [65], un diseñador humano (**HD 1**), que usa los mapas de Karnaugh, un segundo diseñador humano (**HD 2**) que usa el método de Quine-McCluskey y **Miller et al.** [189].

GASA2	**NGA**	**HD 1**	**HD 2**	**Miller et al.**
7 puertas	9 puertas	8 puertas	12 puertas	9 puertas

ejemplos obteniendo unos resultados muy notables e incluso consigue encontrar la mejor solución conocida, aunque el número de veces que lo consigue es muy bajo (5 %).

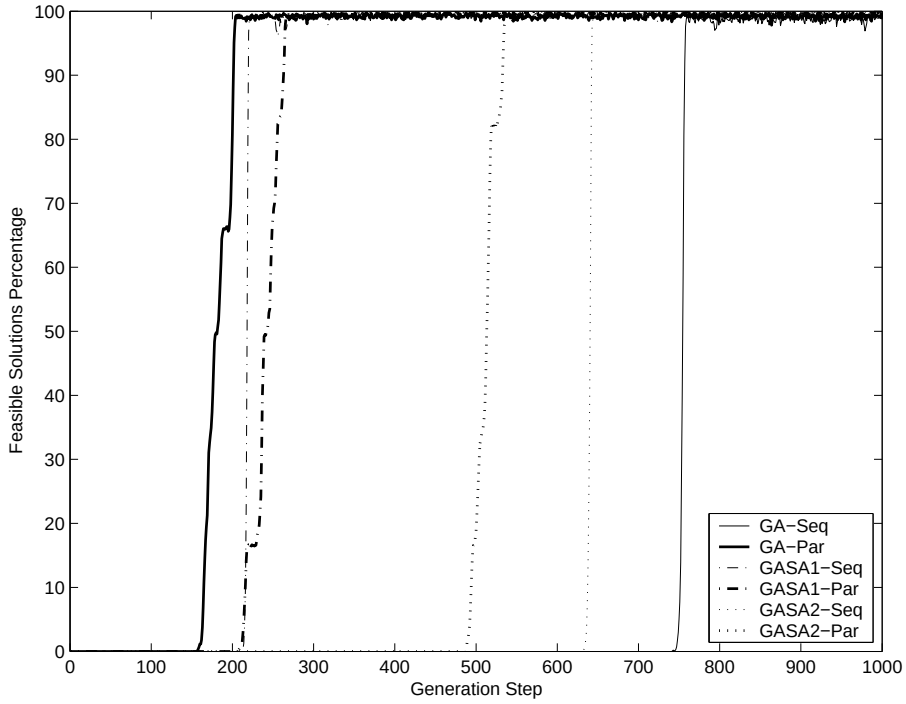


Figura 10.7: Porcentaje de soluciones factibles a lo largo de la ejecución de los algoritmos para el quinto ejemplo.

Tabla 10.11: Comparación de los resultados para el quinto ejemplo.

Algoritmo	**secuencial**				**paralelo**			
	opt	**hits**	**avg**	**#evals**	**opt**	**hits**	**avg**	**#evals**
GA	113	5 %	100.20	933120	114	5 %	102.55	825603
CHC	102	5 %	89.35	546240	104	10 %	90.76	540632
SA	111	10 %	94.85	280883	112	5 %	98.64	256234
GASA1	114	10 %	101.94	1013040	114	20 %	104.52	1010962
GASA2	114	20 %	106.75	1382540	114	35 %	106.90	1313568

Analizando el tema de la factibilidad de las soluciones (Figura 10.7), observamos que para este ejemplo los mejores algoritmos son el GA secuencial y las dos versiones del GASA1 que alcanzan el 100 % de factibilidad entre 200 y 300 generaciones. El resto de los algoritmos también logran alcanzar la zona factible pero mucho más lentamente.

Por último, comparamos nuestra solución con la de otros esquemas que habían sido aplicados para este quinto ejemplo (Tabla 10.3.6). En este caso se compara contra un algoritmo genético multiobjetivo [62] y un esquema basado en enjambre de partículas [166]. Todos ellos obtienen una solución equivalente con 7 puertas y además todos realizan un esfuerzo de cómputo muy similar (en torno a un millón de evaluaciones de la función de fitness).

Tabla 10.12: Comparación para el quinto ejemplo entre nuestro mejor algoritmo (GASA2), un GA multiobjetivo (**MGA**) [62], y un enjambre de partículas (**PSO**) [166].

GASA2	**MGA**	**PSO**
7 puertas	7 puertas	7 puertas

10.4. Conclusiones

Después de este estudio se pueden extraer una cuantas conclusiones genéricas. Primero es que la hibridación de un algoritmo genético con el enfriamiento simulado para ser beneficiosos para el diseño de circuitos combinacionales, al menos cuando se comparan respecto a GA y SA por separado. De los dos híbridos considerados, GASA2 es el que ofrece mejores resultados y de forma consistente, ya que es el que consigue la mejor calidad de las soluciones y de forma muy estable a lo largo de todas las instancias probadas.

Por otro lado, a pesar de que las características propias del CHC parecían adecuadas para el diseño de circuitos, nuestros resultados contradicen esa creencia y han mostrado que es el algoritmo que peores resultados ofrece de todos los heurísticos probados. Aparentemente, ni las restricciones de cruzamiento del CHC (*prevención de incesto*) ni su mecanismo de reinicio son suficiente para compensar la falta de diversidad que causa su selección elitista y el algoritmo tiene problemas para converger a la zona de las soluciones factibles en el espacio de búsqueda.

El SA también presenta pobres resultados cuando se compara con las aproximaciones híbridas o con el GA. Aunque, en las instancias más simples, el SA es capaz de obtener soluciones similares a las del GA (aunque su media de fitness sigue siendo peor). La razón de este mal comportamiento es que el algoritmo rápidamente alcanza un óptimo local del cual le cuesta salir. En contrapartida este algoritmo aunque no encuentra soluciones óptimas, encuentra "buenas" soluciones factibles en muy poco tiempo.

Finalmente, si analizamos el comportamiento de las versiones paralelos, hemos visto como en general consiguen mejorar la calidad de las soluciones encontradas y en muchos casos también permite reducir el número de soluciones del espacio que hay analizar para encontrar las mejores soluciones. Sin embargo, en muchos casos ese aumento en la calidad media de las soluciones viene acompañado con una ligera caída en número de veces que se encuentran esas soluciones. En otras palabras, sacrifica parte de su robustez a cambio de lograr soluciones de mayor calidad.

Capítulo 11

Resolución del Problema de Planificación y Asignación de Trabajadores

La toma de decisiones asociada a la planificación de trabajadores es un problema de optimización muy difícil ya que involucra muchos niveles de complejidad. El problema que nosotros tratamos consta de dos decisiones: selección y asignación. El primer paso consiste en seleccionar un pequeño conjunto de trabajadores del total del personal disponible, que generalmente es un conjunto mucho mayor. Una vez elegidos los trabajadores, se deben asignar para que realicen las tareas programadas, teniendo en cuenta las restricciones existentes de horas de trabajo, cualificación, etc. El objetivo perseguido es minimizar el coste asociado a los recursos humanos necesarios para completar todos los requisitos de las tareas a completar.

La complejidad del problema no permite el uso de métodos exactos para la resolución de instancias de tamaño realista. Por lo que nosotros proponemos varias metaheurísticas paralelas para su resolución. En concreto usamos un algoritmo genético paralelo y una búsqueda dispersa paralela.

En este capítulo inicialmente daremos una descripción formal del problema que queremos resolver (Sección 11.1). Luego, en la Sección 11.2 discutiremos como aplicar los algoritmos seleccionados para resolver este problema problema de planificación y asignación de trabajadores. La Sección 11.3 muestra los resultados computacionales obtenidos y finalmente en la última sección resumimos las principales conclusiones que se pueden extraer de estos experimentos.

11.1. Definición del Problema

A continuación mostramos una descripción formal del problema [117]. Tenemos un conjunto de tareas $J = \{1, \ldots, m\}$ que deben ser completados durante cierto periodo de tiempo (por ejemplo, una semana). Cada trabajo j requiere d_j horas de dicho periodo para ser completada. Por otro lado, existe un conjunto $I = \{1, \ldots, n\}$ de trabajadores disponibles para realizar esas tareas. La disponibilidad del trabajador i durante el periodo de planificación es s_i horas. Debido a razones de eficiencia cada trabajador debe estar asignado a una tarea un mínimo de h_{min} horas, ya que menos horas produciría una caída en el rendimiento del trabajo. Además, ningún trabajador puede estar asignado a más de j_{max} tareas. Los trabajadores tienen diferentes habilidades y no todos están cualificados para realizar todas las tareas. Consideraremos que A_i es el conjunto de trabajadores

que son capaces de realizar la tarea i. La planificación resultante no debe sobrepasar los t y con dichos trabajadores se debe poder llevar a cabo todas las tareas de forma completa. El objetivo del problema es conseguir seleccionar esos t trabajadores de entre los n disponibles, asignarlos a las tareas y todo ello minimizando alguna función objetivo prefijada (en general, será el coste asociado al pago de los trabajadores).

En concreto para formular este problema nosotros usamos el coste c_{ij} de asignar el trabajador i a la tarea j. A continuación mostramos el modelo matemático asociado al problema:

$$x_{ij} = \begin{cases} 1 & \text{si el trabajador } i \text{ es asignado a la tarea } j \\ 0 & \text{en otro caso} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{si el trabajador } i \text{ es seleccionado} \\ 0 & \text{en otro caso} \end{cases}$$

$$\begin{array}{rcl} z_{ij} & = & \text{número de horas que el trabajador } i \text{ está asignado a la tarea } j \\ Q_j & = & \text{conjunto de trabajadores asignados a la tarea } j \end{array}$$

$$\text{Minimizar} \sum_{i \in I} \sum_{j \in A_i} c_{ij} \cdot x_{ij} \tag{11.1}$$

Sujeto a

$$\sum_{j \in A_i} z_{ij} \leq s_i \cdot y_i \qquad \forall i \in I \tag{11.2}$$

$$\sum_{i \in Q_j} z_{ij} \geq d_j \qquad \forall j \in J \tag{11.3}$$

$$\sum_{j \in A_i} x_{ij} \leq j_{max} \cdot y_j \qquad \forall i \in I \tag{11.4}$$

$$h_{min} \cdot x_{ij} \leq z_{ij} \leq s_i \cdot x_{ij} \quad \forall i \in I, j \in A_i \tag{11.5}$$

$$\sum_{i \in I} y_i \leq t \tag{11.6}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I, j \in A_i$$

$$y_i \in \{0,1\} \quad \forall i \in I$$

$$z_{ij} \geq 0 \quad \forall i \in I, j \in A_i$$

En el modelo de arriba, la función objetivo (11.1) minimiza el coste total de la asignación. La restricción (11.2) limita el número de horas que está seleccionado cada trabajador. Si el trabajador no es elegido, entonces esta restricción no permite asignarle ninguna hora de trabajo. La restricción (11.3) valida que todas las tareas se realicen en su totalidad. La restricción (11.4) limita el número de número de tareas que puede realizar un trabajador elegido. La restricción (11.5) valida que una vez que un trabajador ha sido asignado a una tarea, él o ella debe estar asignado a esa tarea el mínimo número de hora especificado en los requisitos del problema. Además, esta misma restricción comprueba que ningún trabajador realice más horas de trabajo de las que puede. Finalmente, la restricción (11.6) limita el número de trabajadores que puede ser elegidos.

Este mismo modelo puede ser utilizado para optimizar otras funciones objetivo. Sea $\hat{c}_{ij}$ el coste por hora del trabajador i cuando realiza la tarea j. Entonces, la siguiente función objetivo minimiza el coste total de la asignación basado en las horas trabajadas (en la fórmula anterior era por trabajador seleccionado)

$$\text{Minimizar} \sum_{i \in I} \sum_{j \in A_i} \hat{c}_{ij} \cdot z_{ij} \tag{11.7}$$

Otra posible alternativa podría ser considerando que p_{ij} la preferencia del trabajador i para realizar la tarea j, y por lo tanto, el objetivo sería maximizar la preferencia total de la asignación:

$$\text{Maximizar} \sum_{i \in I} \sum_{j \in A_i} p_{ij} \cdot x_{ij} \tag{11.8}$$

También se pueden considerar otra variantes, pero nosotros a la hora de resolver este problema, asumimos que el que toma las decisiones quiere minimizar el coste total asociado a la asignación realizada calculada como se muestra en la ecuación 11.1.

Como se indica en [151], este problema está relacionado con diferentes problemas de localización de recursos con capacidad como es el problema de la p-mediana con capacidades [1, 147].

La dificultad de resolver este problema está caracterizado por la relación que exista entre h_{min} y d_j. En particular, las instancias de este problema para las cuales d_j es múltiplo de h_{min} (a las que nos referimos como "estructuradas") son más simples de manejar que aquellas en las que d_j y h_{min} no tienen ningún tipo de relación (a las que nos referiremos como "no estructuradas").

11.2. Aproximación Algorítmica para su Resolución

En esta sección se comentará las aproximaciones elegidas para tratar con este problema mediante diferentes algoritmos.

11.2.1. Algoritmo Genético

Como vimos en el Capítulo 2 los algoritmo genéticos son unos algoritmos basados en población y están clasificados como un subtipo de algoritmo evolutivo. A la hora de resolver este problema con un GA hemos tenido que desarrollar una representación específica, operadores que trabajen sobre ella y un método de mejora/reparación. A continuación describimos cada uno de esos aspectos.

Representación

Las soluciones son representadas como una matriz Z de $n \times m$ elementos, donde z_{ij} representa el número de horas que el trabajador i está asignado a la tarea j. En esta representación, se considera el trabajador i está asignado a la tarea j si $z_{ij} > 0$. Por lo tanto se puede establecer la siguiente relación entre los valores en Z y el modelo presentado en la sección anterior.

$$x_{ij} = \begin{cases} 1 & \text{si } z_{ij} > 0 \\ 0 & \text{en otro caso} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{si } \sum_{j \in A_i} z_{ij} > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Evaluación de las Soluciones

Las soluciones son evaluadas de acuerdo a la función objetivo presentada en la ecuación 11.1 más un término de penalización. Este termino adicional penaliza los incumplimientos de las restricciones (11.2), (11.3), (11.4) y (11.6). p_2, p_3, p_4 y p_6 son los coeficientes de penalización, que sirven para ponderar la importancia de cada uno de los incumplimientos de los diferentes tipos de restricciones.

Antes de calcular este valor de fitness, se le aplica a las soluciones un método de mejora/reparación, que disminuye el número de violaciones de restricciones cometidos por la solución y garantiza que la restricción (11.5) se satisface siempre. El operador también asegura que ningún trabajador realiza una tarea para la que no tiene cualificación.

Operador de Mejora/Reparación

El propósito de este operador es reparar las soluciones de tal manera que se vuelva factibles con respecto a las restricciones indicadas por el problema o que al menos las inconsistencias sean reducidas. Este operados realiza los cuatro pasos mostrados en la Figura 11.1.

1.	Eliminar las restricciones con respecto a h_{min}
2.	Eliminar las restricciones con respecto a la asignación de trabajadores no cualificados
3.	Asignar más trabajo a los trabajadores factibles
4.	Reducir incumplimiento de restricciones

Figura 11.1: Operador de Mejora/Reparación.

El primer paso de este operador repara las soluciones con respecto al mínimo número de horas que un trabajador debe estar asignado a una tarea. Matemáticamente, la operación realizada consiste en lo siguiente: si $0 < z_{ij} < h_{min}$ para todo $i \in I$ y $j \in A_i$, entonces $z_{ij} = h_{min}$.

El segundo paso corrige aquellas asignaciones de trabajadores a tareas para las cuales no tiene cualificación. A estas posiciones de la matriz Z se les pone el valor 0 indicando que el trabajador deja de estar asignado a esa tarea. Usando la notación matemática sería: si $z_{ij} > 0$ para todo $i \in I$ y $j \notin A_i$, entonces $z_{ij} = 0$.

El tercer paso considera que un trabajador es factible si él o ella satisfacen las restricciones (11.2) y (11.4). En este paso intenta ocupar el tiempo sobrante del trabajador. Consideramos como tiempo sobrante, el tiempo total que puede trabajar menos lo que tiene asignado (es decir, $s_i - \sum_{j \in A_i} z_{ij}$). Este tiempo es repartido a parte iguales entre las tareas que ya tiene asignadas. Esto permite la alta aprovechamiento de las horas de trabajo de los trabajadores ya asignados a tareas para así facilitar el cumplimiento de la satisfacción de la restricción (11.6).

El último paso comienzo con la ordenación parcial de los trabajadores de tal manera que menos factibles (con respecto a las restricciones (11.2) y (11.4)) tiendan a aparecer en las primeras posiciones del ordenamiento. Esta operación no es un orden total porque el operador para su correcto funcionamiento necesita un cierto grado de no determinismo en este paso. Una vez se ha establecido este orden, se aplica el proceso de reducir la infactibilidad de los trabajadores según ese orden.

Operador de Recombinación

Hemos diseñado un operador de cruce específico para que trate con la representación de este problema. El operador emplea un parámetro ρ_c que representa la probabilidad de que dos soluciones intercambien las asignaciones actuales del trabajador i. El proceso completo está descrito en la Figura 11.2.

Dadas dos soluciones $Z1$ y $Z2$, el operador de recombinación de la Figura 11.2 selecciona, con probabilidad ρ_c, al trabajador i. Si el trabajador es seleccionado, entonces las tareas asignadas en la solución $Z1$ se intercambiarán con las asignadas en $Z2$.

```
for (i = 1 to n) do
    if rand() < ρ_c then
        for (j = 1 to m) do
            z_ij^1 ↔ z_ij^2
        endfor
    endif
endfor
```

Figura 11.2: Operador de Recombinación.

Operador de Mutación

Además del operador de recombinación descrito antes, también hemos diseñado un operador específico de mutación para tratar con este problema. Este operador trabaja sobre una solución intercambiando las tareas asignadas a dos trabajadores. Este intercambio se produce con una probabilidad ρ_m, como se muestra en la Figura 11.3.

```
for (i = 1 to n) do
    for (j ∈ A_i) do
        k = trabajador aleatorio |k ≠ i and k ∈ Q_j
        if rand() < ρ_m then
            z_ij ↔ z_kj
        endif
    endfor
endfor
```

Figura 11.3: Operador de Mutación.

Dada una solución Z, el operador de mutación considera todos los trabajadores y tareas que el trabajador puede realizar de acuerdo con su cualifiación. Se elige otro trabajador aleatorio k y se intercambia la asignación de las tareas entre estos dos trabajadores.

Versión Paralela

A la hora de paralelizar el GA, hemos seguido un modelo distribuido (Sección 2.2.2). En este modelo un se ejecutan en paralelo pequeño número de GAs independientes, que periódicamente intercambian individuos para cooperar en la búsqueda global. Puesto que entre nuestros objetivos está comparar el comportamiento de este algoritmo con respecto a la versión secuencial, se ha configurado para que la población total del algoritmo paralela sea del mismo tamaño que la del algoritmo secuencial. Es decir, si el algoritmo secuencial usa una población de K individuos, cada una de las islas en el método paralelo usa *K/num_islas*.

Para terminar de caracterizar nuestra propuesta de algoritmo genético distribuido para este problema, debemos indicar como se lleva a cabo la migración. Nuestra implementación usa como topología un anillo unidireccional, donde cada GA recibe información del GA que le precede y la envía a su sucesor. La migración se produce cada 15 generaciones, enviando una única solución seleccionada por torneo binario. La solución que llega sustituye a la peor solución existente en la población si es mejor que ella.

11.2.2. Búsqueda Dispersa

Para resolver este problema, también hemos puesto un búsqueda dispersa (Capítulo 2) que al igual que los GAs están basados en trabajar sobre un conjunto de soluciones, pero que tiene ciertas peculiaridades que lo diferencian del resto de los algoritmos evolutivos.

La implementación concreta del SS para resolver este problema comparte ciertos aspectos con la implementación del GA, como son la representación (Sección 11.2.1), la función de fitness (Sección 11.2.1), el operador de mejora/reparación (Sección 11.2.1) y el operador de recombinación (Sección 11.2.1). El resto de las características específicas del SS se explican en los próximos párrafos

Población Inicial

En nuestra implementación, la población inicial está formada por varias soluciones generadas aleatoriamente que han sido modificadas mediante el proceso de mejora que se explica en la próxima sección.

Método de Mejora

Hemos diseñado un operador de mejora especifico para este problema. Este operador emplea un parámetro llamado ρ_i que representa que la probabilidad de que una solución peor que la actual sea seleccionada en cada paso del algoritmo de mejora. El proceso completo está resumido en la Figura 11.4.

for ($i = 1$ to $MaxIter$) **do**
  $Z' =$ generar vecino de Z
  if $fitness(Z') < fitness(Z)$ **or** $rand() < \rho_i$ **then**
    $Z = Z'$
  endif
endfor

Figura 11.4: Operador de Mejora.

Dada una solución Z, este operador genera un vecino mediante la aplicación del operador de mutación descrito en 11.2.1. Si esta nueva solución Z' es mejor que la original Z, entonces es aceptada y el proceso se repite durante $MaxIter$ iteraciones. Debido a que este esquema básico producía pobres resultados, también se permite que se acepten peores soluciones con una probabilidad definida por ρ_i. Este esquema es parecido al seguido al SA (aunque en el SA, esta probabilidad depende de los valores de fitness y de la temperatura). De hecho también se hicieron pruebas con SA, pero debido a que este proceso no puede ejecutar muchas iteraciones, el resultado ofrecido al usar como método de mejora el SA eran peores que esta propuesta que presentamos.

Versión Paralela

Como vimos en el Capítulo 2 se han propuesto varias alternativas para la paralelización de la SS. Nuestro objetivo es conseguir una mejora en la calidad de las soluciones (a parte del obvio ahorro en tiempo de cómputo). Por lo tanto, modelos como el maestro-esclavo no han sido considerados.

Finalmente nuestra implementación está basada en el modelo distribuido, donde tenemos varios SS secuenciales ejecutándose en paralelo y cada cierto número de iteraciones se intercambian una solución del RefSet. La topología de comunicación es la misma que la utilizada en el GA paralelo. Las soluciones emigrantes son elegidas por torneo binario, permitiendo así enviar tanto soluciones

de buena calidad (que saldrían del RefSet_1) como soluciones que permitan aumentar la diversidad (RefSet_2). Las soluciones emigrantes son introducidas en la población son introducidas en la población siguiendo el esquema general de actualización del conjunto de referencia.

Para que el coste computacional entre el SS paralelo y el secuencial sean similares, se ha decidido reducir el número de combinaciones consideradas por el método paralelo. En concreto, el número de combinaciones de soluciones analizadas en cada paso, es el mismo que el producido en la versión secuencial pero dividido entre el número de islas. En este caso, los subconjuntos de soluciones son generados de forma aleatoria, pero sin permitir que el mismo subconjunto sea seleccionado más de una vez.

11.3. Análisis de los Resultados

En esta sección vamos a analizar los resultados a la hora de resolver este problema. Aquí se muestra un resumen de los más interesantes, los resultados completos se pueden consultar en [17] y [27]. Primero presentaremos las instancias usadas. Entonces, analizamos el comportamiento de los algoritmos tanto respecto a su capacidad de producir buenas soluciones como su capacidad para reducir el tiempo de ejecución.

Los algoritmos en este trabajo han sido implementado bajo las especificaciones de MALLBA en C++ y han sido ejecutados en un clúster de Pentium 4 a 2.8 GHz con 512 MB de memoria y SuSE Linux 8.1 (kernel 2.4.19-4GB). La red de comunicación es una Fast-Ethernet a 100 Mbps.

11.3.1. Instancias

Para medir la calidad de nuestras propuestas, hemos generado varias instancias de este problema,. Dados los valores n, m y t, generamos las instancias de acuerdo con las siguientes características:

$$
\begin{array}{lcl}
s_i & = & U(50,70) \\
j_{max} & = & U(3,5) \\
h_{min} & = & U(10,15) \\
\text{Categoría(trabajador i)} & = & U(0,2) \\
P(i \in Q_j) & = & 0{,}25 \cdot (1 + \text{Categoría(trabajador i)}) \\
d_j & = & max\left(h_{min}, U\left(\frac{\bar{s}\cdot t}{2\cdot m}, \frac{1{,}5\cdot\bar{s}\cdot t}{m}\right)\right) \\
\text{donde } \bar{s} = \frac{\sum_i s_i}{n} \text{ y } \frac{\sum_j d_j}{\bar{s}\cdot t} \leq \alpha & & \\
c_{ij} & = & |A_i| + d_j + U(10,20)
\end{array}
$$

El generador establece una relación entre la flexibilidad de un trabajador y su coste correspondiente. Es decir, los trabajadores que tienen mayor categoría (y por tanto puede realizar más tarea) tiene un coste mayor. Hemos construido 20 instancias, 10 estructuradas (de s1 a s10) y 10 no estructuradas (de u1 a u10). Las diez instancias no estructuradas fueron generadas siguiendo los siguientes parámetros:

$$
\begin{array}{lcl}
n & = & 20 \\
m & = & 20 \\
t & = & 10 \\
\alpha & = & 0{,}97
\end{array}
$$

Se debe hacer notar que este generador de instancias, usa α como límite para la carga esperada para cada trabajador. Las instancias estructuradas fueron creadas usando esos mismos parámetros

pero h_{min} fue fijado a 4 y los valores d_j fueron ajustados siguiendo la siguiente fórmula:

$$d_j = d_j - mod(d_j, 4)$$

donde $mod(x, y)$ calcula el resto de x/y.

11.3.2. Resultados: Calidad de las Soluciones

En este apartado analizamos los resultados desde el punto de vista de la calidad de las soluciones obtenidas por el GA y el SS. En las tablas se resumen los valores medios de 30 ejecuciones independientes. Puesto que estamos trabajando con metaheurísticas estocásticas, hemos realizado una serie de análisis estadísticos para asegurar la consistencia de nuestras conclusiones. Como se comentó en el apartado 3.2.3, primero aplicamos el test de Kolmogorov-Smirnov para comprobar la normalidad de los datos. Si son normales aplicamos el análisis ANOVA, sino realizamos el el test de Kruskal-Wallis. De hecho, todos nuestros test en este capítulo son Kruskal-Wallis (con el 95 % de confianza) ya que los resultados obtenidos de aplicar Kolmogorov-Smirnov no puede asegurar la normalidad de los datos.

Por último, antes de analizar los resultados, queremos hacer notar al lector, que las versiones paralelas de GA y SS no sólo han sido ejecutados en paralelo, sino que también han sido ejecutado en un único procesador. La primera razón que motiva estos experimentos es que el modelo paralelo es independiente de la plataforma de cómputo. Como esperábamos, los correspondientes análisis estadísticos (incluidos en la columna KW_2 de las Tablas 11.1 y 11.2) indican que no hay diferencias significativas entre los resultados obtenidos por el modelo paralelo en las dos plataformas (símbolos "−"). Como consecuencia, a la hora de comparar la versión secuencial y las paralelas de cada algoritmo, sólo debemos considerar los resultados de las ejecuciones independientes y los test estadísticos solo involucran a esos tres conjuntos de datos (columna KW_3). En segundo lugar, al ejecutar el modelo paralelo en una única CPU permitiremos realizar de forma adecuada el análisis de tiempo de los algoritmos (véase la Sección 11.3.3 para más detalles). Los resultados de los mejores algoritmos para cada instancias están indicadas en **negrita**.

Resultados del GA

La primera conclusión que se puede extraer de la Tabla 11.1 es que cualquier configuración de los PGA son capaces de resolver las instancias de este problema con mayor precisión que el GA secuencial y con confianza estadística (véase los símbolos "+" de la columna KW_3). La instancia no estructurada u8 es la única excepción a esta aseveración ya que el test de Kruskal-Wallis dio resultados negativos, lo que indica que no es posible asegurar que haya diferencia estadística entre los algoritmos. Se puede desatacar también la gran precisión obtenida por los GAs paralelos en algunas instancias, como u1, u3 y u9, donde la reducción de coste respecto a las versiones secuenciales es mayor del 20 %.

Si comparamos las versiones paralelas entre sí, la Tabla 11.1 muestra que la versión paralela con 8 (PGA-8) islas obtiene la mejor solución en 13 de las 20 instancias, mientras que el la versión de 4 islas (PGA-4) sólo es capaz de encontrar la mejor planificación en 6 de las 20. Sin embargo, también debemos remarcar que las diferencias entre las soluciones de PGA-4 y de PGA-8 son bastante pequeñas, indicando que ambos tienen una habilidad similar para resolver este problema de planificación y asignación.

Resultados del SS

Ahora vamos a proceder a analizar los resultados del SS (Tabla 11.2) de la misma forma que lo hemos hecho para el GA en el apartado anterior. Los conclusiones que se pueden extraer son

Tabla 11.1: Resultados del GA para todas las instancias.

Prob.	GA Sec.	PGA-4			PGA-8			KW_3
		1 p.	4 p.	KW_2	1 p.	8 p.	KW_2	
s1	963	880	879	–	**873**	**873**	–	+
s2	994	943	940	–	**920**	922	–	+
s3	1156	**1013**	1015	–	1018	1016	–	+
s4	1201	1036	1029	–	1008	**1003**	–	+
s5	1098	1010	1012	–	**998**	1001	–	+
s6	1193	1068	1062	–	**1042**	1045	–	+
s7	1086	954	961	–	960	**953**	–	+
s8	1287	1095	1087	–	**1068**	1069	–	+
s9	1107	**951**	956	–	984	979	–	+
s10	1086	932	927	–	**924**	926	–	+
u1	1631	1386	1372	–	**1302**	1310	–	+
u2	1264	1132	**1128**	–	1153	1146	–	+
u3	1539	**1187**	1193	–	1254	1261	–	+
u4	1603	1341	1346	–	1298	**1286**	–	+
u5	1356	**1241**	1252	–	1254	1246	–	+
u6	1205	1207	1197	–	1123	**1116**	–	+
u7	1301	1176	1179	–	1127	**1121**	–	+
u8	**1106**	1154	1151	–	1123	1128	–	–
u9	1173	950	938	–	**933**	935	–	+
u10	1214	**1160**	1172	–	1167	1163	–	+

Tabla 11.2: Resultados del SS para todas las instancias.

Prob.	SS Sec.	PSS-4			PSS-8			KW_3
		1 p.	4 p.	KW_2	1 p.	8 p.	KW_2	
s1	939	896	901	–	**861**	862	–	+
s2	952	**904**	905	–	916	913	–	+
s3	1095	1021	1019	–	1005	**1001**	–	+
s4	1043	1002	**991**	–	997	994	–	+
s5	1099	**999**	1007	–	1009	1015	–	+
s6	1076	1031	1034	–	1023	**1022**	–	+
s7	987	956	942	–	941	**933**	–	+
s8	1293	1113	1120	–	**1058**	1062	–	+
s9	1086	**948**	950	–	952	950	–	+
s10	945	**886**	891	–	915	909	–	+
u1	1586	1363	1357	–	1286	**1280**	–	+
u2	1276	1156	1158	–	1083	**1078**	–	+
u3	1502	1279	1283	–	**1262**	1267	–	+
u4	1653	1363	1356	–	1307	**1305**	–	+
u5	1287	1176	1192	–	1175	**1169**	–	+
u6	1193	1168	1162	–	1141	**1136**	–	–
u7	1328	1152	1151	–	1084	**1076**	–	+
u8	1141	1047	1039	–	**1031**	1033	–	+
u9	1055	906	908	–	886	**883**	–	+
u10	1178	1003	998	–	**952**	958	–	+

similares a las obtenidas para el GA, es decir, los SS paralelos siempre obtienen mejores resultados que la versión secuencial para todas las instancias y con confianza estadística. En algunas instancias

esta mejora conseguida por los SS paralelos respecto a la versión secuencial es bastante importante, por ejemplo, en la instancia s8 pasamos de 1293 a 1048 (una reducción del 18 %), o en u8 donde se mejora de 1653 a 1305 (una reducción del 21 %). De media las ejecuciones paralelas consiguen una reducción en las planificaciones producidas de entre un 8.35 % para las instancias estructuradas y un 14.98 % para las no estructuras.

Si ahora comparamos las versiones paralelas entre sí, vemos que para las instancias estructuradas, cada uno de ellos obtienen la mejor solución en la mitad de las instancias, con lo que no es posible dar una conclusión clara. En cambio, en las instancias no estructuradas, la mejor solución siempre es encontrada por la versión paralela con 8 islas.

Comparativa entre GA y SS

Tras analizar los resultados obtenidos por el GA y el SS por separado, en esta sección queremos analizarlos en conjunto y compararlos entre sí. Ya que hay una gran cantidad de resultados, su análisis es bastante complejo y puede dificultar concluir algo con claridad, hemos resumido la información de las Tablas 11.1 y 11.2 en la Tabla 11.3 de la siguiente forma: hemos normalizada el coste de la planificación resultante para cada instancia con respecto a la peor (máximo) coste obtenido por cualquiera de los algoritmos propuesto, con lo que podemos comparar más fácilmente ya que eliminamos los problemas de escalados. La Tabla 11.3 son los valores medios para todas las instancias.

Una conclusión clara a la que se puede llegar es que todas las configuraciones del SS mejoran las soluciones de los GAs. Estos resultados nos permiten concluir que la búsqueda dispersa parece ser un acercamiento más prometedor para este problema que el esquema seguido por el algoritmo genético.

11.3.3. Resultados: Tiempos de Ejecución

Como vimos en el Capítulo 3, comparar los tiempos de ejecución de algoritmos paralelos no es un proceso simple. Para realizar una comparación justa, se debe comparar exactamente el mismo algoritmo, ya que comparar la versión paralela con la versión canónica secuencial puede llevarnos a conclusiones erróneas. Por este motivo, nosotros hemos ejecutado el mismo algoritmo paralelo en una plataforma paralela y en una plataforma secuencial. En la Tabla 11.4 se muestran los resultados medios de 30 ejecuciones independientes. Para estos valores hemos realizado los mismos test estadísticos que los que hicimos en las secciones anteriores.

Empezamos analizando los tiempos de aquellas configuraciones que han sido ejecutadas en una única CPU. Se puede observar que las versiones secuenciales son más rápidas que la ejecución monoprocesador de las versiones paralelas y con significancia estadística (símbolos "+" en la columna KW_6). Esta situación se obtiene para 17 de los 20 casos del GA y 15 de 20 en el SS.

Tabla 11.3: Resultados medios de todos los algoritmos e instancias.

Problemas		s1 - s10		u1 - u10	
Algoritmo		GA	SS	GA	SS
Secuencial		0.9994	0.9410	0.9896	0.9744
4 Islas	1 p.	0.8858	0.8743	0.8885	0.8605
	4 p.	0.8847	0.8747	0.8879	0.8598
8 Islas	1 p.	0.8783	0.8677	0.8735	0.8308
	8 p.	0.8776	0.8663	0.8718	0.8292

Tabla 11.4: Tiempo de ejecución (en segundos) para todas las instancias.

				4 Islas						8 Islas						KW_6
	Secuencial			1 CPU			4 CPUs			1 CPU			8 CPUs			
Prob.	GA	SS	KW_2	PGA-4	PSS-4	KW_2	PGA-4	PSS-4	KW_2	PGA-8	PSS-8	KW_2	PGA-8	PSS-8	KW_2	
s1	61	72	+	62	74	+	17	19	+	66	77	+	9	10	+	+
s2	32	49	+	32	53	+	9	14	+	37	58	+	6	8	+	+
s3	111	114	−	113	118	+	29	31	+	115	127	+	15	17	+	+
s4	87	86	−	93	84	+	24	23	−	95	87	+	13	13	−	+
s5	40	43	−	41	45	+	13	12	−	46	47	−	9	7	+	+
s6	110	121	+	109	122	+	34	33	−	114	128	+	18	18	−	+
s7	49	52	+	53	47	+	16	14	+	57	55	−	9	8	+	+
s8	42	46	−	45	48	−	13	13	−	48	50	−	7	7	−	+
s9	67	70	+	73	71	−	21	19	+	76	74	−	13	10	+	+
s10	102	105	+	105	101	+	28	28	−	109	106	+	16	15	−	+
u1	95	102	+	98	108	+	29	29	−	102	111	+	16	16	−	+
u2	87	94	+	89	95	+	28	26	+	92	99	+	15	14	−	+
u3	51	58	+	55	55	−	17	17	−	59	59	−	10	11	+	+
u4	79	83	+	79	86	+	26	24	+	86	92	+	15	15	−	+
u5	57	62	+	62	62	−	21	18	+	63	68	+	12	10	+	+
u6	75	111	+	72	115	+	20	30	+	70	119	+	13	16	+	+
u7	79	80	−	81	81	−	24	24	−	89	83	+	15	14	−	+
u8	89	123	+	88	118	+	23	35	+	92	123	+	14	20	+	+
u9	72	75	−	78	77	−	22	22	−	85	80	+	13	12	−	+
u10	95	99	+	96	96	−	25	28	−	99	101	+	13	17	+	+

Este hecho es debido a que al ejecutar el proceso paralelo en una única máquina se añade una sobrecarga debido al manejo de procesos (cambio de contexto, sincronización, etc.). Sin embargo hay casos donde las versiones paralelas aún siendo ejecutadas en un único procesador, consiguen reducir el tiempo de ejecución (como pueden ser las instancias s6 y u8 para GAs y s4, s7, s10, u3, y u8 para SS). Esta situación es debido a que la sobrecarga de ejecutar el algoritmo paralelo en un sistema monoprocesador, el modelo de búsqueda del algoritmo permite encontrar la solución más fácilmente. De todas formas se observa que la sobrecarga es en general un factor más importante ya que las ejecuciones secuenciales a excepción de los pocos casos comentados, computan más rápido.

Analizando los tiempos absolutos, podemos observar que los GAs son generalmente más lentos que los SS cuando la plataforma de cómputo está formada por una única CPU. Sin embargo, esas diferencias se ven disminuidas e incluso se invierten cuando nos movemos a las plataformas paralelas (véase las columnas “4 CPUs” y “8 CPUs” de la Tabla 11.4). En general, esos tiempos son muy similares y no hay diferencias significativas en muchos casos (véase símbolos “-” en la columna KW−2).

Para enriquecer nuestro entendimiento de los efectos del paralelismo en nuestros algoritmos hemos calculado dos métricas: la eficiencia paralela (η) y la fracción serie (sf) (véase una descripción de estas métricas en el Capítulo 3). En la Tabla 11.5 incluimos los valores resultantes de estas medidas. Los valores de la eficiencia paralela (entorno al 0.8 y 0.9) indican que todas las versiones paralelas del GA y del SS rinden muy bien en la plataforma de cómputo paralela.

Si comparamos la eficiencia de los algoritmos cuando se incrementa el número de procesadores, podemos ver una ligera reducción en la eficiencia. Es en este punto cuando la fracción serie tiene un papel importante. Si los valores de esta métrica permanecen constantes indican que esa pérdida de eficiencia es debida al limitado paralelismo del modelo en sí mismo. Un claro ejemplo de este hecho es la instancia s5 con PGAs: cuando incrementamos el número de procesadores la eficiencia se reduce en casi un 15 % (de 0.78 en PGA-4 a 0.63 en PGA-8) pero la fracción serie es casi constante (0.089 en PGA-4 y 0.080 en PGA-8).

Tabla 11.5: Eficiencia paralela y fracción serie para todas las instancias.

	PGA-4		PSS-4		PGA-8		PSS-8	
Problem	η	sf	η	sf	η	sf	η	sf
s1	0.91	0.032	0.97	0.009	0.91	0.012	0.96	0.005
s2	0.88	0.041	0.94	0.018	0.77	0.042	0.90	0.014
s3	0.97	0.008	0.95	0.016	0.95	0.006	0.93	0.010
s4	0.96	0.010	0.91	0.031	0.91	0.013	0.83	0.027
s5	0.78	0.089	0.93	0.022	0.63	0.080	0.83	0.027
s6	0.80	0.082	0.92	0.027	0.79	0.037	0.88	0.017
s7	0.82	0.069	0.83	0.063	0.79	0.037	0.85	0.023
s8	0.86	0.051	0.92	0.027	0.85	0.023	0.89	0.017
s9	0.86	0.050	0.93	0.023	0.73	0.052	0.92	0.011
s10	0.93	0.022	0.90	0.036	0.85	0.024	0.88	0.018
u1	0.84	0.061	0.93	0.024	0.79	0.036	0.86	0.021
u2	0.79	0.086	0.91	0.031	0.76	0.043	0.88	0.018
u3	0.80	0.078	0.80	0.078	0.73	0.050	0.67	0.070
u4	0.75	0.105	0.89	0.038	0.71	0.056	0.76	0.043
u5	0.73	0.118	0.86	0.053	0.65	0.074	0.85	0.025
u6	0.90	0.037	0.95	0.014	0.67	0.069	0.92	0.010
u7	0.84	0.061	0.84	0.061	0.74	0.049	0.74	0.049
u8	0.95	0.015	0.84	0.062	0.82	0.031	0.76	0.042
u9	0.88	0.042	0.87	0.047	0.81	0.031	0.83	0.028
u10	0.96	0.013	0.85	0.055	0.95	0.007	0.74	0.049

11.4. Conclusiones

En este capítulo hemos tratado con el problema de planificación y asignación de trabajadores. Para resolver este problema hemos propuesto dos metaheurísticas paralelas: un algoritmo genético paralelo y una búsqueda dispersa paralela. Hemos desarrollado estas versiones paralelas para poder tratar con instancias de este problema que sean de tamaño realista.

Las conclusiones de este trabajo se pueden resumir atendiendo a diferentes criterios. Primero, como era esperado, las versiones paralelas de los métodos han permitido conseguir una importante reducción del tiempo de ejecución con respecto a las versiones secuenciales. De hecho, nuestras implementaciones paralelas obtienen un speedup muy bueno (cerca de lineal). En varias instancias, se ha notado una pequeña perdida de eficiencia cuando se incrementa el número de procesadores de cuatro a ocho. Pero esta pérdida de eficiencia es debido principalmente debido al límite de paralelismo del programa, como demostró que la variación en la fracción serie fuese muy pequeña.

En segundo lugar, hemos observado que el paralelismo no solo permitió reducir el tiempo de ejecución sino que también ha posibilitado el incremento en la calidad de las soluciones encontradas. Incluso cuando ejecutamos los algoritmos paralelos en un único procesador, sus resultados mejoran a los de la versión secuencial, indicando claramente que los métodos secuencial y paralelos son diferentes algoritmos con diferentes comportamientos.

Finalmente, debemos destacar que los resultados del SS son mejores que los del GA para ambos tipos de instancias (estructuradas y no estructuradas). Esto parece indicar que el esquema de búsqueda de SS es más apropiado para este problema que el seguido por el GA. Una hipótesis en la que estamos trabajando es que quizás el operador de mejora usado por el SS es beneficioso para este problema y posiblemente un algoritmo híbrido que incorporara al GA un mecanismo de búsqueda local podría mejorar los resultados obtenidos. Esa mejora es necesaria para poder abordar instancias mucho mayores a las que hemos tratado hasta ahora (por ejemplo con $n \approx$ y $m \approx 200$).

Parte IV

Conclusiones

Conclusiones

En esta tesis hemos desarrollado un estudio genérico sobre técnicas metaheurísticas paralelas para su aplicación para resolver problemas complejos de interés real. Comenzaremos este capítulo con un resumen de las conclusiones alcanzadas en esta tesis siguiendo un nivel de abstracción por encima del meramente numérico.

Inicialmente propusimos un estudio unificado de todas las metaheurísticas, en especial de sus variantes paralelas, ofreciendo un modelo formal de ellas. Este modelo formal nos ha permitido caracterizar las metaheurísticas, observando los puntos en común que poseen y los diferenciadores que provocan las diferentes dinámicas de búsqueda resultantes, siempre centrándonos principalmente en modelos paralelos. Nuestro modelo es lo suficientemente genérico para abordar los modelos existentes y nuevos modelos, pero a la vez permite controlar detalles de bajo nivel, que son especialmente interesantes en las metaheurísticas paralelas.

Una vez estudiados esos modelos, nos dedicamos a un paso intermedio pero muy importante dentro de la investigación que en general está muy ignorado y desatendido por los investigadores: el diseño e implementación de metaheurísticas y el diseño experimental y análisis de resultados riguroso. Para ello, creamos dos guías metodológicas de lo pasos más importantes que se deben dar en estos aspectos. Vimos que el diseño software realizado para la implementación es un aspecto muy importante a la hora de obtener un código eficiente y reutilizable y que la aplicación de los estándares y metodologías de la Ingeniería del Software es un aspecto clave en el desarrollo de un software para metaheurísticas paralelas de calidad. En concreto, en esta tesis se desarrolló parte de la biblioteca Mallba, que es una biblioteca de esqueletos de código que permiten la utilización simple de múltiples metaheurísticas, tanto para plataformas secuenciales como paralelas.

Como hemos dicho, también estudiamos cómo realizar el diseño experimental y el análisis de los resultados de forma correcta. Para ello, definimos de manera formal las diferentes métricas que pueden ser utilizadas para medir el rendimiento de las metaheurísticas paralelas, cuándo se pueden utilizar cada una de ellas y cómo se deben interpretar sus resultados. Vimos que esto es un aspecto muy delicado, ya que la incorrecta utilización de estas métricas pueden llevar a conclusiones erróneas. También vimos que la utilización de test estadísticos para asegurar la significancia de nuestras conclusiones es un tema cada vez más importante y solicitado dentro de la comunidad científica.

Después pasamos a estudiar el comportamiento de las metaheurísticas paralelas, para crear un cuerpo de conocimiento que sirva a los investigadores para poder utilizarlos de forma eficiente cuando se quiere resolver un problema y así como para el desarrollo de nuevos métodos más eficientes y eficaces. En estos estudios vemos la importancia de la parametrización utilizada. Por ejemplo vemos que si el sistema no es homogéneo en cuanto a la comunicaciones la topología es un elemento clave en el rendimiento de la metaheurística paralela. Esto ocurre muy frecuentemente cuando se están usando sistemas que tiene enlaces rápidos conviviendo con el enlaces muchos más lentos (por ejemplo, a través de Internet). Observamos que si se limitaba el uso de los enlaces más lentos, el rendimiento del algoritmo completo mejoraba. Además el modelo resultante, donde tenemos diferentes grupos de procesadores que cooperan internamente mucho y poco entre los distintos grupos, produce un equilibro de intensificación/diversificación muy bueno y adecuado para problemas complejos, ofreciendo mejores resultados que ambientes homogéneos. También observamos como una comunicación asíncrona es muy beneficiosa para obtener un buen rendimiento en la ejecución paralela, en especial cuando los enlaces pueden ser lentos (sistemas WAN y Grid). En relación con este punto, también observamos que el ratio comunicación/cómputo es un aspecto muy importante. Por ejemplo, en nuestros experimentos observamos que un ajuste preciso de este parámetro nos permitía elevar el rendimiento casi un 50 %.

También hemos hecho un estudio más teórico sobre la dinámica de los modelos paralelos dis-

tribuidos, ofreciendo un modelo matemático bastante exacto de su comportamiento y que nos permite estudiar el efecto de los diferentes parámetros que rigen el modelo. En concreto, hemos visto que los parámetros que definen la topología y la frecuencia del intercambio de información son los que más condicionan la dinámica del problema y que un buen ajuste de estos parámetros es vital para un rendimiento óptimo del método. Una extensión muy interesante es estudiar cómo incorporar esta información dentro del propio algoritmo para que tome decisiones de acuerdo a ella y permita un comportamiento más eficiente y eficaz.

Finalmente aplicamos todo este conocimiento extraído de los análisis previos y de otros experimentos para diseñar metaheurísticas paralelas que aborden problemas complejos de interés real. Hemos visto cómo nuestras metaheurísticas paralelas han posibilitado tratar con instancias muy complejas de problemas muy variados que poseen diferentes características como una espacio de búsqueda muy grande (etiquetado léxico y ensamblado de fragmentos de ADN) o un número muy grande de restricciones muy fuertes (planificación de trabajadores y diseño de circuitos).

Hemos visto como las versiones paralelas no sólo nos han permitido reducir de forma muy importante el tiempo de ejecución, sino que también en mucho casos han permitido producir una mejora en la calidad de las soluciones significativa. Aunque también hemos observado que en algún caso muy particular, el paralelismo ha producido una pequeña pérdida en la calidad de las soluciones encontradas, aunque en general ocurre lo contrario y nos ha permitido obtener nuevas mejores resultados para varias instancias de los problemas abordados.

El esquema general de metaheurísticas paralelas ha permitido abordar instancias de problemas que los métodos clásicos no podían, como por ejemplo considerar el contexto derecho a la hora de hacer el etiquetado de un texto, o la resolución de una instancia de 77k en el ensamblado de ADN (cuando en general, las instancias tratadas en la literatura suelen constar de 30-50k bases). También se ha observado que para determinados problemas un esquema genérico no permite al algoritmo resolver de forma adecuada el problema. Esto se ha solucionado incorporando conocimiento del problema al algoritmo mediante su hibridación, lo que ha permitido obtener en muchos casos, métodos más eficientes y precisos que los existentes en la literatura. Esta incorporación de información se puede realizar de diversas formas, por ejemplo, en el problema de diseño de circuitos, diseñamos unos algoritmos híbridos que combinan varios métodos para formar uno nuevo, o en el problema de planificación de trabajadores incluimos unos operadores específicos al problema.

Conclusions

In this PhD thesis we have performed a generic study on parallel metaheuristic techniques to solve real-world applications. This chapter includes a summary of the main conclusions that we have extracted. To do that, we follow an organization from a high-level of abstraction to a numerical point of view.

The first step of our research has been to perform a unified study of metaheuristics. Specially, we focused on parallel variants of these methods. We also offer a new common mathematical model to describe any parallel metaheuristic. This model allows us to characterize the different metaheuristics, indicating their common features and highlighting their differences that provoke different parallel dynamics of search.

Once we have understood these models, we focus on two important issues: implementing and evaluating metaheuristics. With respect to implementation, we have discussed the necessity of a structured and flexible design in parallel metaheuristic. Software quality is often missed in this domain since researchers are mostly worried about solving the target application. Also, many non-computer scientists are unaware of the advantages that software engineering can bring to their future applications and experimentation phases, making them easier and methodological. We have also offered some hints to help interested researchers in profiting from the numerous advantages of using OOP in their application and theoretical studies. Mallba [31] is a library containing some of these results.

We also considered the issue of reporting experimental research with parallel metaheuristics. Since this is a difficult task per se, the main issues of an experimental design are highlighted. We do not enter the complex and deep field of pure statistics in this study (space problems), but just some important ideas to guide researchers in their work. As could be expected, we have also defined parallel performance metrics that allow to compare parallel approaches against other techniques of the literature. Besides, we have shown the importance of the statistical analysis to support conclusions, also in the parallel metaheuristic field.

In a subsequent phase, we perform several experimental test to study the behavior of parallel metaheuristics. Our aim has been to provide a clear seminal evidence to create a body of knowledge that will lead us from our understanding of LAN parallel optimization heuristics to their WAN, and even to their Grid execution. In these studies we observed that the parametrization of the algorithm is very important. For example when we use heterogeneous networks to communicate our processes, we noticed that the topology is one of the more important parameter influencing the performance. This is a common situation when we use systems with different network technologies, e.g., a Fast/Gigabit Ethernet for intranet and ATM circuits for Internet. Configurations which minimize the number of messages over the slowest communication link (e.g., over the Internet) allow to reduce the communication overhead provoking a smaller overall runtime. In addition, the resulting model where the topology is divided in several groups with high interaction intragroup and sparse communication intergroup, provoke a balance between diversification and intensification very adequate for complex problems.

We also analyzed the dynamics of the parallel metaheuristics from a theoretical point of view. In concrete, we have studied study the influence of the migration policy on the selection pressure. We propose a very accurate model to predict actual behaviors. We observed that the topology and the migration frequency are the most important parameters, and their correct configuration can lead to a optimum performance of the method. A clear future step is studying how we can use this information in the parallel algorithm quantitatively to change the configuration and to guide the search during the execution, maybe in the way already made in other algorithms [15].

Finally, we apply the knowledge obtained in the previous analyses and experiment in order to design parallel metaheuristics that tackle adequately with real-world problems. We have observed

that our parallel metaheuristics have allowed to deal with very large and complex instances of many problems. The selected problems have different features, e.g., the design of logic circuit or the workforce planning problem has a large number of hard equality constraints, and the search space of the DNA fragment assembly or of the tagging is very large. We have observed that the parallelism did not only allow to reduce the execution time but it also allowed to improve the quality of the solutions. Even when the parallel algorithms were executed in a single processor, they outperformed the serial one, proving clearly that the serial and the parallel methods are different algorithms with different behaviors.

The general dynamic of the parallel metaheuristics have allow dealing with instances that classical methods can not solve adequately. For example, previous works on DNA fragment assembly have usually faced 30K base pairs (bps) long instances, while we have tackled here a 77K bps long one. As a further example, classical tagging methods can not be applied in contexts that takes into account right tags, while our parallel metaheuristics have solved accurately instances with this kind of contexts. However this generic behavior is not powerful enough to solve adequately an arbitrary problem. In some cases, we have been forced to include problem-dependent knowledge in a general search algorithm. This information can be incorporated in two ways: (a) the problem-knowledge is included as problem-dependent representation and/or operators, and (b) several algorithms are combined in some manner to yield a new hybrid algorithm. We have applied these two paradigms in this thesis; for example, we designed an specific operator for the workforce planning problem, and also, we have designed two hybrid algorithms that combine a GA and a SA, to solve the design of combinatorial circuit.

Trabajos Futuros

De la investigación realizada para el desarrollo de la tesis han surgido diferentes líneas de trabajo que siguen bajo estudio, las principales las resumimos a continuación.

Una primera línea de trabajo futuro consiste en incorporar la información obtenida de los modelos teóricos de la presión selectiva para algoritmos paralelos distribuidos en el comportamiento del propio algoritmo, permitiendo así ajustar esta presión de forma dinámica durante la ejecución dependiendo del estado de la búsqueda, guiando al proceso de forma más efectiva. Esta técnica de ajustar de forma dinámica la presión selectiva ya está dando muy buenos resultados en otros tipos de algoritmos descentralizados como los celulares [15]. Una idea para este ajuste dinámico es partiendo de la evolución del algoritmo hasta el momento y utilizando nuestro modelo matemático predecir la presión selectiva, y utilizar esta información para modificar los parámetros de la migración. Por ejemplo si la presión selectiva se detecta que es demasiada alta y va a producir la convergencia prematura, se puede intentar corregirla, aumentando el periodo de migración o disminuyendo el número de individuos intercambiados.

Como vimos en nuestros análisis experimentales (Capítulo 5), la definición de la topología en las plataformas de área extensa en un aspecto muy importante en el rendimiento del algoritmo, por tanto parece interesante diseñar un nuevo algoritmo que pueda acceder al estado del entorno donde se ejecuta y modificar su comportamiento para ajustarse a él. Este aspecto es también muy interesante en relación con los sistemas grid, donde tenemos una plataforma totalmente heterogénea y cambiante durante la ejecución. En este sentido, los sistemas grid suelen proporcionar información sobre el entorno (carga de las máquinas, retardo en los enlaces, etc.) y la idea es de disponer de esta información para tomar decisiones que afecten a la migración. Por ejemplo, se pueden crear pequeños nichos de procesos unidos por enlaces muy rápidos que se comuniquen con bastante frecuencia, y comunicar estos nichos mediante migraciones dispersas que utilicen los enlaces más lentos.

Finalmente, viendo los buenos resultados obtenidos con los modelos paralelas en las aplicaciones probadas, todas ellas de gran dificultad, una línea interesante es extender estos modelos hacia otros dominios de aplicaciones de gran interés en la actualidad como los problemas de telecomunicaciones, como por ejemplo, optimización en MANETs (*Mobile Ad Hoc Networks*) o asignación de frecuencias. Por ejemplo, en este último problema mencionado, existen muchas restricciones en cuanto a las frecuencias que se pueden asignar para evitar los conflicto entre antenas cercanas. En este sentido (poseer muchas restricciones) se puede asemejar al diseño de circuitos que también posee muchas restricciones y por lo tanto el conocimiento extraído de ese problema se puede extrapolar a este problema de telecomunicaciones.

Parte V

Apéndice: Publicaciones

En este apéndice se relacionan las publicaciones derivadas del proceso de desarrollo de esta tesis. Estas publicaciones avalan su interés en la comunidad científica internacional al haberse producido en foros de prestigio y revisadas por investigadores especializados. En la Figura 11.5 se muestra la relación entre las publicaciones y las fases que han guiado la realización de la tesis. Después se listan los datos concretos de cada una de estas publicaciones.

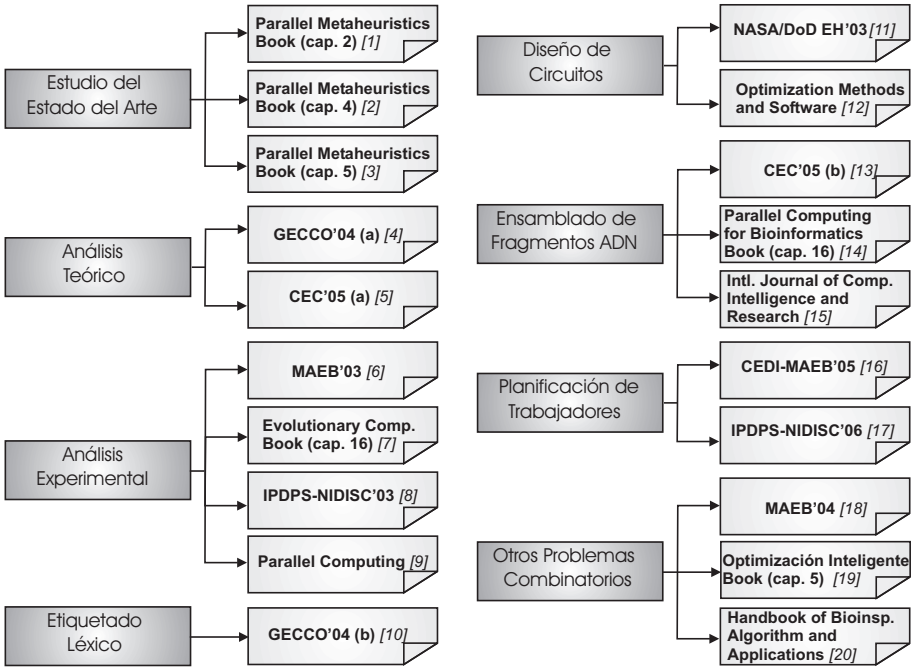


Figura 11.5: Publicaciones realizadas durante el desarrollo de la tesis.

[1] G. Luque y E. Alba, **"Measuring the Performance of Parallel Metaheuristics"**, E. Alba (editor), *Parallel Metaheuristics: A New Class of Algorithms*, capítulo 2, pp. 43-62, John Wiley & Sons, Octubre 2005. (*Capítulo de libro*). Revisión donde se muestra las diferentes medidas del rendimiento paralelo y cómo se deben utilizar para evaluar las metaheurísticas paralelas.

[2] E.-G. Talbi, E. Alba, G. Luque y N. Melab, **"Metaheuristics and Parallelism"**, E. Alba (editor), *Parallel Metaheuristics: A New Class of Algorithms*, capítulo 4, pp. 79-104, John Wiley & Sons, Octubre 2005. (*Capítulo de libro*). Revisión de los modelos paralelos para metaheurísticas y breve estado del arte por cada tipo de metaheurística.

[3] G. Luque, E. Alba y B. Dorronsoro, **"Parallel Genetic Algorithm"**, E. Alba (editor), *Parallel Metaheuristics: A New Class of Algorithms*, capítulo 5, pp. 107-126, John Wiley & Sons, Octubre 2005. (*Capítulo de libro*). Revisión en detalle de los modelos paralelos aplicados a los algoritmos genéticos y un estado del arte y tendencia de GA paralelos.

[4] E. Alba y G. Luque, **"Growth Curves and Takeover Time in Distributed Evolutionary Algorithms"**, K. Deb et al. (editores), *Genetic and Evolutionary Computation Conference GECCO'04*, capítulo 5, pp. 864-876, volumen 3102 de Lecture Notes in Computer Science, Springer-Verlag, Seattle, Washington, EE.UU., 2004. (*Congreso internacional*).

Estudio teórico inicial de la influencia de la frecuencia y el ratio de migración en la presión selectiva.

[5] E. Alba y G. Luque, **"Theoretical Models of Selection Pressure for dEAs: Topology Influence"**, *IEEE Congress on Evolutionary Computation CEC'05*, pp. 214 - 222, Edinburgh, UK, 2005. (*Congreso internacional*). Extensión del anterior donde se propone un modelo matemático preciso para modelar la topología, la frecuencia y el ratio de migración y es evaluado con diferentes selecciones.

[6] E. Alba y G. Luque, **"Algoritmos Híbridos y Paralelos para la Resolución de Problemas Combinatorios"**, *II Congreso Español sobre Metahurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'03)*, Gijón, España, pages 353-362, 2003. (*Congreso nacional*). Estudio inicial sobre el comportamiento de algoritmos paralelos en LAN respecto a su comportamiento secuencial sobre un banco de problemas muy extenso.

[7] E. Alba y G. Luque, **"Performance of Distributed GAs on DNA Fragment Assembly"**, N. Nedjah, E. Alba y L. de Macedo Mourelle, *Parallel Evolutionary Computations*, Capítulo 16, Springer, 2006. (*Capítulo de libro*). Estudio sobre el comportamiento de un algoritmo evolutivo distribuido sobre diferentes plataformas LAN, donde se examina la influencia de la política de migración.

[8] E. Alba y G. Luque, **"Parallel LAN/WAN Heuristics for Optimization"**, *IPDPS-NIDISC03*, p. 147, Nize, France, 2003. (*Congreso internacional*). Estudio experimental donde se examina el cambio de comportamiento de los algoritmos paralelos cuando pasan de ser ejecutados en plataformas LAN canónicas a WAN canónicas

[9] E. Alba y G. Luque, **"Parallel LAN/WAN Heuristics for Optimization"**, *Parallel Computing* vol. 30, issue 5-6, pp. 611-628. 2004. (*Revista internacional*). Extensión del trabajo anterior, donde se estudian en más detalle los comportamientos LAN y WAN, con diferentes plataformas y topologías WAN.

[10] L. Araujo, G. Luque y E. Alba, **"Metaheuristics for Natural Language Tagging"**, K. Deb et al. (editores), *Genetic and Evolutionary Computation Conference GECCO'04*, capítulo 5, pp. 864-876, volumen 3102 de Lecture Notes in Computer Science, Springer-Verlag, Seattle, Washington, EE.UU., 2004. (*Congreso internacional*). Aplicación de varias metaheurísticas para la resolución del problema del etiquetado léxico.

[11] E. Alba, C. Coello, G. Luque y A. Hernández, **"Comparing Different Serial and Parallel Heuristics to Design Combinatorial Logic Circuits"**, *NASA/DoD Conference on Evolvable Hardware, EH'03*, Chicago, pp. 3-12, 2003. (*Congreso internacional*). Estudio inicial donde se aplican de forma preliminar varias metaheurísticas paralelas para resolver el problema del diseño de circuitos.

[12] E. Alba, G. Luque, C. Coello, y E. Hernández, **"A Comparative Study of Serial and Parallel Heuristics Used to Design Combinational Logic Circuits"**, *Optimization Methods and Software*, 2006. (*Revista internacional*). Trabajo donde se estudia el comportamiento de un conjunto de metaheurísticas aplicadas a un amplio conjunto de circuitos de gran complejidad y donde se muestra su excelente comportamiento, obteniendo en algunos casos las mejores soluciones encontradas hasta el momento.

[13] E. Alba, G. Luque y S. Khuri, **"Assembling DNA Fragments with Parallel Algorithms'**, *IEEE Congress on Evolutionary Computation CEC'05*, pp. 57-65, Edinburgh, UK,

2005. (*Congreso internacional*). Aplicación de varias metaheurísticas paralelas para resolver el problema del ensamblado de fragmentos de ADN.

[14] E. Alba, S. Khuri y G. Luque, **"Parallel Algorithms for Solving the Fragment Assembly Problem in DNA Strands"**, A. Y. Zomaya (editor), *Parallel Computing for Bioinformatics and Computational Biology*, capítulo 16, John Wiley & Sons, 2006 (a aparecer). (*Capítulo de libro*). Estudio inicial donde se estudia el efecto de la parametrización del algoritmo a la hora de resolver el problema del ensamblado de fragmentos de ADN.

[15] G. Luque y E. Alba, **"Metaheuristics for the DNA Fragment Problem"**, *International Journal of Computational Intelligence Research (IJCIR)* vol. 1, issue 2, pp. 98-108, 2005. (*Revista internacional*). Extensión del artículo del CEC'05 [26], donde se estudia el rendimiento de un conjunto ampliado de metaheurísticas con respecto al algoritmo original.

[16] E. Alba, M. Laguna y G. Luque, **"Workforce Planning with a Parallel Genetic Algorithm"**, In CEDI-MAEB'05, pages 911 - 919, Granada, España, 2005. (*Congreso nacional*). Estudio inicial donde se estudia el comportamiento de un algoritmo genético paralelo para el problema de planificación de trabajadores.

[17] E. Alba, G. Luque y F. Luna, **"Workforce Planning with Parallel Algorithms"**, IPDPS-NIDISC'06, Rhodas, Grecia, 2006. (*Congreso internacional*). Aplicación de varias metaheurísticas paralelas para resolver el problema de planificación de trabajadores.

[18] E. Alba, J. F. Chicano, B. Dorronsoro y G. Luque, **"Diseño de Códigos Correctores de Errores con Algoritmos Genéticos"**, C. Hervás et al. (editores), *Actas del Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'04)*, pp. 51-58, Córdoba, España, 2004. (*Congreso nacional*). Aplicación de varias técnicas metaheurísticas para resolver el problema del diseño de códigos correctores de errores, donde se observa que las técnicas híbridas son las que mejores resultados ofrecen.

[19] E. Alba, J. F. Chicano, C. Cotta, B. Dorronsoro, F. Luna, G. Luque y A. J. Nebro, **"Metaheurísticas Secuenciales y Paralelas para Optimización de Problemas Complejos"**, G. Joya, M. Atencia, A. Ochoa, S. Allende (editores), *Optimización Inteligente: Técnicas de Inteligencia Computacional para Optimización*, capítulo 5, pp. 185-214, 2004, (*Capítulo de libro*). Resumen de los resultados obtenidos al aplicar diferentes técnicas paralelas para resolver un amplio conjunto de problemas complejos.

[20] E. Alba, J. F. Chicano, F. Luna, G. Luque y A. J. Nebro, **"Advanced Evolutionary Algorithms for Training Neural Networks"**, S. Olariu, A. Y. Zomaya. (editores), *Handbook of Bioinspired Algorithms and Applications*, capítulo 26, pp. 453-467, CRC Press, 2006. (*Capítulo de libro*). Aplicación de diferentes técnicas metaheurísticas que incluyen tanto técnicas puras e híbridas para el entrenamiento de redes neuronales.

Bibliografía

[1] K. Aardal. Capacitated Facility Location: Separation Algorithm and Computational Experience. *Mathematical Programming*, 81:149–175, 1998.

[2] M. Abd-El-Barr, S. M. Sait, B. A.B. Sarif, and U. Al-Saiari. A modified ant colony algorithm for evolutionary design of digital circuits. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, pages 708–715, Canberra, Australia, December 2003. IEEE Press.

[3] P. Adamidis and V. Petridis. Co-operating Populations with Different Evolution Behaviors. In *Proc. of the Third IEEE Conf. on Evolutionary Computation*, pages 188–191, New York, 1996. IEEE Press.

[4] C.W. Ahn, D.E. Goldberg, and R.S. Ramakrishna. Multiple-deme parallel estimation of distribution algorithms: Basic framework and application. Technical Report 2003016, University of Illinois, 2003.

[5] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:293–430, 2003.

[6] R.M. Aiex, S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez. Cooperative multi-thread parallel tabu search with an application to circuit partitioning. *LNCS 1457*, pages 310–331, 1998.

[7] R.M. Aiex and M.G.C. Resende. A methodology for the analysis of parallel GRASP strategies. *AT&T Labs Research TR*, Apr. 2003.

[8] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path relinking for the three-index assignment problem. *TR, AT&T Labs Research, Florham Park, NJ 07932, USA*, 2000.

[9] S.G. AKL. *Diseño y Análisis de Algoritmos Paralelos*. RA-Ma, 1992.

[10] U. S. Al-Saiari. Digital circuit design through simulated evolution. Master's thesis, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, November 2003.

[11] E. Alba. Parallel evolutionary algorithms can achieve super-linear performace. *Information Processing Letters*, 82:7–13, 2002.

[12] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, October 2005.

[13] E. Alba, J.F. Chicano, B. Dorronsoro, and G. Luque. Diseño de códigos correctores de errores con algoritmos genéticos. In *Proceedings of the Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB04)*, pages 51–58, Córdoba, Spain, February 2004.

[14] E. Alba, J.F. Chicano, F. Luna, G. Luque, and A.J. Nebro. *Handbook of Bioinspired Algorithms and Applications*, volume 6 of *Chapman & Hall/CRC Computer & Information Science Series*, chapter 26. Advanced Evolutionary Algorithms for Training Neural Networks, pages 453–467. CRC Press, 2005.

[15] E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.

[16] E. Alba, S. Khuri, and G. Luque. *Parallel Computing for Bioinformatics and Computational Biology*, chapter 16. Parallel Algorithms for Solving the Fragment Assembly Problem in DNA Strands. Wiley, May 2006. (to appear).

[17] E. Alba, M. Laguna, and G. Luque. Workforce planning with a parallel genetic algorithm. In *CEDI-MAEB'05*, pages 911–919, Granada, Spain, September 2005. Thomson.

[18] E. Alba, F. Luna, A.J. Nebro, and J.M. Troya. Parallel heterogeneous GAs for continuous optimization. *Parallel Computing*, 30:699–719, 2004.

[19] E. Alba and G. Luque. Algoritmos híbridos y paralelos para la resolución de problemas combinatorios. In *Proceedings of the Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB03)*, pages 353–362, Gijón, Spain, February 2003.

[20] E. Alba and G. Luque. Parallel LAN/WAN heuristics for optimization. In *IPDPS-NIDISC03*, page 147, Nize, France, 2003. IEEE Press.

[21] E. Alba and G. Luque. Growth curves and takeover time in distributed evolutionary algorithms. In K. Deb, editor, *Genetic and Evolutionary Computation Conference GECCO-04*, number 3102 in LNCS, pages 864–876, Seattle, Washington, July 2004. Springer-Verlag.

[22] E. Alba and G. Luque. Parallel LAN/WAN heuristics for optimization. *Parallel Computing*, 30(5–6):611–628, 2004.

[23] E. Alba and G. Luque. Theoretical models of selection pressure for deas: Topology influence. In *IEEE Congress on Evolutionary Computation CEC-05*, pages 214–222, Edinburgh, UK, September 2005. IEEE Press.

[24] E. Alba and G. Luque. *Parallel Evolutionary Computations*, chapter Performance of Distributed GAs on DNA Fragment Assembly. Springer, 2006.

[25] E. Alba, G. Luque, C. A. Coello Coello, and E. Hernández Luna. A comparative study of serial and parallel heuristics used to design combinational logic circuits. *Optimization Methods and Software*, 2006 (to appear).

[26] E. Alba, G. Luque, and S. Khuri. Assembling DNA Fragments with Parallel Algorithms. In B. McKay, editor, *CEC-2005*, pages 57–65, Edinburgh, UK, 2005.

[27] E. Alba, G. Luque, and F. Luna. Workforce planning with parallel algorithms. In *IPDPS-NIDISC'06*, Rhodes Island, Greece, April 2006 (por aparecer). ACM Press.

[28] E. Alba and A.J. Nebro. *Parallel Metaheuristics: A New Class of Algorithms*, chapter 3. New Technologies in Parallelism, pages 63–78. John Wiley & Sons, October 2005.

[29] E. Alba, A.J. Nebro, and J.M. Troya. Heterogeneous Computing and Parallel Genetic Algorithms. *J. of Parallel and Distributed Computing*, 62:1362–1385, 2002.

[30] E. Alba, E.-G. Talbi, G. Luque, and N. Melab. *Parallel Metaheuristics: A New Class of Algorithms*, chapter 4. Metaheuristics and Parallelism, pages 79–104. John Wiley & Sons, October 2005.

[31] E. Alba and the MALLBA Group. MALLBA: A library of skeletons for combinatorial optimization. In R.F.B. Monien, editor, *Proceedings of the Euro-Par*, volume 2400 of *LNCS*, pages 927–932, Paderborn, Germany, 2002. Springer-Verlag.

[32] E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[33] C.F. Allex. *Computational Methods for Fast and Accurate DNA Fragment Assembly*. UW technical report CS-TR-99-1406, Department of Computer Sciences, University of Wisconsin-Madison, 1999.

[34] E. Alba amd J.F. Chicano, C. Cotta, B. Dorronsoro, F. Luna, G. Luque, and A.J. Nebro. *Optimización Inteligente: Técnicas de Inteligencia Computacional para Optimización*, chapter 5. Metaheurísticas Secuenciales y Paralelas para Optimización de Problemas Complejos, pages 185–214. Universidad de Málaga. Servicio de Publicaciones e Intercambio Científico, July 2004.

[35] D. Andre and J.R. Koza. Parallel genetic programming: a scalable implementation using the transputer network architecture. In *Advances in genetic programming: volume 2*, pages 317–337. MIT Press, 1996.

[36] D. Andre and J.R. Koza. A parallel implementation of genetic programming that achieves super-linear performance. In H.R. Arabnia, editor, *Proceedings of the International Conf. on Parallel and Distributed Processing Techniques and Applications*, volume III, pages 1163–1174, 1996.

[37] L. Araujo. Part-of-speech tagging with evolutionary algorithms. In *Proc. of the Int. Conf. on Intelligent Text Processing and Computational Linguistics, LNCS 2276*, pages 230–239. Springer-Verlag, 2002.

[38] L. Araujo, G. Luque, and E. Alba. Metaheuristics for Natural Language Tagging. In K. Deb et al., editor, *Genetic and Evolutionary Computation Conference (GECCO-2004)*, volume 3102 of *LNCS*, pages 889 – 900, Seattle, Washington, 2004.

[39] M. G. Arenas, P. Collet, A. E. Eiben, M. Jelasity, J. J. Merelo, B. Paechter, M. Preub, and M. Schoenauer. A framework for distributed evolutionary algorithms. In J. J. Merelo, P. Adamidis, H. G. Beyer, J. L. Fernández-Villacañas, and H. P. Schwefel, editors, *Seventh International Conference on Parallel Problem Solving from Nature*, pages 665–675, 2002.

[40] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[41] R.A. Baeza-Yates and B.A. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[42] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming. An Introduction. On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, San Francisco, California, 1998.

[43] R.S. Barr and B.L. Hickman. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. *ORSA Journal on Computing*, 5(1):2–18, 1993.

[44] J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[45] T.C. Belding. The distributed genetic algorithm revisited. In L.J. Eshelman, editor, *6th International Conference on Genetic Algorithms*, pages 114–121, Los Altos, CA, 1995. Morgan Kaufmann.

[46] M.J. Blesa, Ll. Hernandez, and F. Xhafa. Parallel Skeletons for Tabu Search Method. *In the* 8^{th} *Intl. Conf. on Parallel and Distributed Systems, Korea, IEEE Computer Society Press*, pages 23–28, 2001.

[47] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms.

[48] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[49] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit Per Second Local Area Network. *IEEE Micro*, pages 29–36, 1995.

[50] M. Bolondi and M. Bondaza. Parallelizzazione di un Algoritmo per la Risoluzione del Problema del Commesso Viaggiatore. Master's thesis, Politecnico di Milano, Dipartamento di Elettronica e Informazione, 1993.

[51] B. Bullnheimer, G. Kotsis, and C. Strauß. Parallelization Strategies for the Ant System. Technical Report 8, University of Viena, October 1997.

[52] M. Cadoli, M. Lenzerini, and A. Schaerf. LOCAL++: A C++ framework for local search algorithms. Technical Report DIS 11-99, University of Rome "La Sapienza", 1999.

[53] S. Cahon, N. Melab, and E-G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004.

[54] S. Cahon, E-G. Talbi, and N. Melab. PARADISEO: A framework for parallel and distributed biologically inspired heuristics. In *IPDPS-NIDISC'03*, page 144, Nize, France, 2003.

[55] A. E. Calaor, B. O. Corpus, and A. Y. Hermosilla. Parallel Hybrid Adventures with Simulated Annealing and Genetic Algorithms. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks* $(ISPAN)'02$, pages 39–45, 2002.

[56] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*, chapter 7. Migration, Selection Pressure, and Superlinear Speedups, pages 97–120. Kluwer, 2000.

[57] U. K. Chakraborty, K. Deb, and M. Chakraborty. Analysis of Selection Algorithms: A Markov Chain Approach. *Evolutionary Computation*, 4(2):133–167, 1997.

[58] J.A. Chandy and P. Banerjee. Parallel Simulated Annealing Strategies for VLSI Cell Placement. *Proc. of the 1996 Intl. Conf. on VLSI Design, Bangalore, India*, Jan. 1996.

[59] E. Charniak. *Statistical Language Learning*. MIT press, 1993.

[60] T. Chen and S.S. Skiena. Trie-based data structures for sequence assembly. In *The Eighth Symposium on Combinatorial Pattern Matching*, pages 206–223, 1998.

[61] W.S. Cleveland. *Elements of Graphing Data*. AT&T Bell Laboratories, Wadsworth, Monteray, CA, 1985.

[62] C. A. Coello Coello and A. Hernández Aguirre. Design of combinational logic circuits through an evolutionary multiobjective optimization approach. *Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, 16(1):39–53, January 2002.

[63] C. A. Coello Coello, A. Hernández Aguirre, and B. P. Buckles. Evolutionary Multiobjective Design of Combinational Logic Circuits. In J. Lohn, A. Stoica, D. Keymeulen, and S. Colombano, editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170. IEEE Computer Society, 2000.

[64] C. A. Coello Coello, E. Alba, G. Luque, and A. Hernández Aguirre. Comparing Different Serial and Parallel Heuristics to Design Combinatorial Logic Circuits. In *NASA/DoD Conference on Evolvable Hardware*, pages 3–10. IEEE Press, 2003.

[65] C. A. Coello Coello, A. D. Christiansen, and A. Hernández Aguirre. Automated Design of Combinational Logic Circuits using Genetic Algorithms. In D. G. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 335–338. Springer-Verlag, University of East Anglia, England, April 1997.

[66] C. A. Coello Coello, A. D. Christiansen, and A. Hernández Aguirre. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design*, 2(4):299–314, June 2000.

[67] C. A. Coello Coello, R. L. Zavala Gutiérrez, B. Mendoza García, and A. Hernández Aguirre. Ant Colony System for the Design of Combinational Logic Circuits. In Julian Miller, Adrian Thompson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware*, pages 21–30, Edinburgh, Scotland, April 2000. Springer-Verlag.

[68] C. A. Coello Coello, E. Hernández Luna, and A. Hernández Aguirre. Use of particle swarm optimization to design combinational logic circuits. In Pauline C. Haddow, Andy M. Tyrell, and Jim Torresen, editors, *5th International Conference in Evolvable Systems: From Biology to Hardware, ICES 2003*, volume 2606 of *Lecture Notes in Computer Science*, pages 398–409, Trondheim, Norway, 2003. Springer-Verlag.

[69] C. Cotta and J. M. Troya. On Decision-Making in Strong Hybrid Evolutionary Algorithms. In A. P. Del Pobil, J. Mira, and M. Ali, editors, *Tasks and Methods in Applied Artificial Intelligence*, volume 1416 of *Lecture Notes in Computer Science*, pages 418–427. Springer-Verlag, Berlin Heidelberg, 1998.

[70] T.G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacitated Network Design. *Journal of Heuristics*, 8:601–627, 2002.

[71] T.G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative Parallel Variable Neighborhood Search for the p-Median. *Journal of Heuristics*, 10(3), 2004.

[72] T.G. Crainic and M. Toulouse. *Handbook of Metaheuristics*, chapter Parallel Strategies for Metaheuristics, pages 475–513. Kluwer Academic Publishers, 2003.

[73] B. Le Cun. Bob++ library illustrated by VRP. In *European Operational Research Conference (EURO'2001)*, page 157, Rotterdam, 2001.

[74] J. M. Daida, S. J. Ross, and B. C. Hannan. Biological Symbiosis as a Metaphor for Computational Hybridization. In L. J. Eshelman, editor, *Sixth International Conference on Genetic Algorithms*, pages 328–335. Morgan Kaufmann, 1995.

[75] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[76] I. de Falco, R. del Balio, and E. Tarantino. Testing parallel evolution strategies on the quadratic assignment problem. In *Proc. IEEE Int. Conf. in Systems, Man and Cybernetics*, volume 5, pages 254–259, 1993.

[77] S. J. DeRose. Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics*, 14:31–39, 1988.

[78] K.F. Doerner, R.F. Hartl, G. Kiechle, M. Lucka, and M. Reimann. Parallel Ant Systems for the Capacited VRP. In J. Gottlieb and G.R. Raidl, editors, *EvoCOP'04*, pages 72–83. Springer-Verlag, 2004.

[79] V. Donalson, F. Berman, and R. Paturi. Program speedup in heterogeneous computing network. *Journal of Parallel and Distributed Computing*, 21:316–322, 1994.

[80] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartamento di Elettronica, Politecnico di Milano, 1992.

[81] M. Dorigo and T. Stützle. *Handbook of Metaheuristics*, volume 57 of *International Series In Operations Research and Management Science*, chapter 9.-The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, pages 251–285. Kluwer Academic Publisher, 2003.

[82] Dimitris C. Dracopoulos and Simon Kent. Bulk synchronous parallelisation of genetic programming. In Jerzy Waśniewski, editor, *Applied parallel computing : industrial strength computation and optimization ; Proceedings of the third International Workshop, PARA'96*, pages 216–226, Berlin, Germany, 1996. Springer Verlag.

[83] R. Uzsoy E. Demirkol, S.V. Mehta. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109:137–141, 1998.

[84] J. Eckstein, C. A. Phillips, and W. E. Hart. PICO: An object-oriented framework for parallel branch and bound. Technical report, RUTCOR, 2000.

[85] J. Eggermont, A.E. Eiben, and J.I. van Hemert. A Comparison of Genetic Programming Variants for Data Classification. In David J. Hand, JoostÑ. Kok, and Michael R. Berthold, editors, *Advances in Intelligent Data Analysis, Third International Symposium, IDA-99*, volume 1642, pages 281–290, Amsterdam, The Netherlands, 9–11 1999. Springer-Verlag.

[86] A.E. Eiben and M. Jelasity. A critical note on experimental reseeach methodology in ec. In *Congress on Evolutionary Computation 2002*, pages 582–587. IEEE Press, 2002.

[87] M.L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16, 1993.

[88] J. A. Erickson, R. E. Smith, and D. E. Goldberg. SGA-Cube, a simple genetic algorithm for ncube 2 hypercube parallel computers. Technical Report 91005, The University of Alabama, 1991.

[89] L.J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *Foundations of Genetic Algorithms, 1*, pages 265–283. Morgan Kaufmann, 1991.

[90] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optmization*, 6:109–133, 1999.

[91] F. Fernández, M. Tomassini, W.F. Punch, III, and J.M. Sánchez-Pérez. Experimental study of multipopulation parallel genetic programming. In *Proc. of the European Conf. on GP*, pages 283–293. Springer, 2000.

[92] C.N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 51:243–267, 1994.

[93] A. Fink, S. Vo, and D. Woodruff. Building Reusable Software Components for Heuristc Search. In P. Kall and H.-J. Luthi, editors, *Operations Research Proc.*, pages 210–219, Berlin, Germany, 1998. Springer.

[94] M. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, C-21:94, 1972.

[95] D. B. Fogel and Z. Michalewicz, editors. *How to Solve It: Modern Heuristics*. Springer, 1999.

[96] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, 1966.

[97] G. Folino, C. Pizzuti, and G. Spezzano. CAGE: A tool for parallel genetic programming applications. In J.F. Miller et al., editor, *Proceedings of EuroGP'2001*, LNCS 2038, pages 64–73, Italy, 2001. Springer-Verlag.

[98] G. D. Forney. The viterbi algorithm. *Proceedings of The IEEE*, 61(3):268–278, 1973.

[99] Message Passing Interface Forum. Mpi: A message passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):165–414, 1994.

[100] I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Fransisco, 1999.

[101] I. Foster and C. Kesselman. Globus: a metacomputing infraestructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[102] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 7–9, San Francisco, California, 2001.

[103] C. Gagne and M. Parizeau. Open BEAGLE: A new C++ evolutionary computation framework. In *Proceeding of GECCO 2002*, New York, NY, USA, 2002.

[104] A. A. El Gamal, L. A. Hemachandra, I. Shperling, and V. K. Wei. Using Simulated Annealing to Design Good Codes. *IEEE Trans. Information Theory*, 33(1):116–123, January 1987.

[105] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[106] F. García-López, B. Melián-Batista, J. Moreno-Pérez, and J.M. Moreno-Vega. Parallelization of the Scatter Search. *Parallel Computing*, 29:575–589, 2003.

[107] F. García-López, B. Melián-Batista, J.A. Moreno-Pérez, and J.M. Moreno-Vega. The Parallel Variable Neighborhood Search for the p-Median Problems. *Journal of Heuristics*, 8(3):375–388, 2002.

[108] F. García-López, M. García Torres, B. Melián-Batista, J. Moreno-Pérez, and J.M. Moreno-Vega. Solving Feature Subset Selection Problem by a Parallel Scatter Search. *European Jour. of Operational Research*, 2004.

[109] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA, 1979.

[110] L. Di Gaspero and A. Schaerf. EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *4th Metaheuristics International Conference (MIC'2001)*, pages 287–292, 2001.

[111] M. Giacobini, E. Alba, and M. Tomassini. Selection Intensity in Asynchronous Cellular Evolutionary Algorithms. In E. Cantú-Paz, editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 955–966, Chicago, USA, 2003.

[112] M. Giacobini, A. Tettamanzi, and M. Tomassini. Modelling Selection Intensity for Linear Cellular Evolutionary Algorithms. In Pierre Liardet et al., editor, *Artificial Evolution, Sixth International Conference*. Springer Verlag, 2003.

[113] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[114] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.

[115] F. Glover. A template for Scatter Search and Path Relinking. In J.-K-Hao et al., editor, *Altificial Evolution*, number 1363 in Lecture Notes in Computer Science, pages 13–54. Springer, 1998.

[116] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.

[117] F. Glover, G. Kochenberger, M. Laguna, and T. Wubbena. Selection and Assignment of a Skilled Workforce to Meet Job Requirements in a Fixed Planning Period. In *Actas del Tercer Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 636–641, Córdoba, 2004.

[118] F. Glover and M. Laguna. *Tabu search*. Kluwer, 1997.

[119] D. E. Goldberg and K. Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In Gregory J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, San Mateo, CA, 1991.

[120] B. Golden and W. Stewart. Empirical Analisys of Heuristics. In E. Lawlwer, J. Lenstra, A. Rinnooy Kan, and D. Schoys, editors, *The Traveling Salesman Problem, a Guided Tour of Combinatorial Optimization*, pages 207–249, Chichester, UK, 1985. Wiley.

[121] E.D. Goodman. An Introduction to GALOPPS v3.2. Technical Report 96-07-01, GARAGE, I.S. Lab. Dpt. of C. S. and C.C.C.A.E.M., Michigan State Univ., East Lansing, MI, 1996.

[122] T. G. W. Gordon and P. J. Bentley. On evolvable hardware. In S. Ovaska and L. Sztandera, editors, *Soft Computing in Industrial Electronics*, pages 279–323, Heidelberg, Germany, 2003. Physica-Verlag.

[123] M. Gorges-Schleuter. ASPARAGOS an asynchronous parallel genetic optimization strategy. In *Proceedings of the third international conference on Genetic algorithms*, pages 422–427. Morgan Kaufmann, 1989.

[124] M. Gorges-Schleuter. An Analysis of Local Selection in Evolution Strategies. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 847–854, Orlando, Florida, USA, July 1999. Morgan Kaufmann.

[125] R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[126] P. Green. Phrap. http://www.phrap.org/.

[127] A.S. Grimshaw, A. Natrajan, M.A. Humphrey, M.J. Lewis, A. Nguyen-Tuong, J.F. Karpovich, M.M. Morgan, and A.J. Ferrari. From Legion to Avaki: the Persistence of Vision. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 265–298. John Wiley & Sons Inc., 2003.

[128] UEA CALMA Group. Calma project report 2.4: Parallelism in combinatorial optimisation. Technical Report 2.4, School of Information Systems, University of East Anglia, Norwich, UK, September 18 1995.

[129] V. G. Gudise and G. K. Venayagamoorthy. Evolving digital circuits using particle swarm. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, pages 468–472, Porland, OR, USA, 2003.

[130] H. R. Lourenço and O. Martin, and T. Stützle. *Handbook of Metaheuristics*, chapter Iterated local search, pages 321–353. Kluwer Academic Publishers, 2002.

[131] A.M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee. Parallel Algorithms for FPGA Placement. *Proc. of the Great Lakes Symposium on VLSA (GVLSI 2000), Chicago, IL*, 2000.

[132] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithm. *IEEE Transaction in Evolutionary Computation*, 4:43–63, 2000.

[133] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, Massachusetts, first edition, 1975.

[134] J.Ñ. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

[135] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, 9:868–877, 1999.

[136] IBM. COIN: Common Optimization INterface for operations research, 2000. http://oss.software.ibm.com/developerworks/opensource/coin/index.html.

[137] R. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June/July 1988.

[138] M.S. Jones, G.P. McKeown, and V.J. Rayward-Smith. *Optimization Software Class Libraries*, chapter Distribution, Cooperation, and Hybridization for Combinatorial Optimization, pages 25–58. Kluwer Academic Publishers, 2002.

[139] K.A. De Jong, M.A. Potter, and W.M. Spears. Using problem generators to explore the effects of epistasis. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA)*, pages 338–345. Morgan Kaufmann, 1997.

[140] H. Juille and J.B. Pollack. Massively parallel genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, pages 339–358. MIT Press, Cambridge, MA, USA, 1996.

[141] M. Karnaugh. A map method for synthesis of combinational logic circuits. *Transactions of the AIEE, Communications and Electronics*, I(72):593–599, November 1953.

[142] A.H. Karp and H.P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[143] R.M. Karp. Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane. *Mathematics of Operations Research*, 2:209–224, 1977.

[144] S. Kim. *A structured Pattern Matching Approach to Shotgun Sequence Assembly.* PhD thesis, Computer Science Department, The University of Iowa, 1997.

[145] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[146] K. Klohs. Parallel simulated annealing library. http://www.uni-paderborn.de/fachbereich/AG/monien/ SOFTWARE/PARSA/, 1998.

[147] A. Klose. An LP-based Heuristic for Two-stage Capacitated Facility Location Problems. *Journal of the Operational Research Society*, 50:157–166, 1999.

[148] J. R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992.

[149] S.A. Kravitz and R.A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Trans. in Computer Aided Design*, 6:534–549, 1987.

[150] M. Laguna and R. Martí. *Scatter Search. Metodology and Implementations in C.* Kluwer, Boston, 2003.

[151] M. Laguna and T. Wubbena. Modeling and Solving a Selection and Assignment Problem. In B.L. Golden, S. Raghavan, and E.A. Wasil, editors, *The Next Wave in Computing, Optimization, and Decision Technologies*, pages 149–162. Springer, 2005.

[152] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and Modern Heuristics for the Vehicle Routing Problem. *International Transactions in Operational Research*, 7:285–300, 2000.

[153] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization by learning and simulation of Bayesian and Gaussian networks. Technical Report KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[154] S.Y. Lee and K.G. Lee. Synchronous and asynchronous parallel simulated annealing with multiple markov chains. *IEEE Transactions on Parallel and Distributed Systems*, 7:993–1008, 1996.

[155] D. Levine. PGAPack, parallel genetic algorithm library. http://www.mcs.anl.gov/pgapack.html, 1996.

[156] L. Li and S. Khuri. A Comparison of DNA Fragment Assembly Algorithms. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335, 2004.

[157] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for qap. *DIMACS Series in Discrete Mathematics and Theoritical Computer Science*, 16:237–261, 1994.

[158] S. Lin and B.W. Kernighan. An effective heuristic algorithm for TSP. *Operations Research*, 21:498–516, 1973.

[159] S. C. Lin, W. F. Punch, and E. D. Goodman. Coarse-grain parallel genetic algorithms: Categorization and a new approach. In *Sixth IEEE Symposium on Parallel and Distributed Processing*, pages 28–37, 1994.

[160] J. Linderoth, S. Kulkarni, J.P. Goux, and M. Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 43–50, Pittsburg, Pensylvania, 2000.

[161] F.G. Lobo, C.F. Lima, and H. Mártires. An architecture for massively parallelization of the compact genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, LNCS 3103, pages 412–413. Springer-Verlag, 2004.

[162] S. J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.

[163] S. J. Louis and G. J. Rawlins. Using Genetic Algorithms to Design Structures. Technical Report 326, Computer Science Department, Indiana University, Bloomington, Indiana, February 1991.

[164] S.J. Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, August 1993.

[165] J.A. Lozano, R. Sagarna, and P. Larrañaga. Parallel Estimation of Distribution Algorithms. In P. Larrañaga and J. A. Lozano, editors, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academis Publishers, 2001.

[166] E. Hernández Luna. Diseño de circuitos lógicos combinatorios usando optimización mediante cúmulos de partículas. Master's thesis, Computer Science Section, Electrical Engineering Department, CINVESTAV-IPN, Mexico, D.F., Mexico, February 2004. in Spanish.

[167] G. Luque and E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*, chapter 2. Measuring the Performance of Parallel Metaheuristics, pages 43–62. John Wiley & Sons, October 2005.

[168] G. Luque and E. Alba. Metaheuristics for the DNA Fragment Assembly Problem. *International Journal of Computational Intelligence Research (IJCIR)*, 1(2):98–108, 2006.

[169] G. Luque, E. Alba, and B. Dorronsoro. *Parallel Metaheuristics: A New Class of Algorithms*, chapter 5. Parallel Genetic Algorithms, pages 107–126. John Wiley & Sons, October 2005.

[170] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2000.

[171] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the steiner problem in graphs. *LNCS 1457*, pages 285–297, 1998.

[172] E.J. McCluskey. Minimization of boolean functions. *Bell Systems Technical Journal*, 35(5):1417–1444, November 1956.

[173] C. McGeoch. DTowards an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 1(8):1–15, 1996.

[174] N.F. McPhee, N.J. Hopper, and M.L. Reierson. Sutherland: An Extensible Object-Oriented Software Framework for Evolutionary Computation. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, page 241, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.

[175] M. Mejía-Olvera and E. Cantú-Paz. DGENESIS-software for the execution of distributed genetic algorithms. In *Proceedings XX conf. Latinoamericana de Informática*, pages 935–946, 1994.

[176] N. Melab, E-G. Talbi, S. Cahon, E. Alba, and G. Luque. *Parallel Combinatorial Optimization*, chapter Parallel Metaheuristics: Algorithms and Frameworks. John Wiley & Sons, 2006 (to appear).

[177] R. Mendes, J.R. Pereira, and J. Neves. A Parallel Architecture for Solving Constraint Satisfaction Problems. In *Proceedings of Metaheuristics Int. Conf. 2001*, volume 2, pages 109–114, Porto, Portugal, 2001.

[178] A. Mendiburu, J.A. Lozano, and J. Miguel-Alonso. Parallel estimation of distribution algorithms: New approaches. Technical Report EHU-KAT-IK-1-3, Department of Computer Architecture and Technology, The University of the Basque Country, 2003.

[179] A. Mendiburu, J. Miguel-Alonso, and J.A. Lozano. Implementation and performance evaluation of a parallelization of estimation of bayesian networks algorithms. Technical Report EHU-KAT-IK-XX-04, Computer Architecture and Technology, 2004. Submitted to Parallel Computing.

[180] B. Merialdo. Tagging english text with a probabilistic model. *Comp. Linguistics*, 20(2):155–172, 1994.

[181] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[182] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Heidelberg, third edition, 1996.

[183] R. Michel and M. Middendorf. An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem. In A.E. Eiben et al., editor, *Fifth Int. Conf. on Parallel Problem Solving from Nature*, LNCS 1498, pages 692–701. Springer-Verlag, 1998.

[184] M. Middendorf, F. Reischle, and H. Schmeck. Multi Colony Ant Algorithms. *Journal of Heuristic*, 8:305–320, 2002.

[185] M. Miki, T. Hiroyasu, and M. Kasai. Application of the temperature parallel simulated annealing to continuous optimization problems. *IPSL Transaction*, 41:1607–1616, 2000.

[186] J. Miller, T. Kalganova, N. Lipnitskaya, and D. Job. The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. In *Proceedings of the AISB Symposium on Creative Evolutionary Systems (CES'99)*, Edinburgh, UK, 1999.

[187] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part I. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35, April 2000.

[188] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the Evolutionary Design of Digital Circuits—Part II. *Genetic Programming and Evolvable Machines*, 1(3):259–288, July 2000.

[189] J. F. Miller, P. Thomson, and T. Fogarty. Designing Electronic Circuits Using EAs. Arithmetic Circuits: A Case Study. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 105–131. Morgan Kaufmann, 1998.

[190] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers Oper. Res*, 24:1097–1100, 1997.

[191] D.C. Montgomery. *Design and Analysis of Experiments*. John Wiley, New York, 3rd edition, 1991.

[192] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346, 1998.

[193] H. Mühlenbein, M. Schomish, and J. Born. The parallel genetic algorithm as a function optimizer. *Parallel Computing*, 17:619–632, 1991.

[194] E.W. Myers. Towards simplifying and accurately formulating fragment assembly. *J. of Computational Biology*, 2(2):275–290, 2000.

[195] F.W. Nelson and H. Kucera. Manual of information to accompany a standard corpus of present-day edited american english, for use with digital computers. Technical report, Dep. of Linguistics, Brown University., 1979.

[196] C. Notredame and D.G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24:1515–1524, 1996.

[197] J. Ocenásek and J. Schwarz. The parallel bayesian optimization algorithm. In *European Symp. on Comp. Inteligence*, pages 61–67, 2000.

[198] J. Ocenásek and J. Schwarz. The distributed bayesian optimization algorithm for combinatorial optimization. In *Evolutionary Methods for Design, Optimisation and Control*, pages 115–120, 2001.

[199] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications Inc., 1982.

[200] P. Pardalos, L. Pitsoulis, , and M.G. Resende. A parallel GRASP implementation for the quadratic assignment problem. *Parallel algorithms for irregular problems: State of the art, A. Ferreira and J. Rolim eds., Kluwer*, pages 115–133, 1995.

[201] R. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning*, 21:11–33, 1995.

[202] R. Parsons and M.E. Johnson. A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly. *American Journal of Mathematical and Management Sciences*, 17:369–396, 1995.

[203] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, pages 525–532. Morgan Kaufmann Publishers, San Francisco, CA, 1999. Orlando, FL.

[204] E. Islas Pérez, C. A. Coello Coello, and A. Hernández Aguirre. Extracting and re-using design patterns from genetic algorithms using case-based reasoning. *Engineering Optimization*, 35(2):121–141, April 2003.

[205] E. Serna Pérez. Diseño de Circuitos Lógicos Combinatorios utilizando Programación Genética. Master's thesis, Maestría en Inteligencia Artificial, Facultad de Física e Inteligencia Artificial, Universidad Veracruzana, Enero 2001. (In Spanish).

[206] P.A. Pevzner. *Computational molecular biology: An algorithmic approach*. The MIT Press, London, 2000.

[207] L.S. Pitsoulis and M.G.C. Resende. *Handbook of Applied Optimizatio*, chapter Greedy Randomized Adaptive Search procedure, pages 168–183. Oxford University Press, 2002.

[208] F. Pla, A. Molina, and N. Prieto. Tagging and chunking with bigrams. In *Proc. of the 17th conference on Computational linguistics*, pages 614–620, 2000.

[209] S.C. Porto and C.C. Ribeiro. Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. *Journal of Heuristics*, 1:207–223, 1995.

[210] J. C. Potts, T. D. Giddens, and S. B. Yadav. The development and evaluation of an improved genetic algorithm based on migration and artificial selection. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(1):73–86, 1994.

[211] W. Punch. How effective are multiple poplulations in genetic programming. In J.R. Koza et al., editor, *Genetic Programming 1998: Proc. of the Third Annual Conference*, pages 308–313. Morgan Kaufmann, 1998.

[212] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62(9):627–631, 1955.

[213] N. J. Radcliffe and P. D. Surry. The reproductive plan language RPL2: Motivation, architecture and applications. In J. Stender, E. Hillebrand, and J. Kingdon, editors, *Genetic Algorithms in Optimisation, Simulation and Modelling*. IOS Press, 1999.

[214] M. Rahoual, R. Hadji, and V. Bachelet. Parallel Ant System for the Set Covering Problem. In M. Dorigo et al., editor, *3rd Intl. Workshop on Ant Algorithms*, LNCS 2463, pages 262–267. Springer-Verlag, 2002.

[215] M. Randall and A. Lewis. A Parallel Implementation of Ant Colony Optimization. *Journal of Parallel and Distributed Computing*, 62(9):1421 – 1432, 2002.

[216] R.L. Rardin and R. Uzsoy. Experimental Evaluation of Heuristic Optimization Algorihtms: A Tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[217] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

[218] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, UK, 1993.

[219] C.R. Reeves. *Modern Heuristics Search Techniques for Cominatorial Problems*. Blackwell Scientific Publications, 1993.

[220] G. Reinelt. TSPLIB - A travelling salesman problem library. *ORSA - Jorunal of Computing*, 3:376–384, 1991.

[221] J. L. Ribeiro-Filho, C. Alippi, and P. Treleaven. Genetic algorithm programming environments. In J. Stender, editor, *Parallel Genetic Algorithms: Theory and Applications*, pages 65–83. IOS Press, 1993.

[222] G. Robbins. EnGENEer - The evolution of solutions. In *Proc. of the fifth Annual Seminar Neural Networks and Genetic Algorithms*, 1992.

[223] D. Roberts and R. Johnson. Evolving Frameworks. A Pattern Language for Developping Object-Oriented Frameworks. *To be published in Pattern Languages of Program Design 3 (PLoPD3)*, 2003.

[224] P. Roussel-Ragot and G. Dreyfus. A problem-independent parallel implementation of simulated annealing: Models and experiments. *IEEE Transactions on Computer-Aided Design*, 9:827–835, 1990.

[225] R.S.Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[226] G. Rudolph. Takeover Times in Spatially Structured Populations: Array and Ring. In K. K. Lai, O. Katai, M. Gen, and B. Lin, editors, *2nd Asia-Pacific Conference on Genetic Algorithms and Applications*, pages 144–151. Global-Link Publishing, 2000.

[227] G. Rudoplh. Global optimization by means of distributed evolution strategies. In H.P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, volume 496, pages 209–213, 1991.

[228] G. Sampson. *English for the Computer*. Clarendon Press, Oxford, 1995.

[229] E.F. Tjong Kim Sang. Memory-based shallow parsing. *J. Mach. Learn. Res.*, 2:559–594, 2002.

[230] J. Sarma and K. De Jong. An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms. In H. M. Voigt, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *PPSN IV*, volume 1141 of *LNCS*, pages 236–244. Springer, 1996.

[231] J. Sarma and K. De Jong. An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 181–186. Morgan Kaufmann, 1997.

[232] T. Sasao, editor. *Logic Synthesis and Optimization*. Kluwer Academic Press, 1993.

[233] M. Schütz and J. Sprave. Application of Parallel Mixed-Integer Evolution Strategies with Mutation Rate Pooling. In L.J. Fogel, P.J. Angeline, and T. Bäck, editors, *Proc. Fifth Annual Conf. Evolutionary Programming (EP'96)*, pages 345–354. The MIT Press, 1996.

[234] H. Schutze and Y. Singer. Part of speech tagging using a variable memory markov model. In *Proc. of the Association for Computational Linguistics*, 1994.

[235] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*, chapter 4 - Fragment Assembly of DNA, pages 105–139. University of Campinas, Brazil, 1997.

[236] A. Slowik and M. Bialko. Design and optimization of combinational digital circuits using modified evolutionary algorithm. In Leszek Rutkowski, Jörg H. Siekmann, Ryszard Tadeusiewicz, and Lotfi A. Zadeh, editors, *7th International Conference in Artificial Intelligence and Soft Computing - ICAISC 2004*, volume 3070 of *Lecture Notes in Computer Science*, pages 468–473, Zakopane, Poland, June 2004. Springer-Verlag.

[237] M. Soto, A. Ochoa, S. Acid, and L. M. de Campos. Introducing the polytree aproximation of distribution algorithm. In *Second Symposium on Artificial Intelligence. Adaptive Systems. CIMAF 99*, pages 360–367, 1999. La Habana.

[238] J. Sprave. Linear Neighborhood Evolution Strategies. In A.V. Sebald and L.J. Fogel, editors, *Proc. Third Annual Conf. Evolutionary Programming (EP'94)*, pages 42–51. World Scientific, Singapore, 1994.

[239] J. Sprave. A Unified Model of Non-Panmictic Population Structures in Evolutionary Algorithms. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 2, pages 1384–1391, Mayflower Hotel, Washington D.C., USA, July 1999. IEEE Press.

[240] J. Stender, editor. *Paralle Genetic Algorithms: Theory and Applications*. IOS Press, Amsterdam, The Netherlands, 1993.

[241] D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms*. The Charles Babbage Research Center, 2nd. edition, 1987.

[242] V.S. Sunderam. Pvm: a framework for parallel distributed computing. *Journal of Concurrency Practice and Experience*, 2(4):315–339, 1990.

[243] G.G. Sutton, O. White, M.D. Adams, and A.R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Tech.*, pages 9–19, 1995.

[244] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, 1996.

[245] T. Stützle. Parallelization Strategies for Ant Colony Optimization. In R. De Leone, A. Murli, P. Pardalos, and G. Toraldo, editors, *High Performance Algorithms and Software in Nonlinear Oprimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer, 1998.

[246] T. Stützle. Local search algorithms for combinatorial problems analysis, algorithms and new applications. Technical report, DISKI Dissertationen zur K¨unstliken Intelligenz. Sankt Augustin, Germany, 1999.

[247] E-G. Talbi, Z. Hafidi, D. Kebbal, and J-M. Geib. MARS: An adaptive parallel programming environment. In B. Rajkumar, editor, *High Prformance Cluster Computing, Vol. 1*, pages 722–739. Prentice-Hall, 1999.

[248] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel Ant Colonies for Combinatorial Optimization Problems. In Feitelson and Rudolph, editors, *Job Scheduling Strategies for Parallel Processing: IPPS'95 Workshop, LNCS 949*, volume 11. Springer, 1999.

[249] E.G. Talbi, Z. Hafidi, and J.M. Geib. A parallel adaptive tabu search approach. *Parallel Computing*, 24:2003–2019, 1996.

[250] R. Tanese. Distributed genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439. Morgan Kaufmann, 1989.

[251] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 299–335. John Wiley & Sons Inc., 2003.

[252] A. Töorn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1989.

[253] S. Tschöke and T. Polzer. Portable parallel branch-and-bound library, 1997. http://www.uni-paderborn.de/cs/ag-monien/SOFTWARE/PPBB/introduction.html.

[254] S. Tsutsui and Y. Fujimoto. Forking Genetic Algorithms with Blocking and Shrinking Modes. In *Proceeding of the 5th International Conference on Genetic Algorithms*, pages 206–213. Morgan Kaufmann, 1993.

[255] E.R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1993.

[256] V. K. Vassilev, T. C. Fogarty, and J. F. Miller. Information Characteristics and the Structure of Landscapes. *Evolutionary Computation*, 8(1):31–60, Spring 2000.

[257] V. K. Vassilev, J. F. Miller, and T. C. Fogarty. Digital Circuit Evolution and Fitness Landscapes. In *1999 Congress on Evolutionary Computation*, volume 2, pages 1299–1306, Washington, D.C., July 1999. IEEE Service Center.

[258] E. W. Veitch. A chart method for simplifying boolean functions. In *Proceedings of the ACM*, pages 127–133. IEEE Service Center, May 1952.

[259] S. Voss and D. L. Woodruff, editors. *Optimization Software Class Libraries*, volume 18 of *Operations Research and Computer Science Interfaces*. Kluwer Academic Publishers, Boston, 2002.

[260] M. Wall. GAlib: A C++ Library of Genetic Algorithm Components. URL: http://lancet.mit.edu/ga/.

[261] K. Weinert, J. Mehnen, and G. Rudolph. Dynamic neighborhood structures in parallel evolution strategies. *Complex Systems*, 2004 (to appear).

[262] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[263] D. Whitley and T. Starkweather. GENITOR II: A distributed genetic algorithm. *Journal of Experimental and Theoretical Aritificial Intelligence*, 2:189–214, 1990.

[264] J. Whittaker. *Graphical models in applied multivariate statistics.* John Wiley & Sons, Inc., 1990.

[265] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[266] Y. Davidor. A naturally occuring niche and species phenomenon: The model and first results. In R.K. Belew and L.B. Booker, editors, *Proc. of the Fourth Intl. Conf. Genetic Algorithms*, pages 257–263, 1991.

[267] X. Yao. A New SA Algorithm. *Intl. J. of Computer Mathematics*, 56:161–168, 1995.