Instituto Tecnológico Autónomo de México

# Algoritmos Genéticos Paralelos en Arquitecturas de Memoria Distribuida

**Erick Cantú Paz**

Ingeniería en Computación

# Contenido

# Índice de Figuras

# Índice de Tablas

*Capítulo 1*

# *Introducción*

Los algoritmos genéticos (AGs) son métodos de búsqueda basados en mecanismos de selección natural y genética poblacional que se han utilizado exitosamente en problemas de optimización, aprendizaje y programación automática [Gol89].

Los AGs convencionales (seriales) presentan una buena relación entre la calidad de sus resultados y el tiempo que necesitan para encontrarlos. Sin embargo, en aplicaciones muy complejas, el tiempo de ejecución es considerable. Una manera de mejorar significativamente el desempeño de los algoritmos genéticos es paralelizarlos.

La primera propuesta de algoritmos genéticos paralelos (AGPs) es de 1981 (por John Grefenstette [Gre81]), pero la investigación en esta área es escasa hasta el final de la década pasada. Este fenómeno se dio, según mi opinión, por dos causas principales. La primera es que es más difícil explicar formalmente el comportamiento de un AGP que el de un AG serial. El número de parámetros a considerar es mayor y se presentan fenómenos nuevos en los AGPs que no son explicables con la teoría existente. La otra causa probable de la escasez de trabajos es que el número de máquinas paralelas disponibles a los investigadores era muy pequeño en la primera mitad de la década por lo que tampoco hay muchos resultados experimentales.

Al paralelizar a los algoritmos genéticos para disminuir su tiempo de ejecución surge una pregunta importante: ¿serán capaces los algoritmos paralelos de mantener (o mejorar) la calidad de las soluciones de un AG serial? La respuesta parece ser afirmativa. En años recientes, se han llevado a cabo estudios que confirman la idea de que los AGPs son capaces de brindar soluciones significativamente mejores que las de los algoritmos seriales [Tan89][GoWh93]. Estos estudios basan sus conclusiones en resultados experimentales, no existe todavía la teoría necesaria para explicar satisfactoriamente el comportamiento de los AGPs y los resultados observados.

## 1.1. Campo de investigación

Los algoritmos genéticos pertenecen a un área de investigación conocida como *computación evolutiva.* En esta área se estudian algoritmos que operan sobre una población de estructuras aplicando sobre ellas ciertos operadores. Las estructuras se adaptan a cierto ambiente, que depende del problema específico que se intenta resolver, en una clara analogía al proceso natural de evolución. En el caso de los algoritmos genéticos, los operadores utilizados tienen una semejanza muy grande con los de genética natural (de ahí el nombre algoritmos *genéticos*).

Los algoritmos genéticos dominan la investigación en computación evolutiva [DeSp93] (ver figura 1.1) y actualmente la investigación se concentra en varios campos: *programación genética*, *sistemas clasificadores, teoría de AGs, AGPs y conexiones con otras áreas*. Programación genética es un campo dónde se intenta evolucionar programas que ejecuten las acciones deseadas sin tener que programarlos explícitamente [Koz92]. Los sistemas clasificadores imitan procesos de aprendizaje utilizando algoritmos genéticos para modificar reglas que se adaptan a cierto ambiente [Gol89]. Las conexiones de algoritmos genéticos con otras áreas incluyen diseño y entrenamiento de redes neuronales y aplicaciones en campos diversos [BeBo91][For93]. Los AGPs son un campo muy explorado a partir de 1987 cuando el número de publicaciones aumentó considerablemente. Actualmente hay un número muy importante de investigadores trabajando en esta área en todo el mundo y el número de trabajos es cada vez mayor.

La programación evolucionista (*Evolutionary programming*) [FOW66] y las estrategias evolutivas (*Evolutionstrategies*) [Bäc93] son algoritmos que han sido estudiados durante algún tiempo y han sido aplicados exitosamente en la resolución de algunos problemas. Las estructuras contenidas en sus poblaciones representan objetos muy específicos (máquinas de estado finito y números reales respectivamente) y por eso su aplicabilidad no es tan amplia como la de los AGs.

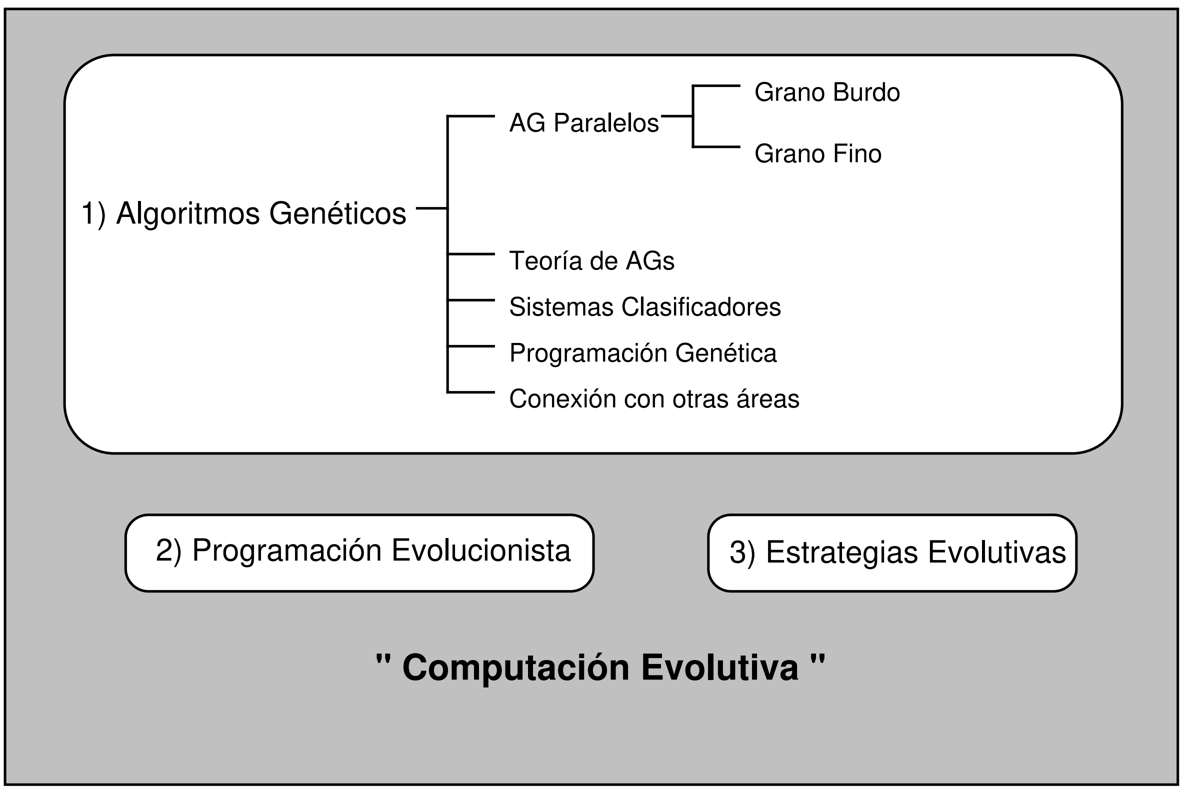



**Figura 1.1** Áreas de estudio de la *Computación Evolutiva.*

La investigación en AGPs seguramente tendrá repercusiones importantes en los demás campos de estudio de los algoritmos genéticos y en otras áreas de la computación evolutiva. Al paralelizar estos algoritmos, se pueden implementar naturalmente mecanismos como formación de especies, creación de nichos e interrupción del equilibrio de una población introduciendo material de otras [DeSp93] [1].

# 1.2. Motivación

En un algoritmo genético es necesario evaluar a todos los miembros de la población al principio de cada experimento. Durante la ejecución del algoritmo, sus operadores modifican las estructuras de la población creando nuevos miembros que deben ser evaluados antes de incorporarse a la población. Si el proceso de evaluación de cada individuo requiere de un tiempo considerable de procesamiento, el tiempo de respuesta del algoritmo será muy elevado. Para intentar mejorar el desempeño se propone paralelizar al algoritmo.

En este trabajo el algoritmo genético se paralelizará dividiendo la población en varias subpoblaciones, identificando periódicamente a los mejores individuos de cada subpoblación y comunicándolos entre ellas. Este modelo tiene asociados ciertos parámetros que controlan su funcionamiento. Los parámetros nuevos son:

- El número de subpoblaciones
- El tamaño de las subpoblaciones
- El número de individuos que migran
- La frecuencia con la que hay migraciones
- La dirección de las migraciones
- El criterio de selección de los migrantes

Algunos de estos parámetros han sido utilizados previamente en AGPs [Tan89][BiBr92], pero todavía no se comprende totalmente el efecto de estos parámetros en la calidad y eficiencia de la búsqueda de soluciones del algoritmo. Para poder observar el efecto de distintas combinaciones de parámetros sobre el AG es necesario tener una herramienta adecuada.

A pesar de que actualmente existen bastantes herramientas que permiten experimentar con AGs, hay pocas que permiten experimentar con AGPs. Los sistemas de AGPs existentes tienen algunos problemas que limitan su utilidad:

- Algunos sistemas están diseñados para explotar la arquitectura de una computadora específica, lo que los hace eficientes, pero limita su uso.
- Implementan un criterio fijo de migración, i.e. no permiten establecer diferentes frecuencias de migración, no se puede cambiar el número de individuos que migran, la dirección de las migraciones ni el criterio para seleccionar a los individuos que migran.

---

[1] En el capítulo tres se explican estos conceptos y la manera en que se implementan en un AGP.

- No están disponibles. El acceso a los sistemas está limitado al personal de los laboratorios en donde se crearon.

# 1.3. Objetivos

Los objetivos de esta tesis son:

1. Realizar un sistema que permita la experimentación con algoritmos genéticos paralelos en problemas de optimización de funciones *simulando* diferentes topologías de comunicación entre los procesos involucrados.

   Se pretende crear un sistema que resuelva algunos de los problemas descritos en la sección anterior. El sistema deberá ser eficiente (pero no dependiente de una arquitectura específica), flexible (permitiendo al usuario modificar los parámetros que controlan la migración) y debe estar disponible a las personas que lo requieran.

2. Realizar experimentos con el sistema que se desarrolle utilizando diferentes topologías y variando la frecuencia de comunicación y el número de individuos que se intercambian entre los procesos para tratar de determinar empíricamente el efecto de estos parámetros sobre la búsqueda de soluciones.

No es la intención de este trabajo resolver un problema específico con algoritmos genéticos. Para los experimentos que se realizarán con el sistema se utilizarán funciones utilizadas frecuentemente en la investigación de AGs y AGPs. El primer conjunto de prueba son las funciones de De Jong [DeJ75], que tradicionalmente se han usado como punto de referencia para probar AGs y para complementar este conjunto se usarán las funciones que Mühlenbein utilizó para probar su AGP [MSB91].

En esta tesis, tampoco se pretende dar una explicación teórica al comportamiento de los algoritmos genéticos paralelos en diferentes topologías.

# 1.4. Organización

El siguiente capítulo es una introducción a los algoritmos genéticos. En él se presentan algunas generalidades acerca de AGs, una breve descripción de la teoría utilizada frecuentemente para explicar su funcionamiento y así como algunas consideraciones de diseño de AGs. El capítulo tres presenta los diferentes tipos de algoritmos genéticos paralelos propuestos, una comparación entre los distintos modelos y la explicación de algunos fenómenos que se pueden observar en ellos.

El cuarto capítulo se dedica al diseño y desarrollo del software creado en esta tesis. En él se presentan los principales problemas que se encontraron en la implementación del sistema y los algoritmos que se utilizaron para solucionarlos. En el capítulo cinco se presenta el diseño de los experimentos y los resultados obtenidos en ellos; se presentan las funciones, las topologías y los parámetros utilizados en las pruebas y los criterios para calificar los experimentos.

En el último capítulo se presentan las conclusiones de este trabajo y las propuestas para extenderlo.

*Capítulo 2*

# *Algoritmos Genéticos*

Este capítulo introduce a los algoritmos genéticos: después de una motivación general se presenta su definición y se explican sus elementos. En seguida, se desarrollan los fundamentos matemáticos que respaldan su funcionamiento y se analizan algunos de los detalles que hay que considerar para su implementación. La intención de este capítulo es proporcionar un marco de referencia adecuado para el desarrollo de este trabajo.

## 2.1. Motivación

En la naturaleza, los individuos de cualquier especie están sometidos a una lucha continua por sobrevivir. Los recursos (alimento, agua, territorio) son limitados y deben repartirse entre los miembros de una población; esto origina una competencia por los recursos y los individuos más aptos tienen una mayor probabilidad de sobrevivir. La adaptación de cada individuo al medio, esto es, la forma en que responde a los estímulos de éste, favorece su supervivencia. Los más aptos tienen mayores probabilidades de reproducirse y transmitir a sus descendientes las características que los hacen superiores al resto de la población mientras que los individuos menos capaces forman una minoría o desaparecen.

La naturaleza tiene una manera de eliminar a los individuos (y eventualmente especies) que no se adaptan adecuadamente al medio y de favorecer a aquellos que se adaptan mejor. A este proceso se le conoce como selección natural [Daw86]. Los mecanismos exactos de la selección natural tienen aún muchos detalles desconocidos, pero la dinámica general es bien conocida.

Los algoritmos genéticos imitan este proceso de selección manipulando una población de individuos que se adaptan a las soluciones de un problema específico. En la naturaleza todas las especies evolucionan de manera similar. Análogamente, el funcionamiento del algoritmo genético es independiente del problema, sólo necesita información que le permita determinar la aptitud de los individuos con los que trabaja.

De esta manera, imitando los procesos naturales, se ha evitado uno de los mayores obstáculos en el desarrollo de software: especificar por adelantado todas las cualidades de un problema y todas las acciones que el programa debe efectuar para resolverlo [Hol92b]; sólo es necesario proveer al algoritmo genético de un método de evaluación para los individuos y dejarlo actuar sobre su población hasta encontrar una respuesta satisfactoria.

En términos generales, un problema se define por un espacio de búsqueda y por las características que deben cumplir ciertos valores dentro de este espacio para ser considerados soluciones. Sin embargo, la mayoría de las veces no basta con cualquier solución, sino que se debe encontrar la mejor solución posible dado un cierto criterio; esto es, se debe optimizar la solución [Rub75].

Existen muchos métodos de resolución de problemas, entre ellos:

- el cálculo diferencial,
- la programación lineal y entera,
- métodos de Inteligencia Artificial (búsquedas heurísticas),
- métodos enumerativos, etc.

Estos métodos permiten obtener buenas e incluso óptimas soluciones a los problemas que se les presentan. Basan su estrategia en ciertas características de los problemas, por ejemplo: que las funciones objetivo sean continuas y derivables, que no tengan ruido, que el espacio de búsqueda sea de un tamaño moderado, etc., y su aplicabilidad se restringe severamente ante problemas reales donde las funciones objetivo no tienen las características necesarias para aplicar el método. En cambio, los algoritmos genéticos requieren poca información sobre el problema específico y prácticamente no hacen suposiciones sobre las propiedades de la función objetivo ni del espacio de búsqueda. Por esto, pueden ser aplicados en dominios muy variados donde la función objetivo puede ser muy compleja (muchas variables), tener ruido, ser multimodal y donde el espacio de búsqueda sea inmenso.

Para evaluar un método de resolución de problemas, es necesario tomar en cuenta dos factores [Est92]:

1. La ***eficacia***: es la capacidad del método para encontrar una (o varias) solución(es) satisfactoria(s) al problema.
2. La ***eficiencia***: es una medida del costo de obtener la(s) solucion(es).

Para poder decidir acerca de utilizar un método sobre otro, hay que decidir si conviene tener resultados buenos con un costo bajo o resultados excelentes con un costo alto. Si la diferencia en la calidad de las soluciones no es importante, se desperdiciarán recursos si se elige un método costoso. Los algoritmos genéticos presentan una buena relación entre eficiencia y eficacia, encontrando buenas (y algunas veces incluso óptimas) soluciones en tiempos bastante razonables por lo que son muy atractivos para ser aplicados a una gran variedad de problemas.

## 2.2. Definición

Los ***algoritmos genéticos*** (AGs) son algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural [Gol89]. Operan sobre una población de individuos que representan posibles soluciones al problema, evaluándolos y escogiendo a los más aptos para reproducirse y transmitir sus características a las siguientes generaciones.

Los AGs actúan sobre los individuos de la población utilizando *operadores genéticos* para obtener a los descendientes de la población y formar con ellos una nueva generación. Utilizan la información histórica contenida en la población para guiar la búsqueda hacia regiones prometedoras del espacio de búsqueda. En las siguientes secciones (2.3 y 2.4) se profundiza en estos dos importantes aspectos.

Se dice que los algoritmos genéticos son *robustos*, dado que se pueden aplicar a una gran variedad de problemas manteniendo una buena relación entre eficacia y eficiencia [Est92][Gol89][Hol92b] y son aplicables a una amplísima variedad de problemas. Esta flexibilidad la obtienen por el hecho de que necesitan muy poca información sobre el problema específico.

El algoritmo genético básico se muestra en la figura 2.1.

```
Generar la población inicial
Evaluar a la población
Mientras los criterios de terminación no se cumplan
{
       Seleccionar a los individuos más aptos
       Aplicar operadores
       Evaluar nuevos individuos
       Insertarlos en la población según la técnica de reemplazo usada
}
```

**Figura 2.1** Algoritmo Genético Básico

# 2.3. Descripción de un AG

Los elementos de un algoritmo genético son:

1. Una *codificación de las soluciones de un problema*. Esto es una representación de los posibles valores de las variables que intervienen en el problema.

2. Una *técnica de arranque*, que se utiliza para producir la población inicial.

3. Una *función de evaluación* con la que se determina la bondad de los individuos de la población. La función de evaluación es la función objetivo que se desea optimizar, posiblemente escalada en cierto intervalo.

4. *Operadores genéticos*, que modifican la composición de la población.

5. Una *técnica de reemplazo*, que determina la interacción de los nuevos individuos y la población anterior.

6. Un(os) *criterio(s) de terminación* que especifica(n) cuándo se deja de operar sobre la población.

7. *Parámetros* que controlan el funcionamiento del AG, por ejemplo, el tamaño de la población, la probabilidad de cruce, de mutación, etc.

### 2.3.1. Codificación

Los valores posibles de las variables que intervienen en el problema se codifican en cadenas. Estas cadenas forman la población de individuos sobre la que trabajará el algoritmo y son conocidas como *cromosomas*[2]. Cada cromosoma está formado por *genes* y normalmente cada gen representa una variable.

En los cromosomas se pueden representar variables enteras o reales y cada gen puede tener una longitud arbitraria. La codificación de cada variable es independiente del resto. Realmente el algoritmo no conoce la forma de codificación que se eligió para un problema particular porque opera sobre las cadenas completas sin conocer divisiones entre genes. La decodificación de los cromosomas se deja a la función de evaluación provista para cada problema.

El trabajo original sobre algoritmos genéticos consideraba que los individuos eran cadenas binarias de 0s y 1s [Hol92a]. Para esta representación se han propuesto varios operadores genéticos y muchas de sus propiedades son bien conocidas.

Se han propuesto otras representaciones y los operadores correspondientes, pero sólo en casos aislados y por esto, sus propiedades son poco conocidas. En la sección 2.5.1 se analizan con detalle las codificaciones binaria y no binarias y las ventajas de utilizar la primera; por el momento se asumirá que la codificación es binaria.

### 2.3.2. Técnica de arranque

Para comenzar un AG normalmente se elige alguno de los siguientes procedimientos:

1. Se genera aleatoriamente a toda la población.

2. Se usan resultados de experimentos anteriores o resultados de otros métodos para formar la población inicial. De cualquier modo, se sugiere que una parte de la población sea generada aleatoriamente para asegurar que haya suficiente diversidad en los individuos y poder encontrar soluciones novedosas.

Con la segunda técnica se puede esperar que la *velocidad de convergencia* de la población sea mayor. Se dice que la población converge cuando los individuos tienden a parecerse unos a otros. La velocidad de convergencia se refiere a la rapidez con que esto ocurre. Si se insertan en la población original estructuras de las que se sabe que tienen valores cercanos a la solución se

---

[2] Los términos *estructura, individuo* y *cromosoma* se usan indistintamente en este trabajo, aunque estrictamente no quieren decir lo mismo. "Estructura" es el término general que fue utilizado originalmente para designar a los componentes de la población de un algoritmo adaptativo [Hol92a]. Un "individuo" es un componente de la población y en teoría puede estar compuesto de varios "cromosomas" [Gol89]. Sin embargo, los AGs normalmente tienen individuos con un sólo cromosoma.

espera que sean seleccionadas con mayor probabilidad para reproducirse y generar más estructuras buenas que desplacen a aquellas que están alejadas de la solución. Así la población tenderá rápidamente a parecerse a estructuras buenas.

El costo que se paga por tener una mayor velocidad de convergencia es que se exploran menos regiones del espacio y posiblemente no se encuentre el óptimo global.

### *2.3.3. Función de evaluación*

En la naturaleza, la aptitud se determina por la habilidad de un organismo para sobrevivir a obstáculos que le impiden llegar a la madurez y reproducirse (e.g. cambios en el clima, depredadores y falta de alimentos). En un sistema adaptativo artificial como los algoritmos genéticos, la función de evaluación de alguna manera representa el medio ambiente al que cada individuo se enfrenta [Hol92a].

En el caso más simple, la función de evaluación es la función objetivo que se desea optimizar. El AG debe encontrar a los individuos que representan valores óptimos. Este es el **único** componente del AG dependiente del problema que se está resolviendo; basta con cambiar la función de evaluación para cambiar el dominio del problema. El algoritmo desconoce el proceso de evaluación y sólo utiliza la información cuantitativa provista por esta función como medida de bondad de los individuos.

Antes de poder evaluar a un individuo es necesario decodificar la información que contiene para encontrar el valor de las variables que se utilizan como entradas a la función de evaluación. Como ya se explicó la codificación es arbitraria por lo que cada función de evaluación debe tener la capacidad de extraer la información de los cromosomas.

Haciendo la analogía con genética natural, el individuo que manipula el algoritmo genético (la cadena con la información codificada) es el *genotipo* y la interpretación del genotipo que se evalúa es el *fenotipo* (ver figura 2.2). El medio ambiente (o la función de evaluación) mide la aptitud de un individuo desarrollado, no mide la aptitud de sus genes.

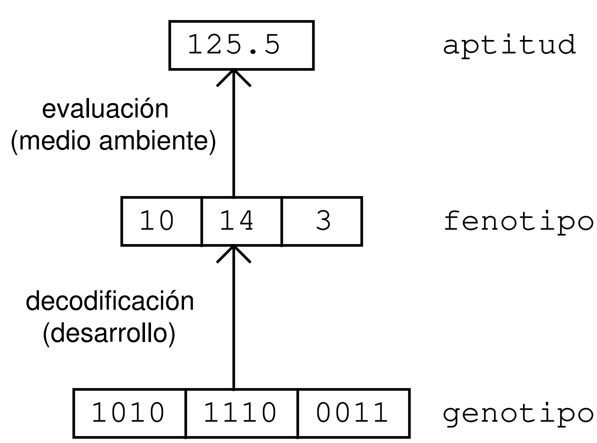



**Figura 2.2** Genotipos y fenotipos

## 2.3.4. Operadores genéticos

Los operadores genéticos son los medios mediante los cuales el algoritmo modifica a su población. Un algoritmo genético básico tiene 3 operadores: selección, cruce simple y mutación, que han sido estudiados durante bastante tiempo y sus propiedades se comprenden relativamente bien. Sin embargo, existen otros operadores que son aún poco comprendidos y su eficacia es todavía dudosa. Por eso en esta explicación se hace énfasis sobre los operadores básicos y sólo se mencionan brevemente los demás.

### *2.3.4.1. Selección*

La selección es el proceso mediante el cual se copian individuos de la totalidad de la población a un conjunto intermedio de donde se derivará la nueva población aplicando otros operadores (cruce y/o mutación). La selección de cada individuo es dependiente de su aptitud, que se mide de acuerdo al valor de la función de evaluación. Entre mejor sea el resultado de la evaluación de un individuo, tendrá mayor oportunidad de ser seleccionado.

Este operador es una versión artificial de la *selección natural:* se determinan los individuos más aptos entre la población de cadenas del algoritmo. Este proceso es crucial para el algoritmo, porque de esta manera se asegura que conforme pasa el tiempo el algoritmo avanza hacia las soluciones dejando atrás los puntos en el espacio de búsqueda que no son buenos.

La selección puede implementarse en muchas formas [deTi93], pero quizás la más común sea una ruleta donde cada individuo tiene una ranura proporcional a su aptitud (de ahí el nombre selección proporcional). La selección se hace simulando un giro de la ruleta. Los mejores individuos tienen una ranura mayor que los menos aptos y por lo tanto tienen mayor probabilidad de ser seleccionados.

### *2.3.4.2. Cruce Simple*

El cruce simple consiste en tomar aleatoriamente dos individuos del conjunto intermedio de estructuras seleccionadas por su aptitud, escoger al azar un punto de cruce e intercambiar el material entre los individuos como se puede ver en la figura 2.3. Aunque podría parecer que no es probable encontrar buenas estructuras haciendo el cruce al azar, este mecanismo funciona bastante bien. La explicación formal se da en la sección 2.4, pero intuitivamente hay que pensar que se están combinando estructuras buenas y que algunas veces se obtienen estructuras mejores.

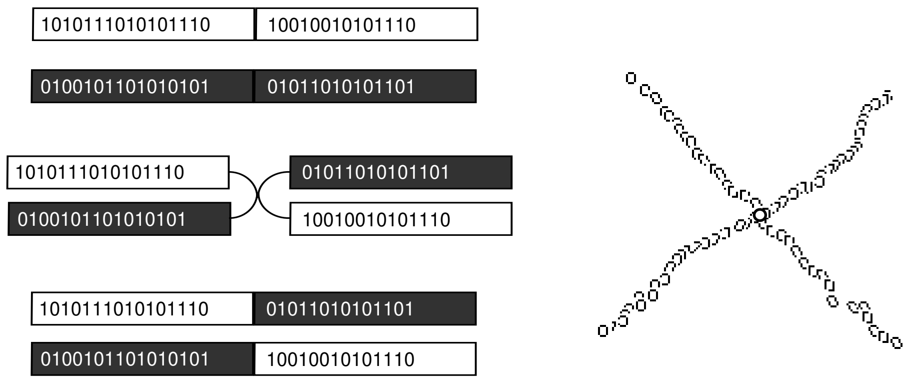



**Figura 2.3** Intercambio de información en cruce simple

Varios investigadores en la década de los setenta propusieron operadores de cruce en múltiples puntos. El más sobresaliente de ellos fue Kenneth De Jong quién notó que el desempeño del algoritmo se degradaba conforme aumenta el número de puntos de cruce [DeJ75]. Intuitivamente es fácil ver que si las estructuras se mezclan demasiado es muy difícil mantener la información histórica contenida en la población que guía la búsqueda. El teorema que se presenta la sección 2.4 expresa formalmente esta degradación. Sin embargo, algunos investigadores han encontrado buenos resultados al utilizar cruce uniforme, esto es, cruzar a los individuos en todos los puntos del cromosoma [Sys89].

#### *2.3.4.3. Mutación*

El efecto de los operadores de selección y cruce es una búsqueda efectiva y eficiente en el espacio de búsqueda definido para el problema. La mutación es necesaria porque ocasionalmente la selección y el cruce sobreexplotan regiones del espacio de búsqueda que han sido identificadas como buenas y pierden material genético que es potencialmente útil. En los AGs, el operador de mutación protege contra la pérdida irreparable de material genético [Gol89].

La mutación consiste en cambiar aleatoriamente el contenido de algún bit de un cromosoma. Cada bit es considerado separadamente, y se cambia su valor aleatoriamente con la probabilidad especificada por el valor del parámetro *probabilidad de mutación*.

La mutación es un operador secundario en un AG. Como en la naturaleza, la probabilidad de aplicar este operador es muy baja. Normalmente se usan probabilidades alrededor de 0.001, lo que quiere decir que se modifica uno de cada 1000 bits de la población.

No se puede depender exclusivamente de la mutación para encontrar la solución a un problema, porque sería un proceso lentísimo. La población evoluciona mucho más rápido recombinando individuos usando el operador de cruce que copiando el material genético de la generación anterior modificado eventualmente por mutación. Parece ser que en la naturaleza ocurre lo mismo.

### ***2.3.5. Técnica de reemplazo***

La técnica de reemplazo determina la relación entre una generación y la siguiente. En el caso más simple, la población es reemplazada completamente por sus descendientes en cada generación. En la naturaleza, los padres de una generación no mueren al nacer sus descendientes, sino que siguen siendo considerados para tener descendencia. De Jong [DeJ75] introdujo un parámetro llamado *brecha generacional* (generation gap) que permite establecer la proporción de la población que se reemplaza en cada generación.

Otra modificación que se hace al algoritmo genético simple para mejorar su desempeño, es asegurarse de que el mejor individuo de la generación anterior sea seleccionado para estar presente en la nueva generación. Si después de generar la nueva población, no se encuentra a este individuo, se le agrega sustituyendo a otro arbitrariamente o según alguna política definida con anterioridad (e.g. se elimina al peor individuo). A la conservación del mejor individuo de la generación anterior se le conoce como el modelo *elitista* de reemplazo y asegura la convergencia del algoritmo [EAV91].

## 2.3.6. Criterios de terminación

La terminación de un algoritmo genético puede especificarse por tres criterios principales:

1. Se alcanza un número máximo de generaciones. El propósito de esto es establecer una cota máxima en el tiempo que se dedica a solucionar un problema.

   En muchas ocasiones se usa como criterio de terminación un número máximo de *evaluaciones*. Esto permite especificar más directamente el límite de tiempo de CPU que se desea dedicar a un problema porque el número de evaluaciones por generación no es constante. Normalmente sólo se evalúan a los individuos nuevos creados por los operadores genéticos, no a toda la población. El número de nuevos individuos producidos en cada generación depende de las probabilidades de aplicación de los operadores genéticos y de la diversidad de la población (que cambia en el tiempo).

2. La población llega a un nivel mínimo de diversidad. Esto es equivalente a decir que la población converge. Si no hay mucha diversidad en la población es muy difícil que el algoritmo explore nuevos puntos en el espacio de búsqueda y muy probablemente no conviene seguir.

   La diversidad se puede medir contando el número de posiciones en todos los individuos de la población que tienen el mismo valor.

3. Se obtiene un valor satisfactorio en la función objetivo.

## 2.3.7. Parámetros

Los parámetros de un algoritmo genético controlan su funcionamiento en varios aspectos.

Los operadores genéticos se aplican a los individuos que han sido seleccionados como progenitores de la siguiente generación, pero no se aplican a todo este conjunto. Se deben especificar las probabilidades de aplicación de los operadores. Normalmente, el operador de cruce tiene una probabilidad de aplicación ($P_c$) que generalmente es de 60 o 70% (0.6 - 0.7). Esto significa que una vez seleccionada una pareja de estructuras para ser cruzadas, hay una probabilidad de 30-40% de que no se crucen y de que pasen inalteradas a la siguiente generación. El operador de mutación usualmente tiene una probabilidad ($P_m$) mucho menor de ser aplicado, algo así como 0.001. Además, se debe seleccionar el tamaño de la población, un criterio de reemplazo y, si existe, la brecha generacional.

Seleccionar los parámetros adecuados para un AG es probablemente la tarea más difícil para su uso. Los mejores parámetros en un dominio pueden ser ineficientes en otro, pero es posible encontrar un conjunto de parámetros robusto para gran cantidad de problemas.

La selección de los parámetros se puede hacer de cuatro maneras básicas:

1. Experimentalmente. Probar diferentes conjuntos de parámetros hasta encontrar uno que funcione adecuadamente. Se usa un problema muestra del dominio específico para encontrar los parámetros y después se usa el algoritmo sobre el resto de los problemas.

2. Usando los parámetros de otros estudios. Se han aplicado AGs con éxito a una gran variedad de problemas y es posible encontrar en la bibliografía experiencias de investigadores que reportan resultados de experimentos que pueden ser similares a los que se quieren realizar.

3. Haciendo una búsqueda exhaustiva discretizando los parámetros. Inicialmente se pueden utilizar granularidades grandes en la discretización para abarcar un área grande del espacio de búsqueda e identificar las regiones mas prometedoras que se deben explorar con granularidades mas finas.

4. Utilizando un AG para guíar la búsqueda. Esto significa que existe un meta algoritmo cuyos individuos representen los parámetros para controlar a un AG [Wei70].

Algunos investigadores proponen cambiar los valores de los parámetros dinámicamente. Por ejemplo si la población converge muy rápidamente se puede incrementar la probabilidad de mutación o cambiar el número de puntos de cruce que se están utilizando. Los parámetros pueden ser controlados centralmente por el AG [Cav70] o pueden estar codificados en los cromosomas junto con las variables del problema [Ros67].

Al modificar los parámetros de un AG se modifican al mismo tiempo su eficiencia y su eficacia. Por ejemplo, es claro que si se utiliza una población grande se necesitarán (manteniendo los demás parámetros fijos) más evaluaciones que con una población pequeña, así disminuye la eficiencia, pero en un buen número de problemas se aumenta la eficacia porque es más probable encontrar buenas soluciones en una población que tiene mayor diversidad. Si se usa una población pequeña, se presenta el problema de que converja muy rápidamente sin dar oportunidad al algoritmo de explorar muchas regiones del espacio de búsqueda.

En 1975, Kenneth De Jong hizo un estudio muy completo [DeJ75] acerca de la influencia de los parámetros en la calidad de búsqueda del algoritmo y en su eficiencia según el tipo de funciones que se desean optimizar.

# 2.4. Fundamentos matemáticos

En esta sección se presentan los fundamentos teóricos utilizados más frecuentemente para explicar el funcionamiento de un AG [Gol89].

### *2.4.1. Plantillas de similitud o esquemas*

En la descripción anterior de un AG se ha ignorado una cuestión fundamental: ¿Qué información está contenida en la población de cadenas y en los valores de la función de evaluación que guíe la búsqueda hacia una solución óptima?

Supongamos que los siguientes datos son parte de un algoritmo genético:

| cadena | valor de la función de evaluación |
|---|---|
| 01101 | 169 |
| 11000 | 576 |
| 01000 | 64 |
| 10011 | 361 |

Si se observa con cuidado se pueden encontrar ciertas similitudes entre las cadenas y si estas similitudes se exploran con mayor profundidad se pueden observar ciertas correlaciones entre las cadenas y los valores de la función de evaluación. Por ejemplo, en esta muestra las cadenas que empiezan con 1 tienen valores más altos que las que empiezan con 0.

La similitud entre cadenas con valores altos puede servir para guiar una búsqueda. Para formalizar la noción de similitud se usa el concepto de esquema. Un *esquema* es una plantilla que describe un conjunto de cadenas con similitudes en ciertas posiciones.

En esta explicación se considerarán únicamente esquemas que describan cadenas binarias construidas con el alfabeto {0, 1}. Para construir esquemas sobre este alfabeto, se agrega el símbolo * para decir que una posición no está especificada. Así, una cadena binaria pertenece al conjunto $C$ denotado por el esquema $H$ si en las posiciones donde aparece 0 en el patrón aparece 0 en la cadena y donde hay un 1 en el patrón hay un 1 en la cadena, no importa qué símbolo está en la cadena si en el esquema hay un *. Por ejemplo el esquema $H$ = **1010 representa al conjunto de 4 cadenas $C$ = {001010, 011010, 101010, 111010} y el esquema $H$ = 1111* representa dos cadenas $C$ = {11110, 11111}.

En los ejemplos anteriores se utilizó una longitud de cadena $l$ = 5. El número total de esquemas posibles es de $3^5$ = 243 porque en cada posición del esquema puede haber un 1, un 0 o un *. El número de esquemas posibles para cadenas binarias de longitud $l$ es de $3^l$ y en general para cadenas sobre un alfabeto de $k$ símbolos el número posible de esquemas es $(k+1)^l$.

En la tabla anterior es difícil entender la relación entre las cadenas y sus valores si no se toma en cuenta la similitud entre las mejores cadenas (las que tienen mejores valores de evaluación). La cuestión ahora es ¿cuánta información nueva se tiene considerando los esquemas? La respuesta a esto está relacionada con el número de esquemas *diferentes* contenidos en la población.

¿Cuántos esquemas están representados en una población de $n$ cadenas? Una cadena de longitud $l$ representa a $2^l$ esquemas porque cada posición de la cadena se puede sustituir o no por un símbolo *. Por lo tanto, $n$ cromosomas representan entre $2^l$ y $n \cdot 2^l$ esquemas dependiendo de la diversidad de la población.

## *2.4.2. Explotación de esquemas*

Para comprender como los esquemas guían la búsqueda de un algoritmo genético hay que entender el efecto de los operadores genéticos sobre la tasa de crecimiento de los representantes de un esquema. Se analizará el efecto de los operadores individualmente y en conjunto sobre los esquemas contenidos en una población de cadenas.

No todos los esquemas son iguales. Algunos son más específicos que otros. Por ejemplo el esquema 010*0** define mayores similitudes entre cadenas que el esquema 0******. Hay esquemas que se extienden más sobre la longitud de la cadena que otros. Por ejemplo, el esquema 1*****1 se extiende sobre una porción mayor de la cadena que el esquema ***1*1*. Para cuantificar estos conceptos, se introducen dos propiedades de los esquemas: el orden del esquema y la longitud de definición.

El *orden* de un esquema *H*, denotado por *o*(*H*), es el número de posiciones fijas en el esquema. Considerando los ejemplos anteriores, el orden del esquema 010*0** es 4 (simbólicamente: *o*(010*0**)=4 ) y el orden del esquema 0****** es 1.

La *longitud de definición* de un esquema *H*, denotada por δ(*H*), es la distancia entre la primera y la última posiciones fijas en el patrón. Por ejemplo, la longitud de definición de un esquema H = 1*****1 es 6 y la del esquema ***1*1* es 2.

### *2.4.2.1. Selección.*

Consideremos primero el efecto del operador de selección. Supongamos que en un tiempo *t* existen *m* representantes de un esquema particular *H* en la población ***A***(*t*), esto se representa como *m*=*m*(*H,t*). Durante la selección una cadena es copiada con una probabilidad proporcional a su aptitud. Más precisamente, la probabilidad de que una cadena $A_i$ sea seleccionada para ser progenitora de la nueva población es:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

donde $f_i$ es el valor de la función de evaluación para el elemento $A_i$ y *n* es el tamaño de la población. Al generar la nueva población, se espera tener *m*(*H, t*+1) representantes del esquema *H* en el tiempo *t*+1 dado por la ecuación:

$$m(H,t+1) = m(H,t) \cdot n \cdot \frac{f(H)}{\sum_{j=1}^{n} f_j}$$

donde f(H) es el promedio de las evaluaciones de las cadenas que representan al esquema *H* en el tiempo *t*. Si tomamos en cuenta que el promedio de aptitud de la población se puede representar como

$$\bar{f} = \frac{\sum_{j=1}^{n} f_j}{n}$$

entonces se puede reescribir la ecuación que describe la tasa de crecimiento de los esquemas en la selección como sigue:

$$m(H,t+1)=m(H,t)\frac{f(H)}{\bar{f}}$$

Lo que quiere decir que un esquema particular crece según la razón entre el promedio de aptitud del esquema y el promedio de aptitud de la población. En otras palabras, los esquemas que tengan un promedio de aptitud superior al de la población recibirán un número creciente de muestras en la siguiente generación, mientras que los esquemas con promedios de aptitud inferiores al promedio recibirán un número decreciente de muestras. Hay que resaltar que este comportamiento ocurre en paralelo para todos los esquemas *H* representados en la población. A esto se le da el nombre de *paralelismo implícito* y es una de las cualidades que hacen de los algoritmos genéticos excelentes métodos de búsqueda.

Si suponemos que la aptitud de un esquema particular *H* permanece sobre el promedio de la población $c\bar{f}$ veces (para una constante *c*) se puede reescribir la ecuación anterior como:

$$m(H,t+1)=m(H,t)\frac{(\bar{f}+c\bar{f})}{\bar{f}}=(1+c)\cdot m(H,t)$$

empezando en *t*=0 y asumiendo un valor constante de *c*, obtenemos la ecuación:

$$m(H,t)=m(H,0)\cdot(1+c)^{t}$$

Esta ecuación describe una progresión geométrica, el equivalente discreto de una función exponencial. Intuitivamente, lo que esta ecuación representa es que cuando un esquema positivo es descubierto, el número de sus representantes crece exponencialmente. El algoritmo explota este esquema asignándole un número exponencialmente creciente de evaluaciones a través del tiempo.

Hay que observar que el operador de selección por sí sólo no hace nada para explorar nuevas regiones en el espacio de búsqueda ya que no se buscan nuevos puntos. Si sólo se copian las estructuras sin hacerles ningún cambio es imposible encontrar soluciones novedosas. Para introducir cambios en la población se introducen los operadores de cruce y de mutación y su efecto sobre los esquemas se explica a continuación.

### *2.4.2.2. Cruce*

El cruce es un cambio de información estructurado, pero aleatorio, entre cadenas. Nuevas estructuras son creadas por este operador con una perturbación mínima del efecto de asignación exponencial de evaluaciones por la selección explicada en la sección anterior.

Para ver como son afectados los esquemas con el operador de cruce consideremos primero un ejemplo: supongamos una cadena *A* con longitud *l* = 7 y dos esquemas representativos:

| | |
|---|---|
| *A* | 0111000 |
| $H_1$ | *1****0 |

| | |
|---|---|
| $H_2$ | ***10** |

Si la cadena *A* es elegida para participar en un cruce entonces se seleccionará una posición para hacer el cruce. Podemos observar que el esquema $H_1$ puede ser destruido por el cruce si la posición en dónde se cortará la cadena está entre el 1 y el 0 (son 5 posibilidades). En total, hay 6 posibles lugares en donde se puede cortar una cadena con longitud 7. El esquema $H_2$ sólo puede ser destruido si el cruce se hace entre la cuarta y la quinta posición, en otras palabras la probabilidad de que sea destruido es de 1/6, bastante buena comparada contra la posibilidad de que $H_1$ sea destruido = 5/6.

Un esquema sobrevive el cruce si 1) este ocurre fuera de su longitud de definición o 2) por casualidad el cruce se realiza con un individuo que aporta la otra parte del esquema.

Formalmente, un esquema *H* es destruido en un cruce con probabilidad

$$p_d \leq \frac{\delta(H)}{l-1}$$

por lo tanto, la probabilidad de que un esquema sobreviva un cruce es $p_s = 1 - p_d$. Si el cruce se realiza con probabilidad $p_c$, entonces la probabilidad de supervivencia de un esquema *H* es:

$$p_s \geq 1 - p_c \cdot \frac{\delta(H)}{l-1}$$

Ahora podemos considerar el efecto combinado de selección y cruce. De la misma manera que cuando consideramos la selección únicamente, estamos interesados en calcular el número esperado de representantes de un esquema particular *H* en la siguiente generación. Así tenemos:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[ 1 - p_c \cdot \frac{d(H)}{l-1} \right]$$

Si comparamos esta ecuación con la que se obtuvo para selección, se puede observar que el efecto combinado de selección y cruce se obtiene multiplicando el número esperado de representantes del esquema en la selección por la probabilidad de supervivencia en el cruce. Con esto podemos observar que los representantes del esquema *H* crecen o decrecen por dos factores: 1) la relación del promedio de las evaluaciones de los representantes del esquema con el promedio de las evaluaciones de la población y 2) la longitud de definición del esquema. Los esquemas con promedio superior al de la población y con longitudes de definición cortas serán muestreados a tasas exponencialmente crecientes. A estos esquemas se les conoce como *bloques constructivos*.

### *2.4.2.3. Mutación*

La mutación es la alteración aleatoria de una posición con una probabilidad $p_m$. Para que un esquema *H* sobreviva la mutación todas sus posiciones definidas deben sobrevivir. Una posición sobrevive la mutación con una probabilidad de $1 - p_m$ y tomando en cuenta que las mutaciones son estadísticamente independientes, un esquema *H* sobrevive la mutación cuando sus *o(H)* posiciones fijas sobreviven. Multiplicando la probabilidad de supervivencia $(1 - p_m)$ por sí misma *o(H)* veces obtenemos que la probabilidad de que un esquema sobreviva la mutación es $(1 - p_m)^{o(H)}$. Para valores pequeños de $p_m$, la probabilidad de supervivencia de un esquema se puede aproximar[3] por la expresión $1 - o(H) \cdot p_m$ .

Así, el efecto combinado de selección, cruce y mutación sobre el número esperado de representantes de un esquema H es [4]:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[ 1 - p_c \cdot \frac{d(H)}{l-1} - o(H) \cdot p_m \right]$$

A este resultado se le conoce como el Teorema de Esquemas o como el *Teorema Fundamental de los Algoritmos Genéticos* [Gol89].

# 2.5. Implementación de un AG

En esta sección se presentan algunos aspectos importantes para el diseño e implementación de un algoritmo genético. Primero se tratan aspectos de la codificación, después se discute el escalamiento de aptitud y finalmente se explica un método para optimizar funciones con restricciones.

## 2.5.1. Codificación

Los algoritmos genéticos explotan las similitudes entre individuos aptos de una manera óptima cuando se encuentran bloques constructivos que guían la búsqueda hacia los puntos solución. No es un problema grave codificar un problema para ser resuelto por un algoritmo genético por la robustez que estos presentan. Es posible imaginar muchas codificaciones para el mismo problema sin preocuparnos por encontrar la mejor y el AG será capaz de encontrar buenas soluciones.

Sin embargo, sí existen algunas codificaciones que son mejores que otras y podemos observar dos principios para guiar las decisiones de codificación:

1. Se deben escoger codificaciones de manera que esquemas cortos y de bajo orden sean significativos al problema y relativamente distintos a otros esquemas sobre las mismas posiciones fijas. Esto es casi un arte (con un poco de suerte) [Gol89], pero no es demasiado difícil prestar atención a la distancia entre posiciones relacionadas; si disminuimos esta distancia, aumenta la posibilidad de que el AG no destruya los esquemas correspondientes.

---

[3] $(1 - p_m)^n = \sum_{i=0}^{n} C_i^n 1^{n-i} (-p_m)^i = 1 - o(H) \cdot p_m$ despreciando los términos de orden superior o igual a 2

[4] En esta ecuación se desprecia el término $p_c \frac{\partial(H)}{l-1} o(H) p_m$, porque se supone que $p_c p_m << 1$

2. El segundo principio es simplemente elegir el menor alfabeto posible que permita una representación natural del problema.

Hasta ahora se han utilizado alfabetos binarios para codificar las soluciones de un problema y no ha sido por casualidad: es más fácil encontrar similitudes en cadenas binarias que en cadenas no binarias. Esto se ejemplifica en la tabla 2.1:

| Cadena Binaria | Valor de la variable | Cadena no Binaria | Aptitud |
|---|---|---|---|
| 01101 | 13 | M | 169 |
| 11000 | 24 | X | 576 |
| 01000 | 8 | H | 64 |
| 10011 | 19 | S | 361 |

**Tabla 2.1**. Comparación de cadenas binarias y no binarias.

Es muy difícil encontrar una relación entre las letras de la cadena no binaria y la aptitud. Esto se debe principalmente a que no tenemos una muestra grande de la población que nos permita relacionar algunas letras con valores óptimos y además, con el propósito de hacer más claro el ejemplo, se escogió un alfabeto con una cardinalidad alta, con lo que se dificulta encontrar relaciones.

Formalmente, lo que sucede es que un alfabeto binario contiene el mayor número posible de esquemas en una codificación. Para representar el mismo número de puntos en cada espacio de búsqueda se requieren $2^l = k^m$ estructuras, dónde $l$ y $m$ son las longitudes de las estructuras binarias y no binarias respectivamente y $k$ es el número de símbolos en un alfabeto. Si se despeja $l$ de la ecuación anterior, es posible mostrar que la codificación binaria permite tener un número mayor de esquemas por unidad de información que cualquier otra:

$$l = m\frac{\log(k)}{\log 2}, \quad k \geq 3$$

El número de esquemas está determinado por $3^l$ en el caso de una codificación binaria y por $(k+1)^m$ en caso contrario. La ecuación anterior implica que $l$ será mayor que $m$ por un factor que depende de la cardinalidad $k$ del alfabeto; y siempre se cumple que $3^l$ es mayor que $(k+1)^m$ . Si la búsqueda de un AG se basa en las similitudes que pueda explorar entre su población, entonces tiene mucho sentido escoger la codificación que maximice el número de esquemas.

Con los dos principios expuestos anteriormente se pueden construir muchas codificaciones que funcionen exitosamente en la resolución de un problema. Una de las codificaciones más usadas en la resolución de problemas con varias variables reales se describe a continuación.

En primer lugar hay que decidir la precisión que se desea para representar cada variable. Esto no es ningún problema con los parámetros enteros ya que simplemente se codifican con el menor número posible de bits necesarios para representar el rango de valores que pueden tomar. Sin embargo, para codificar números reales normalmente lo que se hace es un mapeo lineal del número entero representado en la cadena de bits al rango [*Umin*, *Umax*] de la variable. Es fácil ver que la precisión de la variable está dada por:

$$\pi = \frac{Umax - Umin}{2^l - 1}$$

Así, el número de bits necesarios para representar una variable real con una precisión específica π es:

$$l = \log_2\left(\frac{Umax - Umin}{p} + 1\right)$$

Para codificar un problema de varias variables, simplemente se concatenan las codificaciones de cada una. Cada variable puede tener su propio rango de valores y la longitud de sus codificaciones puede ser diferente.

### *2.5.2. Escalamiento de aptitud*

El escalamiento de aptitud ocurre entre la producción de nuevos individuos y el uso de sus valores de aptitud para selección. Es un método para regular la frecuencia con la que los individuos son seleccionados para reproducirse. Al principio de la ejecución es común tener algunos individuos extraordinarios que sobresalen en la población de individuos mediocres. Si no se escala la aptitud, los individuos extraordinarios dominarán rápidamente sobre el resto de la población en muy pocas generaciones (convergencia prematura) y esto es indeseable porque elimina la posibilidad de explorar nuevas regiones en el espacio. Más tarde en la ejecución puede existir aún bastante diversidad en la población, pero la aptitud promedio posiblemente está muy cercana a la aptitud de los mejores individuos y la supervivencia del más apto se vuelve una búsqueda aleatoria.

Escalar la aptitud puede ayudar en los dos casos. Al principio del experimento disminuye la enorme diferencia entre las mejores cadenas y el resto de la población dando oportunidad a explorar mejor el espacio. Al final sirve para resaltar las pequeñas diferencias entre los mejores y el promedio permitiendo que la población converja.

La forma más sencilla de obtener la aptitud escalada $f'$ de un individuo es obtenerla de su aptitud "bruta" $f$ utilizando la siguiente relación lineal:

$$f' = af + b$$

dónde $a$ y $b$ se escogen de manera que:

1. El promedio de las aptitudes escaladas sea igual al promedio de las aptitudes sin escalar. Así se asegura que el número de copias de individuos con aptitud cercana al promedio se mantenga igual.

2. La aptitud escalada máxima sea un múltiplo de la aptitud promedio para permitir que las cadenas con aptitud mayor que el promedio se reproduzcan un número mayor de veces. El múltiplo es generalmente 2 (ver figura 2.4).

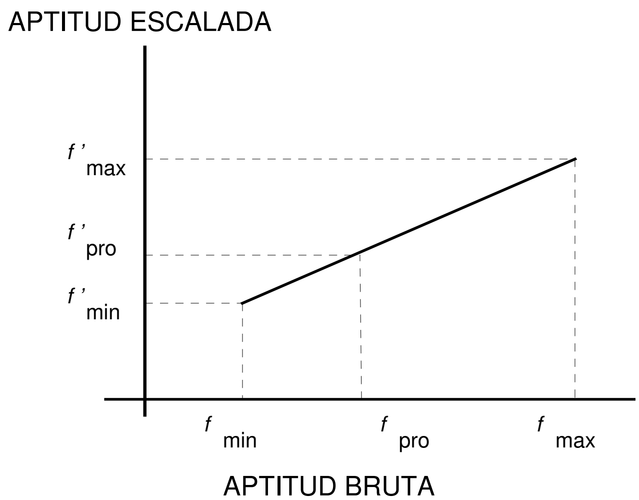



**Figura 2.4** Escalamiento lineal

## *2.5.3. Restricciones*

Muchos problemas reales no sólo requieren que se optimice una función objetivo, sino que además se deben cumplir ciertas restricciones.

El problema de manejar restricciones se puede resolver no asignando ninguna aptitud a estructuras que se detecten fuera de las restricciones y esperar a que sean removidas por el mecanismo de reemplazo por estructuras válidas. Si se sigue este método, en un problema severamente restringido, sería tan difícil encontrar estructuras válidas como encontrar la solución al problema porque no se sabe nada de las características de una estructura no válida. Para tener información acerca de las estructuras ilegales se puede utilizar un *método de penalización* que degrade la aptitud de las estructuras en la medida en la que violen las restricciones.

En un método de penalización un problema con restricciones se transforma en un problema sin restricciones asignando una penalización a las violaciones. Este costo o penalización se incluye en la función objetivo. Simbólicamente, el problema original de minimización:

Minimizar g(x)

Sujeto a: $h_i(x) \geq 0 \quad i = 1, 2, ..., n$

se transforma en la forma no restringida:

$$\text{Minimizar:} \quad g(x) + r\sum_{i}^{n} \Phi\left[h_i(x)\right]$$

dónde $\Phi$ es la función de penalización y *r* es el coeficiente de penalización.

Existen varias alternativas para la función de penalización. En muchos casos es adecuado utilizar el cuadrado de la violación de la restricción, $\Phi = h_i^2(x)$. El valor de *r* se fija diferente para cada problema.

*Capítulo 3*

# *Algoritmos Genéticos Paralelos*

La primera motivación para utilizar Algoritmos Genéticos Paralelos (AGPs) es exactamente la misma que existe para utilizar cualquier tipo de procesamiento paralelo: se desea mejorar el desempeño de los programas para poder enfrentar problemas mayores y entregar los resultados en un tiempo menor.

Otra razón es que se han reportado resultados que presentan a los AGPs como mejores algoritmos (o al menos iguales) para resolución de problemas que los AGs seriales. Los AGPs no sólo reducen el tiempo de procesamiento (incluso reduciendo el número de evaluaciones totales necesarias), sino que en muchas ocasiones entregan resultados significativamente mejores que los algoritmos convencionales [Tan89].

El tercer motivo es muy interesante a largo plazo. Los AGs basan su funcionamiento en la simplificación de un proceso que ocurre en la naturaleza: evolución a través de selección natural. Se puede pensar que si se continúa imitando adecuadamente a la naturaleza es probable encontrar refinamientos que nos lleven a mejores algoritmos [Gol89b]. En ciertos aspectos, los AGPs representan la dinámica de una población mejor que un AG serial, incluyendo aspectos como formación de especies y migración (explicados más adelante). La abstracción de estos mecanismos que se utiliza en un AGP puede contribuir a su comprensión en sistemas naturales [Müh89].

En este capítulo se introducen los algoritmos genéticos paralelos. A continuación se presentan los diferentes tipos de AGPs propuestos hasta ahora y en seguida se hace una comparación crítica entre ellos.

## 3.1. Taxonomía

Se pueden distinguir cuatro niveles de paralelismo entre los distintos modelos de AGPs propuestos. Estos niveles establecen de alguna manera la capacidad de los algoritmos de explotar eficientemente los recursos disponibles en una máquina paralela.

Ejecutar un AG convencional en una máquina paralela es el primer nivel de paralelismo posible. A pesar de que es muy probable que el compilador paralelice algún cómputo, el comportamiento del AG no se ve afectado. La arquitectura de la máquina paralela no importa mucho porque no se hace ningún intento por explotarla explícitamente.

El siguiente nivel de paralelismo se encuentra cuando la evaluación de los individuos se hace de manera paralela usando distintos procesadores, pero manteniendo una sola población. Este nivel de paralelismo se encuentra en dos de los prototipos que John Grefenstette propuso en 1981 [Gre81]. A la implementación utilizando memoria compartida la llamó *AG asíncrono concurrente*. Grefenstette también propuso unos prototipos *maestro-esclavo*, dónde el maestro tenía la población y los esclavos eran utilizados para evaluar individuos. La implementación de este prototipo es muy sencilla en máquinas con memoria distribuida.

Estas dos primeras aproximaciones no cambian el comportamiento del algoritmo. Procesan la población de la misma manera que un AG serial, pero obteniendo resultados en un tiempo menor. Varios investigadores utilizan estas opciones, pero generalmente están conscientes de que la única ventaja que pueden obtener es que el tiempo de procesamiento se reduce.

El siguiente nivel de paralelismo se explota en los algoritmos genéticos paralelos de grano burdo. La granularidad se define como una razón entre el tiempo dedicado a procesamiento y el tiempo dedicado a comunicación. Si esta relación es alta, se dice que los procesos son de grano burdo. Este modelo de AGP normalmente se implementa en arquitecturas de memoria distribuida dónde la comunicación entre procesos puede ser más lenta que utilizando memoria compartida.

Grefenstette propuso el prototipo de *red* siguiendo estas líneas. Su prototipo utiliza procesos independientes que se comunican en cada generación intercambiando a los mejores individuos.

El nivel más alto de paralelismo se encuentra en los AGPs de grano fino. Explota arquitecturas masivamente paralelas eficientemente con un algoritmo que en principio es bastante sencillo. Utiliza intensivamente las capacidades de comunicación entre procesadores, y es fácilmente escalable al número de procesadores disponibles.

Actualmente existe una gran discusión entre cuál de los últimos dos modelos es mejor. Se han hecho y reportado resultados de experimentos, algunos favoreciendo a los algoritmos genéticos paralelos de grano burdo (e.g. [GoWh93]) y otros a los de grano fino (e.g. [Bal93]).

### 3.1.1. AGPs de grano burdo

Una forma muy natural de paralelizar un AG es dividir a la población en subpoblaciones, ejecutar un algoritmo genético convencional en cada subpoblación y permitir la comunicación periódica de información entre las subpoblaciones que auxilie en la búsqueda de la solución. La información que se intercambia normalmente es un subconjunto de los individuos más aptos de cada subpoblación. También se pueden enviar estadísticas como el promedio o la desviación estándar de las evaluaciones de la subpoblación para intentar determinar el momento de intercambiar individuos o de finalizar el algoritmo [MTS93].

A este modelo se le conoce de diversas maneras en la literatura: algunos lo llaman *algoritmo genético distribuido* [Tan89] resaltando el hecho de que normalmente se implanta en máquinas MIMD (Multiple Instriction stream, Multiple Data stream) con memoria distribuida. Otros, utilizando términos de genética poblacional, lo llaman *modelo de isla* [Müh89] porque la subpoblación se encuentra aislada de las demás salvo por algún intercambio más o menos frecuente de individuos.

### *3.1.1.1. Formación de especies*

En los AGPs de grano burdo surge un fenómeno que tiene una gran semejanza con el fenómeno natural de *formación de especies.* La formación de especies es el proceso por el cual una especie se convierte en dos especies, una de las cuales puede ser la original [Daw86]. Supongamos una población grande de alguna especie animal que vive en un territorio limitado en uno de sus extremos por alguna barrera natural (una cordillera de montañas, un río, el mar, un desierto, etc.). En un momento dado, se puede presentar algún fenómeno que facilite el tránsito a través de la barrera natural (por ejemplo una sequía). Cuando esto sucede, algunos individuos cruzan la barrera natural formando una subpoblación de individuos de la misma especie que la población original, pero mucho más pequeña.

Supongamos que después de algunos años el tránsito a través de la barrera se cierra nuevamente. Es posible que la subpoblación se encuentre en un lugar dónde las condiciones son favorables para su supervivencia, pero distintas a las de su hábitat natural. Los individuos de la subpoblación se deben adaptar a las nuevas condiciones de vida. La selección natural favorece a aquellos individuos ligeramente mejor adaptados a las nuevas circunstancias y empiezan a tener descendientes. Como la subpoblación es relativamente pequeña, los cambios genéticos que se presentan se difunden rápidamente en ella.

Las dos poblaciones se enfrentan a ambientes distintos por lo que cada una evoluciona de manera distinta. Cada una explota el nicho que le corresponde. Después de un tiempo es posible que los individuos de la subpoblación hayan cambiado lo suficiente como para considerarlos una raza distinta a la original. Si pasa mucho tiempo sin que exista contacto entre las poblaciones, los cambios en su material genético pueden ser tan severos que no permitan la reproducción entre individuos de las dos poblaciones. Cuando esto sucede se considera que se formó una nueva especie.

Si existe un encuentro entre individuos de especies distintas, es posible que puedan producir descendientes, pero estos seguramente tendrán limitaciones graves en su propia reproducción (e.g. un burro y una yegua producen una mula estéril).

Es muy probable que si las dos poblaciones se vuelven a encontrar, aunque sean especies distintas, compitan por los mismos recursos y en un caso extremo (pero no demasiado raro) una de las dos especies se extinga.

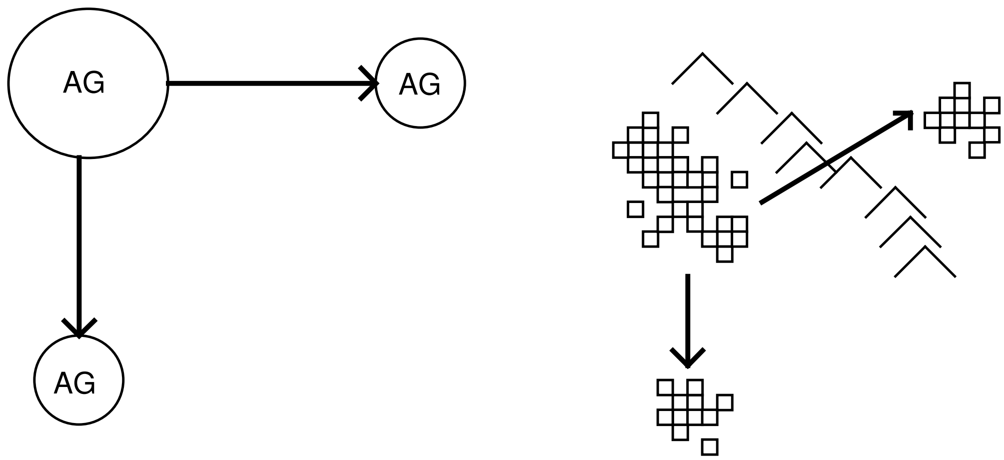



**Figura 3.1** Analogía entre un AGP y diversas poblaciones de organismos.

En los algoritmos genéticos paralelos la situación es muy similar (ver figura 3.1), tenemos subpoblaciones relativamente aisladas unas de otras, con selección y apareamiento locales. En este modelo los individuos sólo tienen oportunidad de aparearse con otros miembros de la subpoblación por lo que las subpoblaciones convergen rápidamente hacia óptimos locales, semejantes a distintas especies.

La pregunta relevante ahora es ¿qué consecuencias tiene la aparición de especies para el algoritmo? Si utilizamos un algoritmo genético convencional para maximizar una función como la de la figura 3.2, es predecible que los individuos de la población, después de un número suficiente de generaciones, converjan en uno sólo de los máximos. Esto se debe a errores de muestreo causados por poblaciones pequeñas.

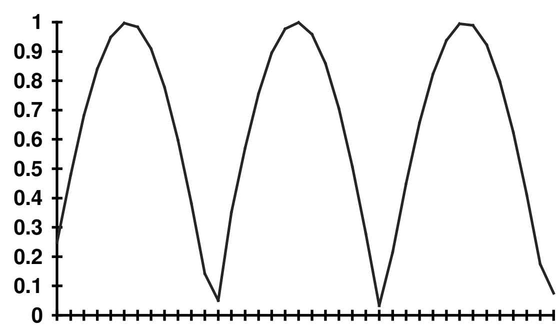



**Figura 3.2** Si no hay formación de especies los individuos convergen a sólo uno de los máximos.

Si en cambio, utilizamos AGPs para propiciar la aparición de especies, es posible esperar que al final de la ejecución existan individuos de la población que representen a los tres máximos. Cada máximo en la gráfica equivale a un *nicho* en el medio ambiente. Los tres máximos tienen el mismo valor y no existe un criterio para preferir a alguno.

Fomentar la creación de especies permite explorar mejor el espacio de búsqueda reduciendo la probabilidad de que la población converja a un sólo óptimo local. La formación de especies no es un fenómeno exclusivo de los AGPs. Goldberg [Gol89] propone un método para fomentar la creación de especies utilizando un AG serial, manipulando el valor obtenido de la evaluación de un individuo con una función de compartición (*sharing*). La función de compartición se usa para dividir la recompensa del medio ambiente (la función de evaluación) entre individuos similares. Cuando hay muchos individuos similares, su aptitud se debilita por lo que este mecanismo limita el crecimiento de una especie particular permitiendo la aparición de otras:

$$f_c(x_i) = \frac{f(x_i)}{\sum_{j=1}^{n} s(d(x_i, x_j))}$$

donde $fc$ es la aptitud resultante, $s$ es la función de compartición y $d$ es la distancia de Hamming entre dos individuos (una forma de medir la similitud entre individuos).

### *3.1.1.2. Migración*

El intercambio de individuos entre subpoblaciones es conocido como *migración* [Tan89]. Al utilizar migración en un AGP surgen varias cuestiones:

1. ¿Con qué frecuencia migran individuos de una población a otra?
2. ¿Cuántos individuos migran cada vez?
3. ¿Hacia dónde migran?
4. ¿Cuáles individuos migran?

A cada una de estas preguntas corresponde un parámetro nuevo para el AGP. La primera pregunta se resuelve introduciendo el *intervalo de migración*, que se define como el número de generaciones que ocurren entre migraciones. La *tasa de migración* es la respuesta de la segunda pregunta y define la proporción de la población que migra en un momento dado.

Al recibir individuos de otras subpoblaciones (migrantes) se introduce diversidad y se incrementa la presión de la selección (se aumenta la competencia por sobrevivir) en cada subpoblación. Sin embargo, si la subpoblación ha convergido a un estado de equilibrio, la introducción de nuevo material puede no ser efectiva porque las subpoblaciones pueden haber encontrado buenas soluciones, pero muy alejadas una de la otra y la combinación de ambas pueden dar como resultado una solución mala. Cada subpoblación ha encontrado un nicho adecuado y se han formado especies distintas. Igual que en la naturaleza, el cruce entre especies generalmente no da buenos resultados.

Goldberg propone un diseño orientado a objetos para AGPs llamado modelo de *polinización de plantas* [Gol89]. Este modelo consiste en una serie de nodos conectados por una red de polinización. Cada liga de esta red tiene asociada una probabilidad de "transmisión de polen". Esta probabilidad es semejante a una combinación de tasa e intervalo de migración y probablemente facilite el análisis del algoritmo, pero desgraciadamente esto solamente es una propuesta y Goldberg no presenta ningún resultado y tampoco ningún análisis.

Es predecible que un algoritmo paralelo con grandes cantidades de intercambio de información entre las subpoblaciones tendrá un desempeño menor que uno que utilice menos los recursos de comunicación. Pero la cantidad de interacción también afecta la efectividad del algoritmo genético. Un AGP con ninguna comunicación es equivalente, en la calidad de la búsqueda, a ejecutar varios AGs seriales sobre subpoblaciones pequeñas. En el otro extremo, en un AGP con demasiada comunicación las ventajas de tener subpoblaciones se pierden, los buenos cromosomas de una subpoblación se extienden rápidamente a las demás y las evoluciones pierden independencia.

Tanese investiga estos dos parámetros, el intervalo y la tasa de migración, utilizando una topología de hipercubo para comunicar a las subpoblaciones y presenta resultados que favorecen la elección de parámetros para una migración moderada [Tan89].

La tercera pregunta, ¿hacia dónde migran los individuos?, define la *topología* de comunicación entre las subpoblaciones. A pesar de que se han hecho algunos estudios comparando algunas topologías [BiBr92][MTS93][GoWh93], hasta ahora no hay estudios concluyentes que demuestren que una topología sea mejor que otras en una amplia variedad de problemas.

Uno de los objetivos de esta tesis es precisamente experimentar con varias topologías usando una variedad de funciones con características diferentes para caracterizar a las topologías que funcionan mejor con problemas diversos.

La última pregunta se refiere al criterio de selección de los migrantes. Es bastante razonable pensar que en una población natural los individuos que migran son generalmente los más fuertes, pues migrar implica recorrer cierta distancia y posiblemente a través de obstáculos. Haciendo una analogía de esto, en un AGP normalmente migran los individuos con mejores valores de evaluación, aunque también se puede pensar en seleccionar aleatoriamente a los migrantes.

Al enviar migrantes de una población a otra en un AGP normalmente se envían copias de los individuos seleccionados para migrar, i.e. no se eliminan a los migrantes de la subpoblación de origen, lo que obviamente es poco natural, pero posiblemente ayude a que el algoritmo converja a una buena solución.

#### *3.1.1.3. Interrupción del equilibrio*

Cuando ocurre algo que propicia la aparición de una especie, los individuos mejor adaptados a las nuevas condiciones se reproducen y dominan la población. Durante esta etapa, los cambios en el material genético de la población son muy severos y ocurren en un período de varias decenas de miles de años (que se considera como un tiempo corto). Cuando la población se adapta a las nuevas condiciones y los cambios empiezan a ser menores, se dice que la población está en equilibrio.

La teoría de Darwin del origen de las especies dice que los cambios en los individuos son *graduales*, que no existen cambios bruscos entre generaciones. Muchas veces, esta teoría se interpreta incorrectamente como que la velocidad a la que estos cambios ocurren es constante o permanente. Las poblaciones pueden quedarse en equilibrio durante mucho tiempo hasta que ocurra algo que favorezca a individuos distintos. De hecho, se puede pensar que las poblaciones presentan una resistencia al cambio directamente proporcional a su tamaño [5].

---

[5] Obviamente no existe ninguna resistencia activa por los miembros de la población contra el cambio. Lo que sucede es que los genes con las características diferentes se difunden más lentamente en una población grande que en una pequeña.

Si los individuos están bien adaptados a su medio, la presión de la selección disminuye. Si no existe una razón para cambiar, la población no cambia, no se exploran variaciones y se pierde diversidad. La aparición de cambios evolutivos se incrementa si ocurre algún fenómeno en la población que tienda a contrarrestar una estabilidad excesiva [May60].

Uno de estos fenómenos es la migración. Si una especie migra a una zona en donde existe otra especie con los mismos hábitos existirá competencia entre ambas. En ecología es bien conocido el principio de que dos especies con los mismos hábitos no viven mucho tiempo en el mismo lugar [Daw86]. La presión de la selección se incrementa y el equilibrio en que se encontraba la población que recibe a los migrantes se interrumpe. La selección favorece nuevos cambios que mejoren la supervivencia.

Los algoritmos genéticos convergen rápidamente hacia la solución en las primeras generaciones. Cuando se encuentran en la región cercana a la solución empiezan a refinar su población acercándose lentamente al óptimo (ver figura 3.3). En esta etapa en que los cambios son muy pequeños se dice que la población está en *equilibrio*.

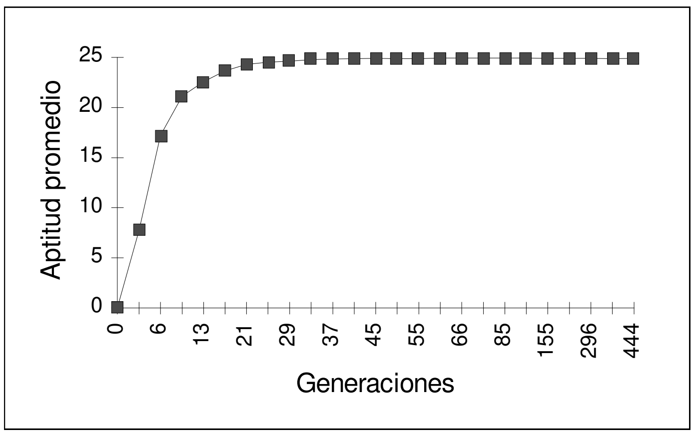



**Figura 3.3** La población de un AG alcanza rápidamente el equilibrio

La migración en un AGP puede romper el equilibrio de cada subpoblación. Los individuos que lleguen competirán con los mejores individuos de la subpoblación y romperán el equilibrio que existía ahí. Se incrementará la diversidad y se explorará mejor el espacio de búsqueda, reduciendo la posibilidad de caer en un óptimo local.

### *3.1.1.4. Sincronización*

En un sistema distribuido, cuando un proceso requiere información de otro, generalmente hay que introducir algún mecanismo de sincronización entre los procesos para evitar que la información se transmita o se utilice en un momento incorrecto. En un algoritmo genético paralelo, el concepto de sincronización se refiere a que el intercambio de individuos entre procesos se realice cuando ambos estén procesando la misma generación.

¿Cuál es el sentido de sincronizar a un AGP? ¿Qué sucede cuando un individuo de la generación 40 de un proceso llega a un proceso que se encuentra en la generación 39 o 41? Los individuos que migran entre poblaciones no contienen ninguna información referente a su origen, se incorporan a la población como si hubieran sido producidos por los operadores genéticos convencionales. Son seleccionados usando como único criterio su aptitud y si sobreviven se reproducen con otros individuos seleccionados de la misma manera. No existe realmente ningún argumento que sostenga la idea de que un individuo esté mejor adaptado que otro sencillamente porque pertenece a una generación mayor. Existen resultados experimentales que muestran que en problemas de optimización de funciones la sincronización no influye mucho en la capacidad del algoritmo de encontrar el óptimo global [ZeKi93].

### 3.1.2. AGPs de grano fino

En este modelo la población se distribuye en una cantidad mucho mayor de subpoblaciones que en un AGP de grano burdo y la comunicación es mucho más abundante. Se implanta fácil y eficientemente en máquinas masivamente paralelas con capacidades de comunicación superiores a las máquinas con memoria distribuida. A este modelo también se le conoce algunas veces como *AG celular* [Whi93].

Normalmente la población se distribuye en un arreglo rectangular de procesadores. Los procesadores de los extremos se comunican con el procesador en el extremo opuesto formando una topología de *torus*. Para cada procesador se define un *vecindario*, que es el conjunto de procesadores con los que se puede comunicar. El número de vecinos de cada procesador está determinado por el *radio* del vecindario que equivale al número de enlaces que separan al procesador más alejado del centro del vecindario (ver figura 3.4).

Se han utilizado otras topologías en AGPs de grano fino. Por ejemplo, Mühlenbein [MSB91] utiliza una topología en forma de escalera y establece vecindarios de una forma similar a la descrita.

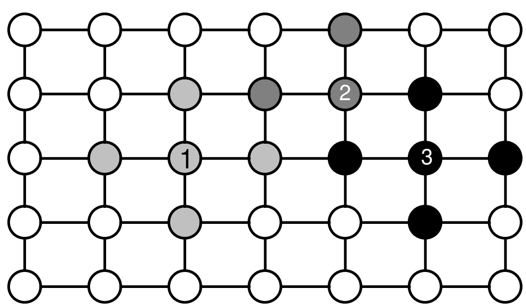



**Figura 3.4** Tres vecindarios de radio 1. Los vecindarios de los individuos 2 y 3 tienen dos individuos en común.

La selección y el apareamiento están limitados al vecindario de cada individuo. Es interesante notar que los vecindarios se traslapan, por lo que existe intercambio de información entre vecindarios (figura 3.4). Así, si algún individuo sobresale en su vecindario, será seleccionado para aparearse y sus características se transmitirán a sus descendientes. Es muy probable que los descendientes también sobresalgan en otros vecindarios a los que pertenezcan y sean seleccionados. Las características que hacían superior al primer individuo se comunican al resto de la población a través de diferentes vecindarios. La velocidad con que éstas características se transmiten a toda la población depende del tamaño de los vecindarios: entre más grandes sean la velocidad será mayor.

# 3.2. Discusión

En esta sección se analizaran las ventajas y desventajas de los AGPs de grano burdo y de grano fino.

Los algoritmos genéticos convergen rápidamente en las primeras generaciones y después se acercan lentamente al óptimo. La velocidad de convergencia está influenciada por varios factores, pero uno de los más importantes es el tamaño de la población. Entre menor sea la población, la convergencia será más rápida porque se perderá diversidad más fácilmente. En la figura 3.5 se puede ver un ejemplo.

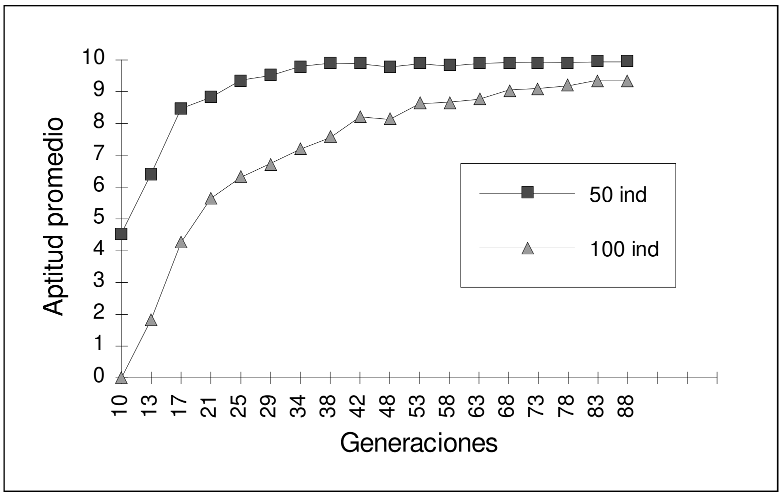



**Figura 3.5** Convergencia de dos poblaciones de diferente tamaño.

El principal problema con los AGPs de grano burdo es que las evoluciones son bastante independientes. Esto puede provocar que las subpoblaciones, que son relativamente pequeñas, converjan muy rápido y posiblemente a óptimos locales. Por otro lado, la independencia de las evoluciones puede ser benéfica en problemas difíciles en dónde es necesario explorar muchas regiones diferentes del espacio de búsqueda [Bal93].

La migración de individuos ayuda a evitar convergencias prematuras en óptimos locales introduciendo diversidad en las subpoblaciones. La migración es un fenómeno muy complejo cuya efectividad probablemente depende de las características del problema que el algoritmo está resolviendo. En algunos problemas, la migración en las primeras generaciones puede beneficiar al algoritmo llevando bloques constructivos de una población a otra, mejorando efectivamente la diversidad. Sin embargo, en otros problemas la migración en esta etapa puede ser un desperdicio. Por ejemplo, en funciones continuas y unimodales no se necesita mucha diversidad en la población para encontrar el óptimo global.

En etapas avanzadas del algoritmo, cuando las subpoblaciones están más o menos estables (i.e. han convergido), los migrantes pueden romper el equilibrio y llevar a la población hacia otra región del espacio de búsqueda. Pero esto también depende del problema, en una función con muchos óptimos locales, conviene romper ocasionalmente el equilibrio para probar regiones no exploradas. Si el algoritmo converge rápidamente hacia el óptimo global, entonces las migraciones no aportan mucho.

Cuando los migrantes son seleccionados y se recombinan con los individuos existentes en la población, puede ocurrir que los descendientes sean inferiores. Esto va a provocar que el promedio de la población descienda por un tiempo corto hasta que estos descendientes sean eliminados por la selección [Tan89]. Esto se puede apreciar en la figura 3.6.

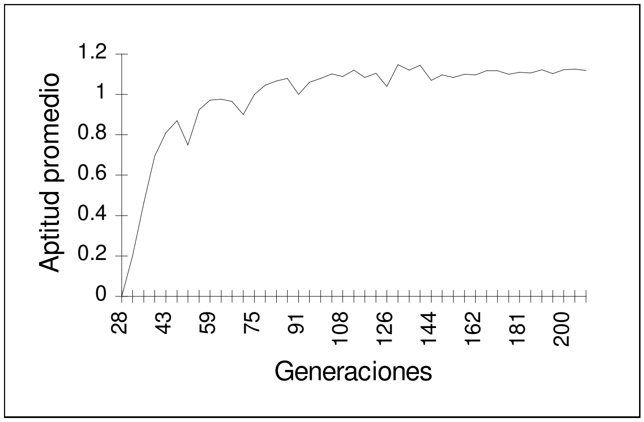



**Figura 3.6** El equilibrio se rompe con las migraciones

En los AGPs de grano fino hay menos independencia en las evoluciones y por eso los óptimos locales se pueden propagar por toda la población [Bal93]. Esto puede retrasar seriamente (o incluso impedir) la aparición de la solución global. Además, el tamaño de las subpoblaciones es pequeño y los esquemas de los óptimos locales dominan rápidamente la información genética de las subpoblaciones. Esto se puede mejorar limitando la cantidad de intercambio de información entre las subpoblaciones. También ayuda el hecho de tener un número grande de subpoblaciones: subpoblaciones distantes pueden evolucionar soluciones únicas de la misma manera en que lo hace un AGP de grano burdo. A esto se le llama *aislamiento por distancia*.

Como las poblaciones se traslapan en los AGPs de grano fino, existe una interacción continua y se elimina el problema del rechazo de soluciones incompatibles. El equilibrio no se alcanza en una subpoblación hasta que se alcanza por los vecinos.

Probablemente, el AGP de grano burdo sea en general un mejor modelo de una población natural. Los fenómenos de formación de especies, interrupción del equilibrio y migración se pueden simular y observar mejor en un AGP de grano burdo que en uno de grano fino. Es razonable pensar que si se imitan fenómenos que dan buenos resultados en la naturaleza se pueden crear técnicas que permiten resolver eficazmente problemas complejos [Gol89b]. Esta es una de las razones principales por las que en esta tesis se decidió implementar un AGP de grano burdo.

*Capítulo 4*

# DGENESIS

El objetivo principal de esta tesis fue desarrollar un sistema que permita la experimentación con AGPs de grano burdo. El sistema hecho para esta tesis se llama Distributed GENESIS (DGENESIS). La intención de este capítulo es dar una visión general sobre el diseño e implantación de DGENESIS. Se exponen los problemas importantes que se presentaron durante el desarrollo de DGENESIS y se comentan los algoritmos utilizados para resolverlos en cada caso. El código no se presenta, pero en algunas figuras se representan las estructuras de datos mas importantes para ayudar a aclarar algunos puntos en el texto.

No se pretende que el lector de este capítulo aprenda a operar el sistema. Los programas son descritos someramente y sólo para dar una visión del funcionamiento del sistema. Se preparó un Manual de Usuario [Can94] que se distribuye con el sistema y la persona interesada en instalarlo y realizar experimentos con él debe acudir a ese documento.

## 4.1. Herramientas

Para determinar la funcionalidad que debía tener el sistema que se pretendía construir para cumplir los objetivos de esta tesis se realizó una búsqueda de herramientas existentes de experimentación con algoritmos genéticos. Fue posible localizar bastantes herramientas y se realizó una evaluación de todas ellas. Al mismo tiempo se contemplaba la posibilidad de utilizar alguna como base del sistema de esta tesis.

Nicol Schraudolph mantenía un documento [Sch93] en dónde se describían sistemas comerciales y experimentales (desarrollados por investigadores). La mayoría de los sistemas experimentales están disponibles vía ftp[6] y se utilizó este medio para adquirirlos. En la tabla 4.1 se presentan los sistemas evaluados, su autor, el lenguaje utilizado para desarollarlos y el sistema operativo en que se ejecutan.

Después de evaluar estos sistemas, se decidió construir el sistema sobre GENESIS 5.0 por su popularidad en la comunidad de investigadores en el área de algoritmos genéticos y por la sencillez y elegancia de su código. GAucsd 1.4 y GENEsYs están basados en GENESIS 4.5,

---

[6] en `ftp.aic.nrl.navy.mil` (Artificial Intelligence Center, National Research Laboratory)

predecesor de la versión elegida, lo que permite suponer que si se desea la modificación del sistema desarrollado para incluir algunas propiedades de estos programas, se podrá hacer sencillamente.

Genetic Algorithm Workbench y BUGS son sistemas interactivos orientados a demostrar el funcionamiento de un algoritmo genético y no se pueden utilizar en problemas de optimización de funciones. En Genetic Algorithm Workbench, el usuario *dibuja* la función de evaluación y permite observar la forma en que la población se adapta a la función. BUGS es un sistema que presenta gráficamente a los individuos de la población y el usuario selecciona (subjetivamente) a los individuos que se deben aparear para formar la siguiente generación.

SGA-Cube y PARAGENESIS requieren de computadoras especiales, por lo que se reduce su utilidad y resultan inadecuados para ser utilizados como base para otros sistemas. PGA 2.4 se obtuvo cuando la etapa de implementación de esta tesis estaba muy avanzada. De cualquier manera, PGA tiene un mecanismo de migración poco flexible y aunque su análisis resultó interesante no influyó en esta tesis.

| Sistema | Autor | Lenguaje | S. O. (máquina) |
|---|---|---|---|
| BUGS | Joshua R. Smith [Smi90] | C | DOS |
| GAucsd 1.4 | Nicol Schraudolph [ScGr92] | C | UNIX |
| GENESIS 5.0 | John Grefenstette [Gre90] | C | UNIX / DOS |
| GENEsYs 1.0 | Thomas Bäck [Bäc92] | C | UNIX |
| Genetic Algorithm Workbench | Mark Hughes [Hug89] | C++ | DOS |
| SGA-Cube | Jeff Earickson, Robert Smith y David Goldberg [ESG91] | C | (NCUBE2) |
| PARAGENESIS | Michael C. van Lent [van92] | C* | (CM-200) |
| PGA 2.4 | Geoffrey H. Ballinger y Peter Ross [BaRo93] | C | UNIX (/ DOS) |

**Tabla 4.1** Sistemas evaluados

El diseño de DGENESIS, el sistema hecho para esta tesis, se centró en modificar adecuadamente a GENESIS 5.0 para convertirlo en un programa distribuido que permitiese la experimentación con algoritmos genéticos paralelos.

La idea general fué distribuir la población entre los nodos participantes, enviar a cada nodo los parámetros que controlan la ejecución del algoritmo, intercambiar individuos periódicamente entre los nodos y, al final, presentar un reporte con la información relevante.

Para DGENESIS, un nodo es un proceso de UNIX. Para la comunicación entre los procesos se utilizaron sockets de Berkeley [LeFa] y la programación se hizo en lenguaje C. El protocolo que se utilizó para la comunicación fue UDP (User Datagram Protocol). Este protocolo no establece conexiones entre los procesos, lo que facilita considerablemente la implementación (especialmente en la fase inicial) y además la transmisión de mensajes es más rápida.

UDP no es un protocolo completamente confiable porque no incluye procedimientos para detectar la pérdida de mensajes y tampoco asegura que los mensajes lleguen a su destino en la misma secuencia en la que fueron enviados. Sin embargo, UDP presenta pocos problemas de confiabilidad cuando se le utiliza en una red local [Ste90], que es precisamente el ambiente en el que se pretende ejecutar DGENESIS.

A grandes rasgos, el funcionamiento del sistema se puede dividir en cuatro fases: la configuración, la comunicación inicial, la comunicación durante el experimento y la terminación.

# 4.2. Configuración del sistema

Antes de ejecutar los procesos del AGP, es necesario que el usuario especifique los parámetros de operación del algoritmo genético, el número de procesos y la manera en que se comunicarán entre sí. Esta información se almacena en archivos que serán utilizados posteriormente. Para facilitar la creación de estos archivos se modificaron algunos programas existentes en GENESIS y se crearon otros.

## *4.2.1. Función de evaluación*

La función que el algoritmo genético utilizará para evaluar a los individuos de su población debe ser proporcionada por el usuario. GENESIS tiene una manera bastante sencilla y práctica para especificar la función objetivo y se decidió incorporar esto sin modificaciones a DGENESIS.

La función de evaluación que el usuario debe proporcionar debe estar escrita en C y tiene el siguiente encabezado:

```
double eval(str, length, vect, genes)
char str[];      /* representación en cadena          */
int length;      /* longitud de la cadena             */
double vect[];   /* representación en punto flotante  */
int genes;       /* numero de elementos en vect       */
```

La función recibe el cromosoma como una cadena de caracteres (`str`) con ceros y unos, y/o como un arreglo de números en punto flotante (`vect`) que representan los valores de las variables del problema. Antes de ejecutar el sistema, el usuario establece si desea que DGENESIS decodifique el cromosoma como números reales antes de llamar a la función de evaluación. Si el problema no involucra variables reales, el usuario es responsable de la decodificación del cromosoma y `vect` no contiene nada significativo.

La cadena de caracteres equivale al genotipo y el arreglo de números en punto flotante es el equivalente al fenotipo de un organismo.

## *4.2.2. Parámetros del algoritmo genético*

Para especificar los parámetros que controlan la ejecución del algoritmo genético, el usuario debe utilizar un programa hecho expresamente para esto (`setup`). Los parámetros se guardan en un archivo que será leído posteriormente por el algoritmo genético y por el programa que genera los reportes.

El programa que se utiliza en GENESIS para especificar los parámetros se modificó para utilizar parámetros usados en experimentos previos como valores por default. Así es muy sencillo cambiar únicamente uno o dos parámetros del AG y volverlo a ejecutar. Antes de esto el usuario tenía que editar manualmente los archivos con el riesgo de modificar sus formatos y hacerlos inútiles.

Los parámetros que se especifican con este programa son:

- Número de experimentos. Un experimento es la ejecución completa de un AGP. Se pueden realizar varios experimentos utilizando la misma función de evaluación y los mismos parámetros del AG, pero con poblaciones iniciales distintas. Esto permite obtener resultados más confiables.
- Número de evaluaciones por experimento en cada nodo. Este es el principal criterio para detener al algortimo.
- Tamaño y formato de las estructuras. `setup` pregunta si se desea que las variables codificadas en los individuos se interpreten como números reales y en caso positivo pregunta los valores extremos de cada variable y el formato que se utilizará para imprimirlas.
- Probabilidad de cruce y de mutación.
- Brecha generacional.
- Ventana de escalamiento. En el capítulo dos se comentó que la aptitud de los individuos se escala para evitar que unos cuantos individuos dominen rápidamente a la población. El problema es que se deben conocer de antemano los valores extremos que puede tomar la función objetivo y normalmente esto es desconocido. Para solucionar esto se toman los valores extremos observados en las últimas $n$ generaciones. El valor de $n$ es el tamaño de la ventana de escalamiento. Durante las primeras $n$ - 1 generaciones solamente se usa la información que se tiene disponible.
- Generaciones sin evaluación. Ciertas combinaciones de parámetros hacen posible que no se generen descendientes diferentes en varias generaciones, especialmente en etapas avanzadas del algoritmo. Si durante el número de generaciones especificado no se encuentran puntos nuevos, el algoritmo es terminado.
- Intervalo entre respaldos. GENESIS permite crear un archivo de respaldo con la población actual y con las variables que controlan la ejecución de un experimento para poder interrumpirlo en un momento dado y continuar posteriormente. Este parámetro especifica el número de generaciones que pasan antes de crear este archivo.
- Número de respaldos que se deben salvar. Si se desea analizar la población del algoritmo en distintos momentos de la ejecución, se pueden salvar varios archivos de respaldo.
- Opciones diversas como colectar estadísticas o imprimirlas en la pantalla o en archivos. En el manual de usuario se describen con detalle.

- Semilla del generador de números aleatorios. Cada nodo del AGP recibe una semilla diferente para que las poblaciones iniciales sean distintas. La semilla de cada nodo es un múltiplo del número que se introduce aquí.

- Rango mínimo. Este parámetro se utiliza para el algoritmo de selección por rango, como este método de selección no se utiliza en esta tesis no se profundizará en el.

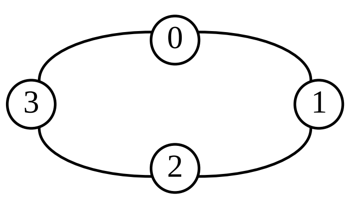
## *4.2.3. Especificación de topologías*

El sistema es muy flexible en este aspecto, permitiendo la conexión de los nodos participantes en formas muy diversas. La topología se especifica en un archivo llamado `link.x`, dónde `x` es el nombre del experimento. El archivo tiene todas las ligas entre los procesos, así como el intervalo y la tasa de migración de cada liga según el siguiente formato:

| origen | destino | intervalo de migración | tasa de migración |
|---|---|---|---|
| [int] | [int] | [int] | [float] |

Los nodos que participan en el AGP están identificados por un número entero. Los campos `origen` y `destino` de `link.x` se refieren a este identificador. El `intervalo de migración` se refiere al número de generaciones que pasan entre cada migración y `la tasa de migración` es el porcentaje de la población del nodo origen que se envía al nodo destino.

En la figura 4.1 se puede ver un ejemplo sencillo con una topología de anillo. Diez por ciento de la población de cada nodo migra cada diez generaciones al siguiente nodo. Los migrantes viajan en el sentido de las manecillas del reloj.




```
# archivo: anillo1
0   1   10   .10
1   2   10   .10
2   3   10   .10
3   0   10   .10
```

**Figura 4.1** Un ejemplo de topología.

El archivo dónde se especifica la topología se puede editar y construir manualmente o se puede utilizar el programa `links` escrito para facilitar esta operación. `links` actualmente permite crear archivos de ligas para cuatro topologías:

- Anillos unidireccionales y bidireccionales
- Hipercubos de cualquier grado
- Matrices
- Completamente conectadas

Es bastante sencillo extender este programa para crear archivos que describan otras topologías.

El programa pregunta al usuario el número de nodos de la topología, la tasa y el intervalo de migración para *todas* las ligas. Si se desea cambiar los parámetros de migración de una liga particular se debe hacer manualmente editando el archivo resultante.

### *4.2.4. Ejecución de los procesos*

Para ejecutar los procesos del AGP, se hizo un programa llamado `rgo`. El programa usa la información contenida en el archivo `usr/GAhosts` en dónde se especifican los nombres de las máquinas que pueden participar en un experimento y la cantidad de procesos que soportará cada una. El formato de este archivo es muy sencillo:

| Nombre de la Máquina | Número de Procesos |
|---|---|
| [string] | [int] |

Se obtendrán mejores rendimientos cuando se pongan números mayores de procesos en las máquinas más rápidas. No es necesario cambiar este archivo en cada experimento.

`rgo` utiliza `rsh` para ejecutar los procesos y `rcp` para copiar los archivos de resultados, por esta razón en `usr/GAhosts` se debe usar el nombre de las máquinas, no se puede utilizar su dirección IP. Por supuesto, los nombres deben estar correctamente registrados en el archivo `/etc/hosts`.

El usuario debe tener acceso a una cuenta en cada máquina registrada en este archivo. En cada cuenta debe otorgar permisos para ejecutar comandos remotos y copiar archivos registrando debidamente las cuentas de las demás máquinas en el archivo `.rhosts`.

`rgo` no sólo proporciona un medio sencillo de ejecutar los procesos del AGP, además juega un papel importante en la etapa de la comunicación inicial. Cómo se describe en la siguiente sección, los procesos deben ejecutarse en cierto orden y deben recibir algunos parámetros en la línea de comandos. `rgo` se encarga de que estos requisitos sean cumplidos adecuadamente y de manera transparente para el usuario del sistema.

## 4.3. Comunicación inicial

El propósito de la comunicación inicial es enviar a todos los procesos participantes en un experimento los parámetros necesarios para controlar el algoritmo genético y la migración. Existe un nodo (al que se llamará "principal") que se encarga de mandar esta información a los demás.

### *4.3.1. Registro de participantes*

Una vez que el usuario ha creado los archivos necesarios para la ejecución del algoritmo, se ejecutan los procesos que intervienen en la simulación del AGP.

El primer problema que se debe resolver en la etapa de comunicación inicial es la localización de los procesos que ejecutarán el AGP. Como ya se mencionó, la comunicación entre estos procesos se hace utilizando sockets, que son la interfaz definida en 4.3BSD para accesar los servicios de TCP/IP. Para poder comunicar procesos mediante sockets, es necesario conocer las direcciones de IP de las máquinas en que se están ejecutando y el número de puerto UDP asignado a los sockets. El número de puerto puede variar desde 5000 hasta 65535.

El algoritmo que se utilizó para resolver este problema es el siguiente:

1. El proceso principal se ejecuta primero, encuentra la dirección IP de la máquina en que se está ejecutando y busca diez[7] puertos UDP. Para facilitar la implementación se buscan puertos consecutivos. La dirección y el número de puerto inicial se guardan en un archivo para ser utilizados posteriormente. El proceso principal se bloquea esperando que los demás procesos se registren.

2. Se ejecutan los demás procesos y reciben en la línea de comandos la dirección y el puerto inicial del principal. Cada proceso determina la dirección de la máquina en que se está ejecutando, busca diez puertos consecutivos y envía un mensaje con esta información al nodo principal para registrarse. Idealmente, sólo los nodos que están interesados en participar responden al nodo principal registrando su intención. El criterio que utiliza cada nodo para decidir si desea participar en un experimento puede estar basado en el uso del procesador en el nodo. En la implementación actual, todos los procesos se registran, pero en el futuro sería interesante implementar algún criterio de registro.

   Hay que recordar que el usuario no se debe preocupar de la ejecución de los procesos, ni de los parámetros que se mandan en la línea de comandos, `rgo` se encarga de todo esto.

3. Al recibir las respuestas de los nodos que desean participar, el nodo principal registra sus direcciones y puertos. Uno de los parámetros que el usuario introduce al principio es el número de procesadores que deben participar en el experimento. Si se reciben menos respuestas, es imposible dar el servicio deseado, el programa indica un error[8] y aborta su ejecución. Si se reciben más respuestas de las requeridas, simplemente se ignoran.

4. Se envían mensajes a los nodos seleccionados (por sus puertos iniciales) para que se preparen a recibir los parámetros del algoritmo. En este mensaje se envía el número de identificación definitivo de cada proceso, que será utilizado para crear archivos de salida únicos. Los procesos que no reciben este mensaje mueren después de un tiempo.

Este algoritmo permite utilizar más de un proceso por máquina, lo que permite la simulación de topologías grandes utilizando pocas máquinas.

---

[7] Este número está definido como una constante en `define.h` y se puede cambiar fácilmente. Esta constante limita el número de ligas por las que se pueden recibir migrantes.

[8] Los errores del sistema se reportan en la salida estándar de errores (`stderr`) y en el archivo `log.error`

El nodo principal almacena la información de todos los procesos y de las ligas por las que deben enviar y recibir migrantes utilizando la información suministrada previamente por el usuario y las direcciones IP y puertos UDP de los procesos registrados. La estructura utilizada se representa en la figura 4.2. Cada registro del arreglo tiene información de un proceso y tiene asociada una lista con las ligas por las que debe enviar y recibir migrantes:

```
typedef struct{
   struct sockaddr_in addr;   /* direc. IP del proceso (incluye puerto)
*/
   int pop;                   /* tamaño de la población */
   int migrecv;               /* # total de migrantes que se recibirán
      */
   int migsend;               /* # total de migrantes que se mandarán
      */
   int sendlinks;             /* # de ligas para mandar migrantes
      */
   int recvlinks;             /* # de ligas para recibir migrantes
      */
   LINKSTRUCT *links;         /* lista de ligas para enviar  */
   LINKSTRUCT *linkr;         /* lista de ligas para recibir */
} PROCESS;
```

Cada nodo de las listas tiene la frecuencia y tasa de migración asociadas a la liga y una estructura con la dirección y el puerto del proceso con el que se hará la comunicación:

```
typedef struct _LINKSTRUCT {
   int in_use;                /* liga en uso? */
   struct sockaddr_in addr;   /* direc. IP del proceso (incluye puerto)
      */
   int migint;                /* intervalo de migración */
   float migrate;             /* tasa de migración      */
   struct _LINKSTRUCT *next_link; /* apuntador a la sig. liga */
} LINKSTRUCT;
```

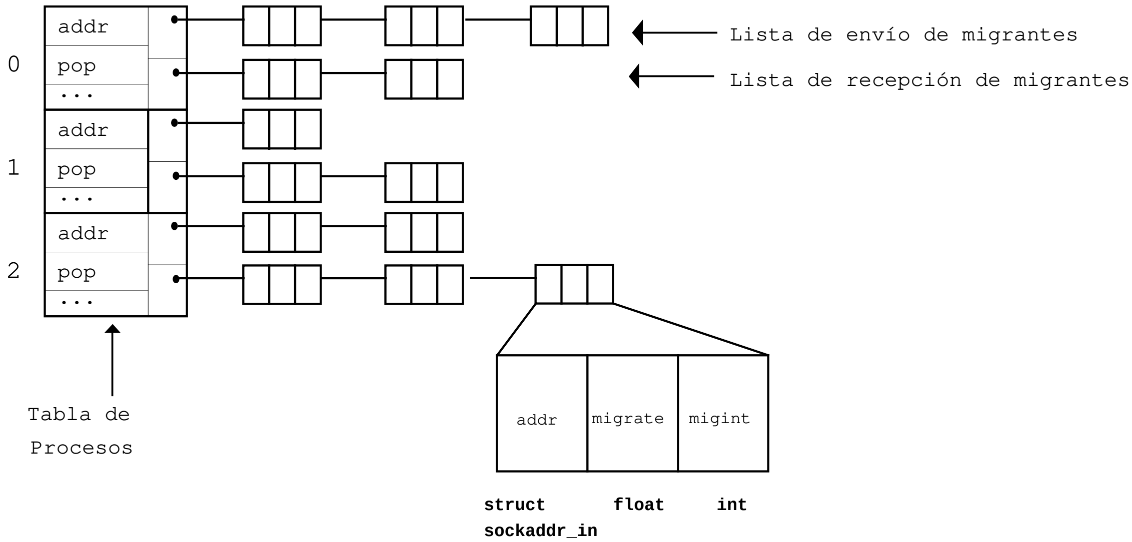



**Figura 4.2** Tabla de procesos y las listas de envío y recepción de migrantes.

## *4.3.2. Envío de parámetros*

Se envían a cada proceso (por su puerto inicial) los parámetros del AG, la plantilla que se utilizará para interpretar los cromosomas cuando estos representan variables reales y los datos de las ligas por las que enviarán y recibirán migrantes. Al recibir la información de las ligas, cada proceso forma tablas que se utilizarán en la siguiente fase de ejecución del programa. La estructura de estas tablas se representa en la figura 4.3.

Cada registro de estas tablas (de tipo `LINKSTRUCT`) es una liga por la que se enviarán o recibirán migrantes. Cada liga tiene una frecuencia y una tasa de migración asociadas y la dirección del proceso con el que se realizará la migración. Además en estas tablas hay un campo que permite habilitar o deshabilitar ligas. La utilidad de este campo se explica en la sección 4.5.

| | **struct sockaddr_in** | **int** | **float** | **char** |
|---|---|---|---|---|
| 0 | addr | migint | migrate | in_use |
| 1 | addr | migint | migrate | in_use |
| 2 | addr | migint | migrate | in_use |

**Figura 4.3** Tabla de migraciones

### *4.3.3. Población inicial*

Como se mencionó en el segundo capítulo, es posible utilizar estructuras generadas por experimentos anteriores o por otros métodos para formar una parte de la población inicial.

A primera vista, lo más apropiado es que el número de individuos que se envían inicialmente a cada nodo sea proporcional al tamaño de cada subpoblación. Esto es lógico si se toma en cuenta que en las subpoblaciones pequeñas los individuos con buenos valores de aptitud dominan más rápido a la población que en las grandes. Normalmente es deseable que las poblaciones no converjan demasiado rápido, para poder explorar adecuadamente el espacio de búsqueda.

Sin embargo, tiene cierto sentido enviar un número igual de individuos iniciales a cada nodo. Las poblaciones más pequeñas recibirían una proporción mayor de individuos iniciales y convergerán rápido. En un AG convencional podría ser desastroso, pero en un AGP es posible que la convergencia prematura de *algunas* subpoblaciones permita alcanzar rápidamente el óptimo global. Esto es obvio en funciones unimodales. En funciones multimodales, es posible que los óptimos locales tengan características comunes con el global y al recombinarse con las estructuras de otras subpoblaciones se acerquen al óptimo.

El sistema tiene la posibilidad de usar las dos formas de inicialización de la población inicial. De cualquier manera, lo que se pretende es tener un método robusto de búsqueda que sea capaz de encontrar buenas soluciones en un tiempo razonable para una cantidad grande de problemas. Mandar el mismo número de individuos a todos los nodos probablemente sólo funciona bien para algunas clases de problemas. Por esto, el default es enviar un número proporcional de individuos iniciales a las subpoblaciones.

## 4.4. Comunicación durante el experimento

El propósito de esta fase es intercambiar individuos entre los procesos participantes utilizando la topología especificada por el usuario antes de la ejecución. En este sistema, la topología es estática, esto es, no cambia durante la ejecución del experimento. Es posible diseñar un algoritmo que cambie dinámicamente la topología según ciertos criterios como la carga de trabajo de las máquinas participantes o la falla de alguna. También tiene cierto sentido cambiar la topología si se detecta alguna condición relevante en las subpoblaciones del algoritmo como falta de diversidad.

A pesar de que hay buenos argumentos a favor de tener topologías dinámicas, se decidió no implementarlas por dos motivos. El primero es que si por alguna razón se decide cambiar la topología en un momento dado, hay que decidir cómo cambiarla y eso es un problema difícil de resolver. Aun así, suponiendo que hemos identificado perfectamente los motivos válidos para cambiar de topología y hemos determinado cómo se debe cambiar en cada caso, el propósito de este sistema es hacer experimentos con ciertas topologías para determinar el efecto del número de vecinos y de la frecuencia e intensidad de las migraciones sobre el desempeño del AG. El segundo motivo para no utilizar topologías dinámicas es que estos parámetros cambiarían y no se podría determinar si la respuesta del algoritmo tiene alguna relación con la topología establecida al principio.

El ciclo principal del algoritmo genético se modificó para incluir un punto de comunicación en el que primero se envían y después se reciben migrantes. El algoritmo modificado está representado en la figura 4.4 y se puede contrastar con el de la figura 2.1. Este algoritmo se ejecuta en cada nodo y el control durante la ejecución del experimento es completamente distribuido.

```
Generar la población inicial
Evaluar a la población
Mientras los criterios de terminación no se cumplan
{
       Seleccionar a los individuos mas aptos
       Aplicar operadores
       Evaluar nuevos individuos
       Enviar migrantes si es necesario
       Recibir migrantes si es necesario
       Insertar nuevos individuos y migrantes en la población
}
```

**Figura 4.4** Ciclo principal del AG modificado.

### *4.4.1. Envío de migrantes*

Cada proceso registra en una tabla la información de las ligas por las que debe enviar migrantes (ver figura 4.3). Cada liga tiene distintos intervalos y tasas de migración, según las especificaciones del usuario.

En cada generación se prueban los registros de esta tabla para determinar si ha llegado el momento de enviar información por alguna liga. Si el resultado es positivo, se calcula el número de migrantes (tamaño de la población * tasa de migración) y se envía un mensaje con este dato al proceso correspondiente. A continuación se envían copias de las cadenas y las evaluaciones correspondientes a los individuos.

En la versión actual de DGENESIS los migrantes son los *mejores* individuos de la población actual y se envían *copias* de ellos a los procesos correspondientes. En el capítulo anterior se mencionó que existen otros criterios (e.g., aleatoriamente) para seleccionar a los migrantes y que se pueden eliminar de su subpoblación original después de ser enviados a otra. Estas opciones se planean añadir en versiones futuras de DGENESIS.

### *4.4.2. Recepción de migrantes*

En la comunicación inicial también se recibe la información de las ligas por las que se recibirán migrantes y también se forma una tabla similar a la tabla de la figura 4.3.

Cada proceso tiene varios sockets asociados y cada uno de estos sockets tiene un buffer en el que se almacenan los mensajes que se reciben hasta que el proceso los lee. Cada socket recibe información de una sola liga.

Al recibir migrantes, se colocan en un arreglo temporal. Este arreglo realmente no cumple ninguna función en el algoritmo, pero puede ser útil posteriormente si se desea ampliar el sistema para realizar algún análisis de los migrantes. Cuando se termina de recibir a los migrantes de todas las ligas, se incorporan a la población de una manera muy sencilla:

1. Se copian los migrantes de la lista temporal al final de la población, creando una población extendida.

2. Se hace selección sobre la población extendida, pero sólo sobrevive el mismo número de individuos que existían en la población inicial.

De esta manera, los migrantes son incorporados a la población si son dignos de ello de una manera muy natural. El algoritmo desconoce la procedencia de los individuos sobre los que hace selección: pueden ser migrantes, pueden haber sido producidos por los operadores genéticos o pueden ser individuos de la generación anterior.

El sistema soporta dos modos para recibir migrantes, uno asíncrono y otro síncrono. El modo de recepción se especifica como una opción del sistema con `setup`.

### *4.4.2.1. Modo asíncrono*

En el modo asíncrono la tabla de información de las ligas por las que se reciben migrantes se ignora y en cada generación se polean los sockets que se crearon al principio. Si existe información disponible en alguno se lee y los migrantes se incorporan a la población utilizando el mecanismo descrito anteriormente. En este modo se reciben migrantes en cualquier momento. Ver la sección 3.1.1.4 para una explicación de las consecuencias de esto.

### *4.4.2.2. Modo síncrono*

En este modo el proceso revisa cada generación la tabla de las ligas de recepción para determinar si debe haber alguna comunicación. En este punto pueden suceder dos cosas:

1. El proceso tiene que esperar a que el otro proceso alcance la generación indicada en la tabla y mande los migrantes.

2. Si el proceso del que se esperan los migrantes se ejecuta mas rápido, los migrantes se enviaron antes de que el primer proceso se bloqueara. Estos migrantes no se pierden: se recibieron por el socket asignado a la liga y se almacenaron en el buffer correspondiente.

De esta manera los procesos se sincronizan: los más rápidos deben esperar a que los lentos los alcancen. Los procesos lentos simplemente leen de sus sockets la información que les llegó previamente y continúan.

# 4.5. Terminación

Cada proceso en DGENESIS termina cuando alcanza el número de evaluaciones especificado por el usuario. Es posible que algunos procesos terminen antes que otros y esto es un problema si se está utilizando el modo síncrono de recepción de migrantes. En modo síncrono, los procesos rápidos esperan a que los lentos los alcancen. El problema es que esta espera es indefinida, por lo que cuando un proceso termina puede dejar bloqueado permanentemente a otro.

Para evitar este problema se propone una solución muy sencilla, pero que asegura que en ningún momento existirán procesos bloqueados esperando a otro que terminó antes. El método propuesto es que cuando un proceso haya terminado la ejecución del experimento debe enviar un mensaje indicando esto a los procesos registrados en su tabla de ligas de envío de migrantes.

Entonces, cuando un proceso determina que ha llegado el momento de esperar migrantes por una de sus ligas, debe revisar si en el buffer correspondiente al socket de la liga se encuentra el mensaje de terminación. Si se encuentra este mensaje la liga se deshabilita y el proceso continúa.

Al terminar el experimento, los procesos también avisan de esto al nodo principal. Cuando todos los procesos han terminado el nodo principal crea un archivo que le indica a `rgo` puede copiar los archivos de salida que cada proceso generó al directorio local y ejecutar el programa que hace los reportes.

# 4.6. Reportes

GENESIS cuenta con un programa que toma la salida producida por el algoritmo genético y presenta un reporte con los parámetros utilizados por el algoritmo y con tablas que registran variables que indican el progreso del AG. Cuando se realizan varios experimentos, este programa obtiene el promedio y la varianza de las variables reportadas.

Este programa se amplió considerablemente. En primer lugar había que considerar los resultados de todos los procesos que intervinieron en el experimento. Se obtiene el promedio y la varianza de estos datos y se introdujeron opciones que permiten dejar en archivos separados esta información. Los archivos con las tablas pueden ser utilizados fácilmente por otros programas para hacer análisis o graficación de los resultados [9].

---

[9] Se utilizó Microsoft Excel para leer estos archivos y producir varias de las gráficas que se incluyeron en diversas partes de este documento.

En GENESIS el tamaño de los reportes está limitado a 200 líneas por una constante definida en el código. En el programa modificado se utiliza memoria dinámica para almacenar los datos, por lo que el límite del tamaño del reporte es la capacidad de memoria de la máquina.

Adicionalmente, se pueden obtener estadísticas acerca del mejor individuo encontrado. Las estadísticas que se presentan son la generación y la evaluación en que se encontró al mejor individuo y su valor. Si se ejecutaron varios experimentos se presentan las estadísticas del mejor individuo de cada experimento y al final se obtiene la media y varianza de las generaciones y evaluaciones de todos los experimentos realizados.

## Capítulo 5

# Experimentación

La primera parte de este capítulo presenta el diseño de los experimentos realizados en esta tesis para intentar determinar el efecto de diferentes parámetros y topologías de comunicación en el desempeño de los AGPs. En la segunda parte del capítulo se presentan los resultados obtenidos en los experimentos y un análisis breve de ellos.

## 5.1. Diseño de los experimentos

En el diseño de los experimentos es necesario escoger cuidadosamente las funciones que se utilizarán en las pruebas y las medidas de desempeño que se utilizarán para comparar los resultados que se obtengan.

### *5.1.1. Funciones de prueba*

En este trabajo se utilizarán dos conjuntos de funciones frecuentemente utilizadas por investigadores para probar el desempeño de AGs y AGPs. Estas funciones son muy variadas y permiten mostrar la robustez de los AGs; algunas de las funciones son sencillas, pero otras son extremadamente complejas.

Como primera prueba se utilizaron las funciones de De Jong [DeJ75]. Estas funciones han sido utilizadas por mucho tiempo por investigadores de algoritmos genéticos. Las cinco funciones y sus mínimos se presentan en la tabla 5.1.

| Número de Función | Función | Mínimo |
|---|---|---|
| 1 | $f(x) = \sum_{1}^{3} x_i^2$ | *f(0,0,0) = 0* |
| 2 | $f(x) = 100(x_1^2 - x_2^2) + (1 - x_1)^2$ | *f(0,0) = 0* |
| 3 | $f(x) = \sum_{1}^{5} \mathrm{int}(x_i)$ | *f(0, ..., 0) = 0* |
| 4 | $f(x) = \sum_{1}^{30} i x_i^4 + Gauss(0,1)$ | *f(0,..,0) = -2.5* |
| 5 | $f(x) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$ | *f(0,0) ≈ 1* |

**Tabla 5.1** Funciones de De Jong

`f1` es una función unimodal que es fácil de optimizar para un AG. `f2` es una función multimodal un poco más difícil. `f3` es una función discontinua en forma de escalera. `f4` tiene ruido y un espacio de búsqueda inmenso. `f5` tiene varios mínimos locales y su mínimo global se encuentra en 0.9980014, utiliza una matriz constante A:

$$A = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \ldots & 32 & 32 & 32 \end{pmatrix}$$

También se utilizaron tres funciones adicionales utilizadas por Mühlenbein para probar un sistema de algoritmos genéticos paralelos [MSB91] y que han sido posteriormente utilizadas por otros investigadores para probar sus propios programas. Las funciones de Mühlenbein son más complejas que las funciones de De Jong y se definen en la tabla 5.2.

| Número de Función | Función | Mínimo |
|---|---|---|
| 6 (Rastringin) | $f(x) = 100 + \sum_{i=1}^{10} x_i^2 - 10\cos(2\pi x_i)$ | *f(0,..0) = 0* |
| 7 (Schwefel) | $f(x) = \sum_{i=1}^{10} -x_i \, sin(\sqrt{\lvert x_i \rvert})$ | *f(420.82,...,420.82) = -4190* |
| 8 (Griewank) | $f(x) = \sum_{i=1}^{10} x_i / 4000 - \prod_{i=1}^{10} \cos(x_i / \sqrt{i}) + 1$ | *f(0,..,0) = 0* |

**Tabla 5.2** Funciones de Mühlenbein

`f6` es una función bastante difícil para los AGs por que tiene un espacio de búsqueda grande y es altamente multimodal. `f7` es un poco más sencilla, pero se caracteriza porque el segundo mejor mínimo se encuentra muy lejos del mínimo global y el AG puede quedar atrapado en una región equivocada. `f8` es la función más difícil porque el término del producto hace que las diez variables sean altamente dependientes entre sí.

### 5.1.2. Parámetros y codificación

Los parámetros del AG utilizados en todos los experimentos se presentan en la tabla 5.3. Se escogieron estos valores porque DeJong [DeJ75] encontró que funcionaban bien para resolver las funciones que propusó para probar AGs.

| Parámetro | Valor |
|---|---|
| Población (por nodo) | 100 |
| Probabilidad de cruce | 0.6 |
| Probabilidad de mutación | 0.001 |
| Brecha generacional | 1.0 |
| Ventana de escalamiento | 5 |
| Opciones[10] | cefglS |
| Semilla | 123456789 |

**Tabla 5.3** Parámetros del AG utilizados.

En la tabla 5.4 se presenta la codificación utilizada en cada función. Las columnas marcadas **mínimo** y **máximo** son los valores extremos que puede tomar cada variable. La columna marcada **valores** determina el numero de valores posibles que cada variable puede tomar, esto determina la longitud en bits necesaria para almacenar cada variable. Por ejemplo, la función `f1` usa tres variables, cómo cada variable puede tomar 1024 valores se necesitan 10 bits para cada una, por lo que cada individuo es de 30 bits.

| función | # de variables | mínimo | máximo | valores |
|---|---|---|---|---|
| f1 | 3 | -5.11 | 5.12 | 1024 |
| f2 | 2 | 2.047 | 2.048 | 4096 |
| f3 | 5 | -5.11 | 5.12 | 1024 |
| f4 | 30 | -1.27 | 1.28 | 256 |
| f5 | 2 | -65.535 | 65.536 | 131072 |
| f6 | 10 | -5.11 | 5.12 | 1024 |
| f7 | 10 | -500 | 500 | 1024 |
| f8 | 10 | -511 | 512 | 1024 |

**Tabla 5.4** Codificación de las variables

[10] Las opciones se explican en el Manual de Usuario. Brevemente: se `c`olectarán estadísticas, se utilizará reemplazo `e`litista, codificación de números de punto `f`lotante, códigos `G`ray, se usa bitácora (`l`ogfile) y la migración será `S`íncrona.

### 5.1.3. Topologías y parámetros de migración

En esta tesis no sólo se intenta probar que los AGPs tienen mejor desempeño que los AG seriales, sino que se pretende establecer que algunas topologías de comunicación entre subpoblaciones permiten obtener mejores resultados que otras.

La elección de las topologías puede estar basada en varios criterios como el número total de nodos, el número de vecinos de cada nodo y el diámetro de la gráfica.

En los experimentos se utilizaron 4 topologías: un anillo unidireccional con 16 nodos, un hipercubo de grado 4, una malla de 4 X 4 con conexiones entre los extremos de cada renglón y columna y una gráfica totalmente conectada de 5 nodos (ver figura 5.1). Estas topologías se eligieron porque son representativas de arquitecturas de computadoras paralelas existentes y además porque tienen ciertas características en común. El anillo, la malla y el hipercubo tienen 16 nodos en total. Cada nodo en el hipercubo, la malla y la topología totalmente conectada tiene cuatro vecinos.

Todos los nodos manejan el mismo número de individuos. Las topologías con más nodos tienen una población más grande, por lo que permiten muestrear con más detalle el espacio de búsqueda del problema. La topología completa, por tener una población menor, muestrea menos puntos pero probablemente converja más rápidamente [11].

Los intervalos de migración que se probaron fueron 5, 10, 20 y 50 generaciones. Las tasas de migración utilizadas fueron 0.05, 0.10, 0.20 y 0.50.

[11] En la figura 3.5 se presenta una gráfica comparando la convergencia de poblaciones de distinto tamaño.

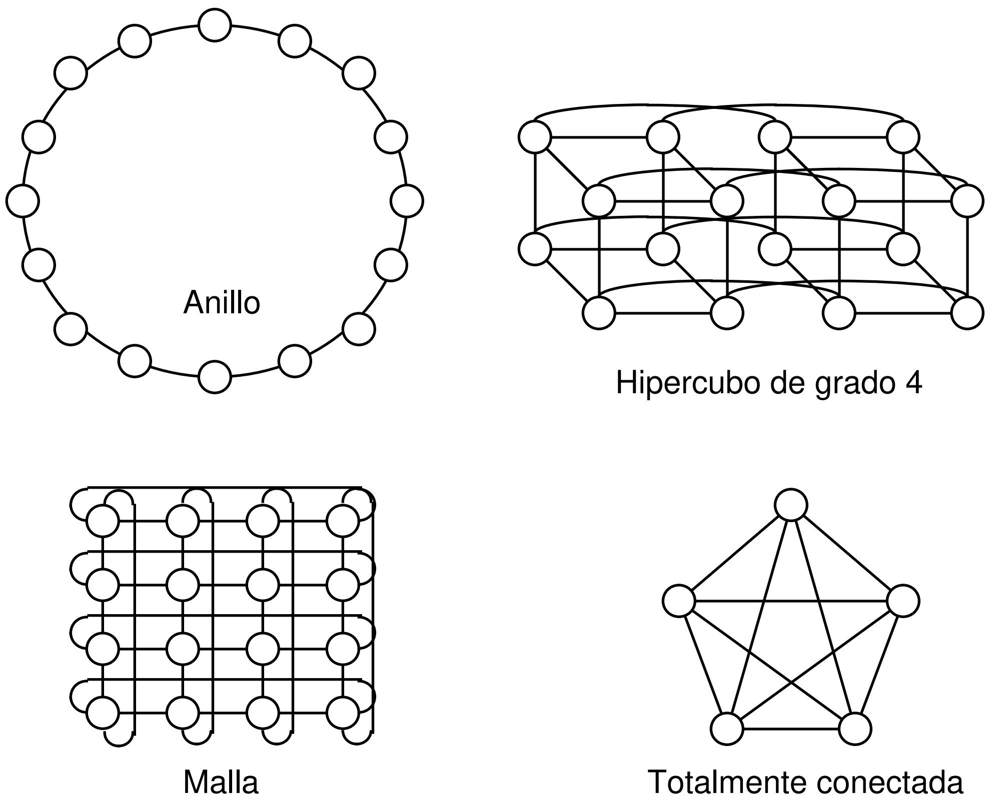



**Figura 5.1** Topologías utilizadas

## *5.1.4. Conjunto de pruebas*

En las pruebas se utilizaron las ocho funciones descritas anteriormente. Se probó cada función con cuatro topologías y para cada topología se probaron 16 combinaciones de parámetros de migración. En total se efectuaron 512 experimentos.

Lo que se pretende estudiar con los experimentos es el efecto de la migración sobre la calidad y la eficiencia de la búsqueda del AGP. Se mantuvieron constantes los parámetros del algoritmo genético variando únicamente la topología utilizada, el intervalo y la tasa de migración.

Las funciones se consideran solucionadas cuando todos los nodos alcanzan el óptimo especificado en las tablas 5.1 y 5.2 con 4 dígitos decimales. Cómo `f4` tiene ruido, se considera solucionada cuando el promedio de los mejores individuos de cada nodo sea igual o menor a -2.5 (Este mismo criterio se utiliza en [GoWh93]).

## *5.1.5. Medida de desempeño*

Existen varios criterios que se pueden utilizar para medir el desempeño de los AGs y AGPs. En esta tesis se consideraron como candidatos los siguientes criterios:

1. El número de evaluaciones (totales y/o por nodo) necesarias para encontrar el resultado óptimo.

2. El número de generaciones (totales y/o por nodo) necesarias para encontrar el resultado óptimo.

3. El promedio de la población después de un número determinado de evaluaciones.

4. El tiempo necesario para encontrar el resultado óptimo.

Para decidir cuáles de estos criterios eran adecuados para comparar el desempeño de los AGPs contra AGs seriales se realizaron los experimentos descritos en la sección anterior, se graficaron[12] los criterios candidatos y se analizaron las gráficas para determinar si existía algún(os) criterio(s) preferible(s) a los demás. Se intentó descubrir cual(es) de los criterios candidatos mostraba más claramente el efecto de los parámetros de migración. En cada función, se tomaron los resultados de cada topología y se graficaron los valores obtenidos en cada combinación de parámetros de migración.

Las gráficas de evaluaciones y de generaciones tienen aproximadamente la misma forma en todos los casos. Esto ocurre porque el número de individuos nuevos por generación es poco variable (en una generación sólo se evalúa a los individuos nuevos). Se prefirió tomar como medida de desempeño el número de evaluaciones porque reflejan más directamente el consumo de CPU y es más fácil comparar los AGPs con otros métodos de optimización.

Por otro lado, las gráficas de los promedios de la población revelaban que los parámetros de migración y la topología afectaban muy poco al promedio de la población, y no se pudo observar ninguna correlación entre los parámetros y los valores obtenidos. Se decidió no utilizar el promedio de la población como un criterio para comparar AGs seriales y AGPs.

La comparación del tiempo de ejecución entre un algoritmo distribuido y uno serial es una estadística muy importante. Desgraciadamente se tuvo que abandonar este criterio por varias razones. En primer lugar las condiciones en las que se realizaron los experimentos no permitían obtener resultados confiables: las computadoras tenían cargas de trabajo variables y la red estaba permanentemente en uso por una cantidad también variable de procesos. Por otro lado, hay que recordar que se está simulando una arquitectura paralela sobre un medio de transmisión lento[13] y compartido por todos los procesos del experimento y por otros procesos ajenos. Esto ocasiona que exista una competencia muy grande por accesar la red, lo que no ocurriría en máquina con conexiones punto a punto entre los procesadores. Para poder medir la ganancia en tiempo que se obtiene usando AGPs sobre AGs se deben hacer implementaciones en máquinas paralelas reales.

---

[12] Se automatizó este proceso usando `awk` y `gnuplot`.

[13] Las pruebas se realizaron en una red local Ethernet con un ancho de banda de 10 Mbps que se comparte por todas las computadoras en el segmento. El ancho de banda disponible en una computadora paralela es mucho mayor; por ejemplo, en una red de transputers T9000 cada transputer tiene cuatro ligas bidireccionales de 100Mbps y en algunas computadoras que utilizan un bus para comunicar a los procesadores este tiene un ancho de banda de alrededor de 300Mbps.

Por todo lo anterior, la única medida de desempeño que se utilizará en este trabajo son las evaluaciones necesarias para obtener el óptimo global.

### 5.1.6. Datos de Referencia

Para determinar si se mejora el desempeño al utilizar varias subpoblaciones, los resultados obtenidos por un AGP deben ser comparados con los obtenidos por un AG serial. Los datos de referencia se obtuvieron usando DGENESIS con un sólo proceso. Estos datos son idénticos a los que se obtienen con GENESIS.

Para poder comparar los datos que se obtuvieron en los experimentos distribuidos con los que se obtuvieron por el AG serial es necesario que se utilicen los mismos parámetros en todos los casos. Todos los parámetros en el AG serial y en el AGP son iguales (ver tabla 5.4), excepto el tamaño de la población.

Hay que recordar que poblaciones de distinto tamaño convergen a distintas velocidades (ver figura 3.5). Cada nodo en un experimento distribuido tiene una población de 100 individuos. Para comparar el número de evaluaciones *por nodo* se utiliza el resultado obtenido por el AG serial con una población de 100. Cuando se comparan las evaluaciones *totales* de cada topología con los resultados obtenidos por un AG serial, se usan los resultados obtenidos con poblaciones de 500 o de 1600 individuos, dependiendo del caso. De esta manera, las comparaciones son justas porque el AG y los AGPs manipulan poblaciones de tamaños iguales y sus velocidades de convergencia deben ser similares.

## 5.2. Resultados

Los experimentos se realizaron utilizando varias máquinas disponibles en el ITAM. DGENESIS se configuró para intentar balancear la carga de trabajo de las máquinas, especificando en el archivo `usr/GAhosts` (explicado en la sección 4.2.4) que se deseaba un máximo de cuatro procesos en cada máquina. En las topologías de 16 nodos algunas máquinas procesaban 3 subpoblaciones y otras 4.

En esta sección se presentará un breve análisis de los resultados obtenidos en las pruebas descritas anteriormente. Los resultados completos se presentan en el Apéndice I.

### 5.2.1. Resultados generales

En general se observó que los parámetros de migración afectan de manera importante el desempeño del AGP. En la figura 5.2 se presenta la gráfica de evaluaciones obtenida para la función f1 utilizando la topología completa y se aprecia el efecto de los parámetros de migración de manera muy notable. El primer número en las etiquetas del eje horizontal es la tasa de migración (expresado como porcentaje del tamaño de la subpoblación) y el segundo número es el intervalo de migración. El número menor de evaluaciones (1831) se encuentra cuando el 20% de la población migra cada 5 generaciones; el máximo de evaluaciones (5024) se requiere cuando migra el 5% cada 40 generaciones.

En casi todos los casos, los mejores resultados se encontraron utilizando un intervalo de migración de 5 generaciones y en general se requieren más evaluaciones conforme aumenta el intervalo. Esto soporta la idea de que la migración es importante y que el aislamiento retarda el encuentro del óptimo global. La tasa de migración parece no tener un efecto tan importante sobre el número de evaluaciones necesarias, por lo que para ahorrar recursos se puede utilizar una tasa moderada.

Entre las topologías que se probaron, la más alentadora es la completa mientras que la que requiere de más evaluaciones es el anillo. Las topologías de malla y el hipercubo requieren de aproximadamente las mismas evaluaciones, excepto en el caso de `f4`.

Se encontraron resultados que muestran que algunas combinaciones de topologías y parámetros de migración requieren más evaluaciones que un AG serial. La determinación de la clase de problemas y de los parámetros en dónde se requieren más evaluaciones queda fuera del alcance de esta tesis. Sin embargo, se observó que frecuentemente se requieren más evaluaciones cuando se utiliza la topología de anillo.

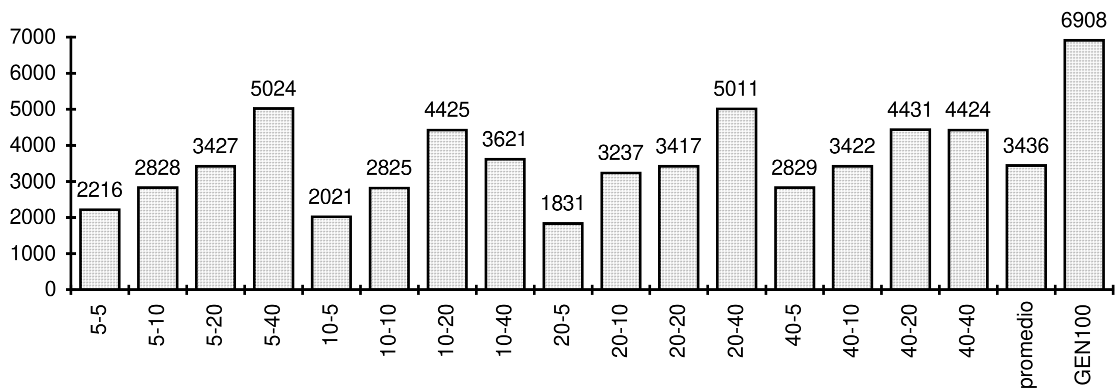



**Figura 5.2.** Evaluaciones por nodo para f1 usando una topología completa. En el eje horizontal **x - y** significa que el **x**% de la población migra cada **y** generaciones.

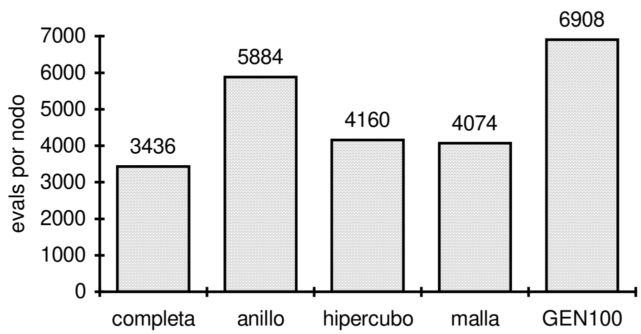
A continuación se presentan los resultados obtenidos para cada función utilizando diferentes topologías. Para cada función se presentan dos gráficas. La primera gráfica presenta el numero promedio de las evaluaciones *por nodo* que requiere cada topología para alcanzar el óptimo global en las 16 combinaciones de parámetros de migración. La segunda gráfica presenta el número *total* de evaluaciones requerido por cada topología para alcanzar el óptimo global. El total de evaluaciones se calculó multiplicando el promedio del número de evaluaciones por nodo (representado en la primera gráfica) por el número de nodos de la topología.

Al promediar los 16 resultados obtenidos en cada topología con distintos parámetros de migración, se obtiene una medida que permite comparar el efecto de las topologías.

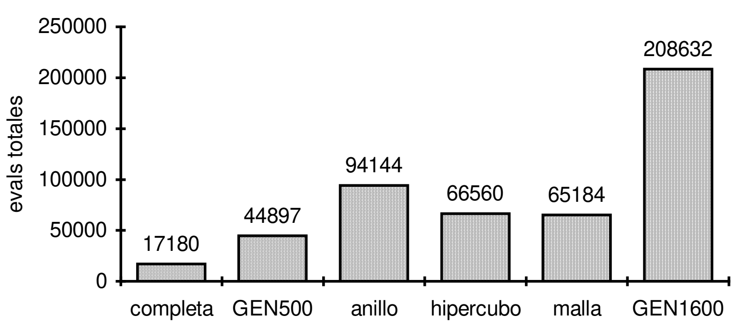
El número de evaluaciones por nodo presenta una idea muy precisa del tiempo de ejecución si se tiene una implementación paralela de un AGP, sobre todo cuando el tiempo de procesamiento es grande comparado con el tiempo dedicado a la comunicación entre nodos. Si se utiliza una máquina serial para simular un AGP, entonces el número total de evaluaciones es más importante.

## 5.2.2. Función f1

En esta función, el número de evaluaciones por nodo en el AG serial es casi el doble de las requeridas por la topología completa. La topología de anillo es la más cercana al resultado del AG serial. Con estos resultados se empieza a manejar la hipótesis de que la topología completa es la más eficiente de todas y que la de anillo es la menos eficiente.



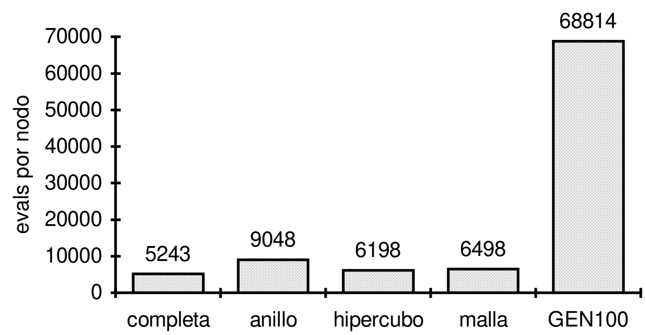

Nótese que en la gráfica anterior se presenta el resultado obtenido por un AG serial con una población de 100 individuos. En la siguiente gráfica se presentan los números totales de evaluaciones necesarios en cada topología y los resultados del AG serial con poblaciones de 500 y 1600 individuos. La topología completa se debe comparar con la población de 500, las demás topologías con la población de 1600.



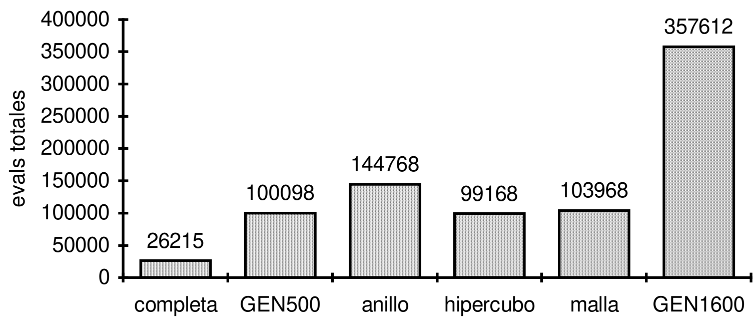

Cuando se comparan las evaluaciones totales se obtienen resultados muy impresionantes. En la gráfica anterior se puede observar que la topología completa requiere de sólo la tercera parte de las evaluaciones que el AG serial necesita con la población de 500. Las topologías con 16 nodos también logran resultados muy impresionantes: el hipercubo y la malla alcanzan el óptimo global con menos de la tercera parte de evaluaciones que el AG serial, el anillo necesita un poco menos de la mitad.

## 5.2.3. Función f2

En esta función se logran los resultados más impresionantes en las dos comparaciones. En la primera gráfica se puede apreciar que el número de evaluaciones por nodo es aproximadamente la décima parte de las que necesita el AG serial.

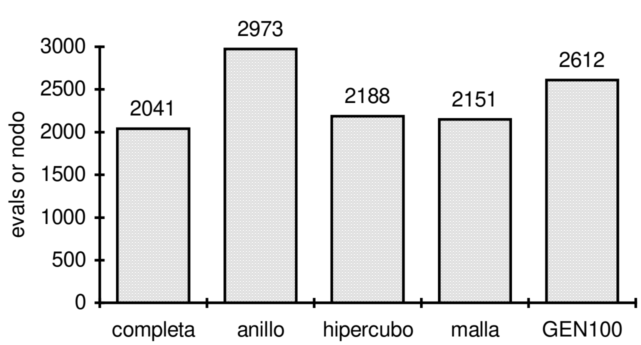



Respecto a las evaluaciones totales, la topología completa requiere de alrededor de la cuarta parte que el AG serial con 500 individuos. Las demás topologías requieren alrededor de la tercera parte que el AG con 1600 individuos.

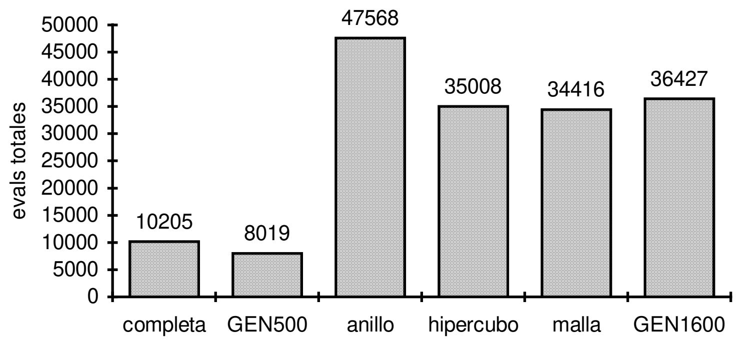



## 5.2.4. Función f3

En los resultados de esta función se puede apreciar que no siempre es conveniente utilizar AGPs y se refuerza la hipótesis de que la topología de anillo es la que requiere de más evaluaciones. Hay que recordar que los datos de estas gráficas son los promedios de los resultados obtenidos con 16 combinaciones de parámetros de migración. En algunas de estas combinaciones los resultados del AGP son mejores que el AG serial, especialmente en la topología completa. (Ver los datos completos en el Apéndice I)



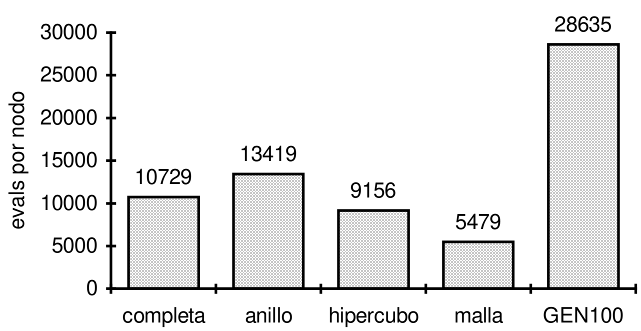
Si tuviéramos una implementación en paralelo del AGP, la gráfica anterior nos indica que usando tres topologías (completa, hipercubo y malla) el desempeño sería ligeramente mejor que un AG serial. Sin embargo, la siguiente gráfica nos dice que no es muy conveniente simular un AGP en una máquina serial para optimizar esta función.



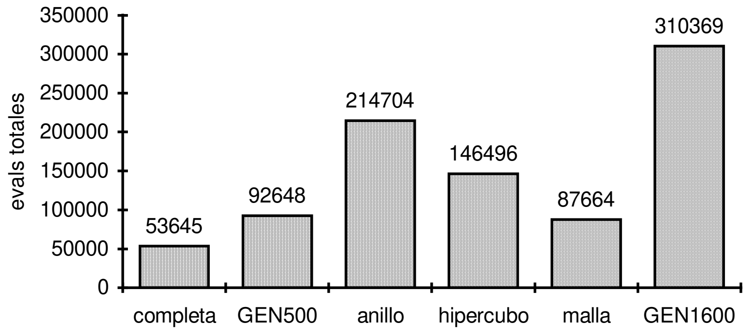

Los resultados obtenidos en esta función apoyan la hipótesis de Mühlenbein [Müh91] de que quizás es más conveniente utilizar AGPs en funciones continuas que en discontinuas.

### *5.2.5. Función f4*

Esta función se consideró optimizada cuando se alcanzó el valor de -2.5. Esta es la única función en que el hipercubo y la malla tuvieron en promedio menos evaluaciones por nodo que la topología completa. Quizás esto se debe a que estas topologías manejan un tamaño de población mayor y esto permite que muestreen espacios con ruido más eficientemente.

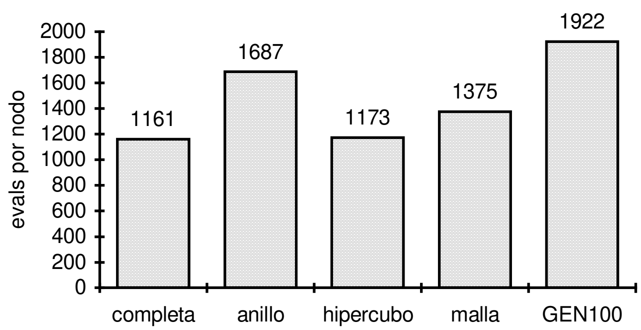



Nótese que la diferencia de evaluaciones (por nodo y totales) entre el hipercubo y la malla es de casi el doble cuando en la mayoría de los casos son muy similares.

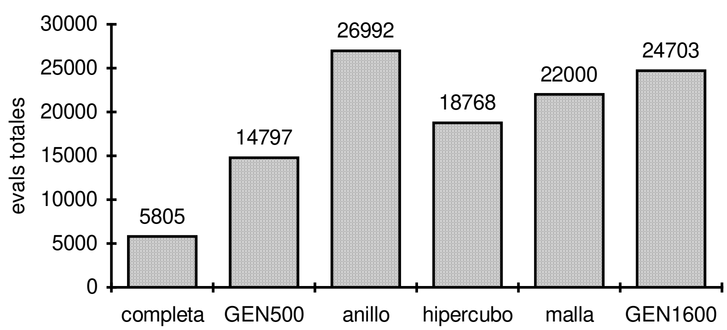



## 5.2.6. Función f5

En esta función el anillo sigue necesitando del mayor número de evaluaciones y se mantiene la superioridad de la topología completa. Las diferencias entre los resultados de los experimentos distribuidos y del AG serial son bastante grandes (excepto el anillo), pero menos acentuadas que en otras funciones.

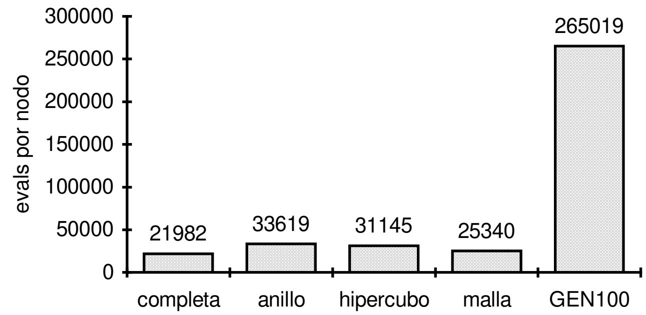



Comparando las evaluaciones totales, se puede observar que sólo la topología completa tiene resultados significativamente favorables. Las demás topologías requieren de casi el mismo número de evaluaciones que GENESIS con 1600 individuos e incluso el anillo requiere de más.

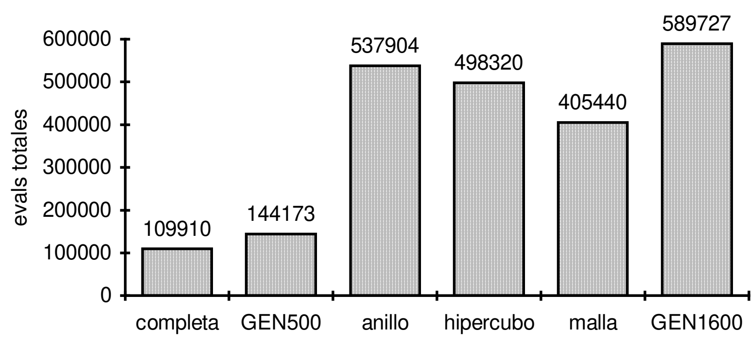


### 5.2.7. Función f6

En esta función hay una enorme diferencia entre las evaluaciones por nodo de los experimentos distribuidos y de las evaluaciones del AG serial. GENESIS con una población de 100 individuos no encontró el óptimo global después de mas de 265,000 evaluaciones, pero se acerca mucho encontrando uno de los subóptimos. Los subóptimos se encuentran en *f6(0,0,...,1,..0)=1*.

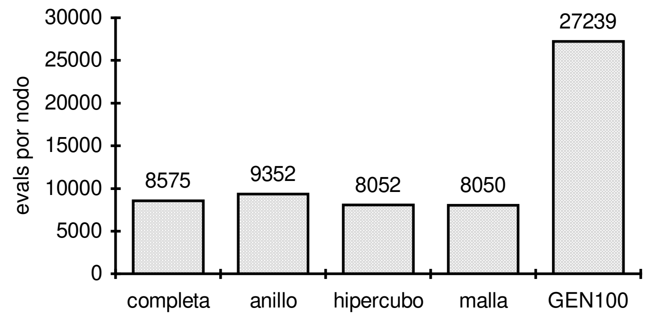



GENESIS encontró el óptimo global con poblaciones de 500 y de 1600 individuos. El número total de evaluaciones es favorable a los experimentos distribuidos, aunque no hay una diferencia muy grande.

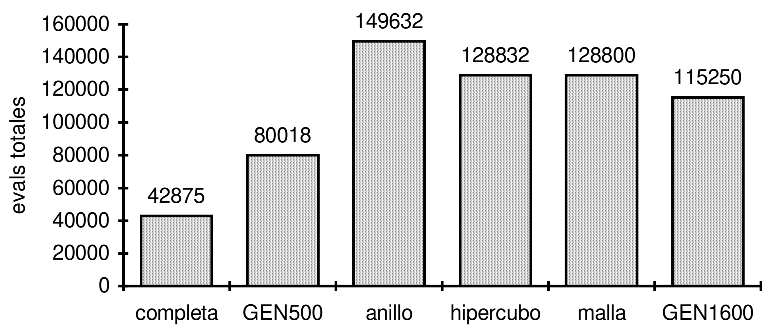



### 5.2.8. Función f7

Todas las topologías requirieron casi el mismo número de evaluaciones por nodo y fueron alrededor de la tercera parte de las que necesitó el AG serial con la población de 100.




Al comparar las evaluaciones totales, se puede observar que sólo la topología completa necesita de menos evaluaciones que el correspondiente AG serial. En esta función, el anillo también resulta ser la topología que requiere de más evaluaciones.




## 5.2.9. Función f8

Esta es la función más difícil de optimizar de las que se utilizaron en estas pruebas. Después de 100,000 evaluaciones por nodo, solo algunos experimentos lograron alcanzar el óptimo que se encuentra en 0, aunque casi todos encontraron puntos menores a 0.1.

Después de 750, 000 evaluaciones los algoritmos seriales no pudieron alcanzar el óptimo, aunque el AG serial con la población de 100 individuos alcanzó un punto muy cercano al óptimo:

| Experimento | Mejor valor encontrado | Evaluaciones |
|---|---|---|
| GEN100 | 0.0321 | 621029 |
| GEN500 | 0.1942 | 732118 |
| GEN1600 | 0.5048 | 696377 |

Los experimentos exitosos se presentan en el Apéndice I. Brevemente, en la topología completa y en el hipercubo solo 2 experimentos lograron alcanzar el óptimo, el anillo fue la topología con mayores éxitos con 6 y la malla alcanzó el óptimo en 5 experimentos.

# 5.3. Resultados de múltiples experimentos

Los resultados presentados en la sección anterior fueron obtenidos ejecutando solamente un experimento para cada combinación de parámetros. Esto permite dar una idea de la influencia de los parámetros en el comportamiento del AGP, pero no permite un análisis formal de los resultados.

En esta sección se muestran los resultados que se obtuvieron al utilizar una muestra de 50 experimentos para cada combinación de parámetros. En la tabla 5.5 se presentan el promedio y la desviación estándar del número de evaluaciones por nodo que se requieren para alcanzar el óptimo en las funciones f1 y f6 utilizando la topología totalmente conectada. En las figuras 5.3 y 5.4 se grafican las medias obtenidas.

| | | f1 | | f6 | |
|---|---|---|---|---|---|
| intervalo de migración | tasa de migración | media | desviación estándar | media | desviación estándar |
| 5 | 5 | 2459 | 402 | 2210 | 301 |
| 10 | 5 | 3234 | 570 | 2740 | 593 |
| 20 | 5 | 4139 | 724 | 3505 | 471 |
| 40 | 5 | 4781 | 634 | 4631 | 717 |
| 5 | 10 | 2294 | 339 | 2139 | 603 |
| 10 | 10 | 3075 | 516 | 2687 | 377 |
| 20 | 10 | 3883 | 600 | 3391 | 402 |
| 40 | 10 | 4891 | 727 | 4331 | 664 |
| 5 | 20 | 2557 | 380 | 2188 | 323 |
| 10 | 20 | 4183 | 680 | 3327 | 508 |
| 20 | 20 | 4170 | 614 | 3431 | 492 |
| 40 | 20 | 4982 | 642 | 4187 | 255 |
| 5 | 40 | 2788 | 401 | 2334 | 352 |
| 10 | 40 | 3348 | 510 | 2896 | 466 |
| 20 | 40 | 4429 | 705 | 3544 | 614 |
| 40 | 40 | 4904 | 641 | 4291 | 452 |

**Tabla 5.5** Evaluaciones por nodo de f1 usando la topología completa (50 experimentos).

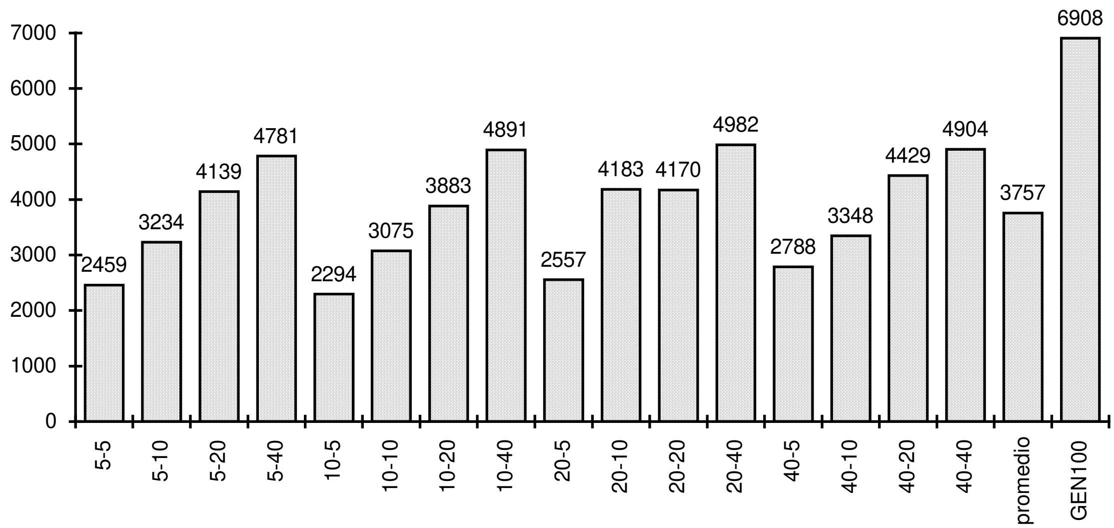



**Figura 5.3** Evaluaciones por nodo promedio de f1 usando la topología completa (50 experimentos).

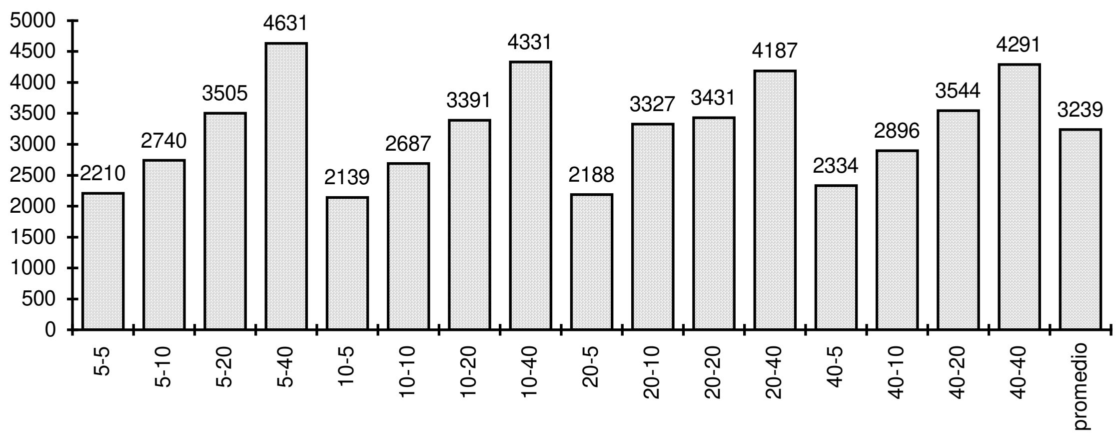



**Figura 5.4** Evaluaciones por nodo promedio de f6 usando la topología completa (50 experimentos).

Se puede observar que las gráficas anteriores conservan la misma forma que la gráfica de la figura 5.2 que presenta los resultados de un solo experimento. Los experimentos con un intervalo de migración de 5 generaciones son los que requieren de menos evaluaciones para alcanzar el óptimo global y el número de evaluaciones aumenta conforme crece el intervalo de migración.

También se realizaron experimentos para comparar los resultados obtenidos por diferentes topologías utilizando los mismos parámetros de migración. Se continúan utilizando las funciones f1 y f6, el intervalo de migración se fijó en cinco generaciones y la tasa de migración en 5%. En la tabla 5.6 se reportan el promedio y la desviación estándar de las evaluaciones por nodo requeridas para alcanzar el óptimo. En las figura 5.5 y 5.6 se grafican las medias.

| topología | media | desviación estándar |
|---|---|---|
| completa | 2459 | 402 |
| anillo | 4239 | 434 |
| hipercubo | 2450 | 295 |
| malla | 2390 | 250 |

**Tabla 5.6** Evaluaciones por nodo de f1 usando diferentes topologías (50 experimentos).

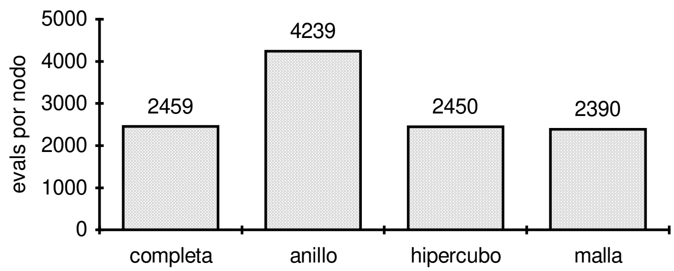



**Figura 5.5** Evaluaciones por nodo de f1 usando diferentes topologías (50 experimentos).

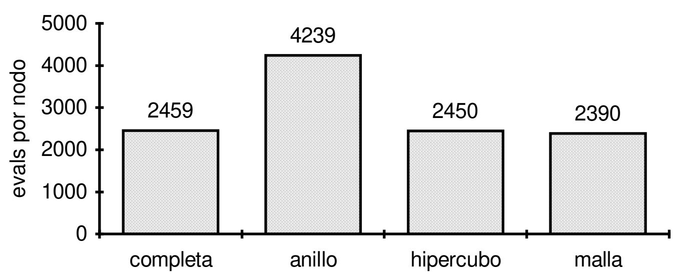



**Figura 5.6** Evaluaciones por nodo de f6 usando diferentes topologías (50 experimentos).

El anillo es la topología que requiere de más evaluaciones y la completamente conectada, el hipercubo y la malla requieren aproximadamente el mismo número de evaluaciones.

El análisis de los datos anteriores permite especular que el número de vecinos de cada nodo es una característica de la topología que afecta al desempeño del algoritmo. El anillo sólo tiene 1 vecino y requiere del mayor número de evaluaciones, mientras que las demás topologías tienen 4 vecinos y requieren de un número muy similar de evaluaciones.

Se requieren de otros experimentos para poder comprobar la hipótesis anterior, y también para tratar de determinar qué otras características de las topologías tienen algún efecto sobre el algoritmo.

*Capítulo 6*

# *Conclusiones y Líneas Futuras*

Con el trabajo desarrollado en esta tesis se lograron obtener algunas conclusiones interesantes. Los objetivos propuestos en el primer capítulo se cumplieron satisfactoriamente, pero existen muchas maneras de ampliar este proyecto. Este último capítulo presenta las conclusiones que se obtuvieron y algunas propuestas para extender el trabajo realizado.

## 6.1. Conclusiones

Los algoritmos genéticos son considerados un método robusto de solución de problemas y su popularidad ha crecido significativamente en los últimos años. La aplicabilidad de AGs en campos diversos ha sido ampliamente demostrada por los trabajos de muchos investigadores alrededor del mundo. La participación de estos investigadores en conferencias, mesas de trabajo y simposia de Inteligencia Artificial, Robótica y diversas áreas de Computación Evolutiva ha permitido la difusión de sus trabajos y sus resultados.

Los AGPs han recibido una atención especial en los últimos años, aunque algunos conceptos de AGPs se manejaban desde hace mucho tiempo. Este interés creciente probablemente se debe a dos factores:

1. El intento de explotar eficientemente las arquitecturas de las computadoras paralelas disponibles actualmente.

2. La capacidad de los AGPs de encontrar soluciones de la misma calidad o mejores que los AGs convencionales en menos tiempo.

Un objetivo de este trabajo es comparar diferentes topologías de comunicación para determinar si tienen algún efecto sobre la eficiencia y la eficacia de los AGPs. Para conseguir esto se siguió un enfoque experimental y esto requería de una herramienta adecuada.

A pesar de que actualmente existen bastantes herramientas para experimentar con AGs, existen pocas que permiten experimentar con AGPs. Las que existen tienen básicamente tres problemas:

1. Fueron diseñadas para explotar la arquitectura de una computadora específica. Esto las hace muy eficientes, pero poco prácticas (este es el caso de PARAGENESIS [van92]).

2. Tienen un criterio de migración poco flexible. Por ejemplo, en cada generación mandan un individuo de cada población a todas las demás y no existe ninguna forma para cambiar el criterio de migración (e.g. PGA 2.4 [BaRo93]).

3. No están disponibles. Aunque muchos investigadores hacen pública una edición electrónica de sus artículos y resultados, muy pocos liberan los programas que utilizaron. Esto es grave, porque hace prácticamente imposible repetir los resultados que se presentan y se tiene que recorrer un camino muy largo para intentar mejorarlos y/o explicarlos.

Estos problemas motivaron la creación de una herramienta que permitiera simular cualquier topología de comunicación con criterios de migración flexibles y que utilizara recursos que se utilizan frecuentemente en universidades o centros de investigación. Con la construcción de esta herramienta quedó cumplido satisfactoriamente el primer objetivo de esta tesis, aunque todavía hay muchas mejoras posibles (ver sección 6.2.1 más adelante).

Existe un documento en USENET que presenta respuestas a preguntas frecuentes (FAQ) [Hei94]. En este documento se incluye una sección de herramientas de AGs que sustituye al documento que mantenía Nicol Schraudolph [Sch93]. DGENESIS está incluido en este documento y hasta el momento se ha tenido contacto con investigadores en E.U., Portugal y Alemania interesados en usar el sistema. DGENESIS estará disponible por ftp anonimo en `lamport.rhon.itam.mx` y en un repositorio de software del Instituto Santa Fe (`ftp.santafe.edu`).

Una vez que se contaba con la herramienta adecuada, se procedió a realizar una serie de experimentos diseñados para obtener datos que permitieran apreciar el efecto de los parámetros de migración sobre el desempeño del AGP. Los resultados obtenidos permiten asegurar que la migración afecta significativamente al desempeño del AGP. Con un simple vistazo a los resultados del Apéndice I, se puede apreciar el enorme efecto que tienen los parámetros de migración sobre el desempeño del algoritmo.

Los mejores resultados se obtienen con un intervalo de migración de 5 generaciones y el número de evaluaciones necesarias crece conforme el intervalo se hace mayor. La tasa de migración parece ser menos importante por lo que para aprovechar mejor los recursos de comunicación se recomienda usar tasas moderadas (5%, 10%). Estos resultados parecen indicar que el aislamiento de las subpoblaciones no da buenos resultados y que lo mejor es tener un intercambio frecuente de individuos valiosos.

Con los resultados obtenidos en los experimentos también se puede concluir que la topología de comunicación entre subpoblaciones tiene un efecto importante sobre el algoritmo. Se logró observar que el número de evaluaciones que se requieren para encontrar el punto óptimo es diferente en cada topología. Entre las topologías que se probaron, la totalmente conectada con cinco nodos es la que en promedio requiere de menos evaluaciones (totales y por nodo), mientras que el anillo unidireccional de 16 nodos resultó ser la topología menos eficiente.

Los datos que se obtuvieron al utilizar los mismos parámetros de migración en diferentes topologías (tabla 5.6) parecen indicar que el número de nodos totales no afecta tanto al desempeño del AGP como el número de vecinos de cada nodo. Los nodos del anillo tienen solamente un vecino y esta topología requiere de más evaluaciones que las demás. Las demás topologías probadas tienen el mismo número de vecinos y requieren de aproximadamente el mismo número de evaluaciones.

Estos resultados son muy importantes porque implican que se necesitan relativamente pocos recursos para implementar eficientemente en hardware un AGP que sea altamente confiable. La topología completa sólo tiene cinco nodos y parece ser la más recomendable, mientras que las demás topologías tienen 16 nodos. Menos nodos significan menos procesadores y esto, en muchos casos, implica un costo menor. Se requiere comunicación frecuente entre los nodos, pero no muy abundante, por lo que las ligas que comunican los procesadores no necesitan tener un ancho de banda muy grande.

En los experimentos que se realizaron, se encontró que los AGPs requieren de menos evaluaciones totales que un AG serial con una población de tamaño equivalente. Utilizar poblaciones grandes es muy conveniente porque permiten explorar mejor el espacio de búsqueda de un problema, pero el número de evaluaciones que se requiere aumenta considerablemente. En un AGP es posible aprovechar las ventajas de utilizar una población grande utilizando menos evaluaciones que en un AG serial y con la ventaja adicional de que las evaluaciones se hacen en paralelo.

# 6.2. Líneas futuras

El trabajo desarrollado para esta tesis es bastante amplio en muchos sentidos y los objetivos propuestos se cumplieron satisfactoriamente. Sin embargo, todavía hay mucho trabajo por hacer para mejorar el sistema que se desarrolló y también es necesario realizar experimentos distintos para tratar de mejorar los resultados que se obtuvieron. Probablemente el trabajo más importante que es necesario realizar en el futuro es una explicación formal del comportamiento de los AGPs.

## *6.2.1. DGENESIS*

DGENESIS es actualmente un sistema de experimentación muy flexible, pero se puede extender en muchos sentidos para hacerlo más poderoso. Algunas de las extensiones que se proponen en esta sección son bastante sencillos, pero algunos requieren de cambios bastante profundos y se proponen para realizarse a largo plazo.

### *6.2.1.1. Operadores genéticos*

DGENESIS heredó los operadores genéticos de GENESIS, que son bastante poderosos, pero para integrar un paquete de experimentación más flexible es necesario agregar otros operadores como:

- Cruce en varios puntos. El uso de varios puntos de cruce todavía es discutido y sería conveniente tener una herramienta que permita establecer el número de puntos de cruce.

- Tasa de mutación adaptativa. La tasa de mutación puede depender de alguna estadística que refleje el grado en que la población ha convergido. Esto es otra manera de romper el equilibrio de una población, pero quizás de una manera poco natural. Ciertamente, en la naturaleza la mutación es más probable en algunos genes que en otros, pero esto no depende del estado de la población. Sería interesante determinar si utilizando migración se puede mantener la tasa de mutación baja y poco variable para "probar" que no es necesario recurrir a mecanismos tan artificiales para romper el equilibrio de una población.

- Algún operador de reorden. Existen algunos operadores genéticos que cambian de posición subcadenas dentro de un cromosoma (e.g. inversión [Gol89]). Estos operadores se propusieron hace bastante tiempo, pero no existen muchos programas que los utilicen porque la efectividad de los AGs sin estos operadores es muy buena y su implementación requiere recursos adicionales.

Algunos de los operadores discutidos son fáciles de agregar y de hecho han sido agregados a GENESIS (e.g. ver [Bäc92]), pero otros (como los operadores de reorden) requerirían una recodificación extensiva.

Un operador muy poco común y muy interesante permite cambiar dinámicamente la codificación de las variables de un problema utilizando un cromosoma de longitud constante. Este operador es llamado *Dynamic Parameter Encoding* (DPE) [ScBe92] y ha sido implementado en GAucsd [ScGr92]. Los resultados obtenidos son muy prometedores y sería muy interesante verlo en alguna versión futura de DGENESIS.

#### *6.2.1.2. Interfaz con el usuario y herramientas de visualización*

Actualmente el usuario se comunica con DGENESIS utilizando una interfaz basada en texto. Se podría hacer una interfaz gráfica más amigable para especificar los parámetros del sistema utilizando X Window o, mejor aún, Motif.

Es importante poder visualizar de alguna manera la composición de la población a través del tiempo para poder estudiar mejor el efecto de distintos operadores genéticos sobre la población. En DGENESIS sería especialmente interesante observar el efecto de la migración. Tanese [Tan89] y Bäck [Bäc92] crearon herramientas que pueden ser útiles, pero que requieren modificaciones para poder ser utilizadas por DGENESIS.

Las herramientas de visualización mencionadas se utilizan al terminar la ejecución del algoritmo, pero también se puede pensar en una herramienta que permita monitorear al algoritmo *mientras* se está ejecutando.

#### *6.2.1.3. Migración*

Actualmente en DGENESIS sólo los *mejores* individuos de cada subpoblación son considerados para migrar. Se envían *copias* de estos individuos a las demás subpoblaciones en intervalos *fijos* utilizando la información contenida en unas tablas de migración.

Se pueden pensar muchas variaciones de este mecanismo. Por ejemplo, un criterio distinto para seleccionar a los individuos que migran puede ser sencillamente elegirlos aleatoriamente de la población. También sería interesante ver el efecto que tendría eliminar de la subpoblación original a los individuos que migren, para que en un momento dado sólo exista una copia de cada uno. La migración sería más realista y con las herramientas adecuadas quizás se podrían observar algunos fenómenos interesantes.

Algunos investigadores han propuesto métodos para hacer más flexible la migración (e.g. ver [MTS93]). En lugar de utilizar intervalos fijos para migrar, proponen utilizar algunas estadísticas acerca de la convergencia de la población para tomar la decisión de migrar o no. La migración se lleva a cabo cuando se detecta que hace falta diversidad en la población. El argumento parece bueno, pero su efectividad es discutible, porque la convergencia de la población puede ser deseable algunas veces. De cualquier manera sería interesante poder hacer experimentos utilizando criterios de migración distintos.

#### 6.2.1.4. Portabilidad

DGENESIS se diseñó e implementó teniendo siempre en cuenta la cuestión de la portabilidad a diversos sistemas UNIX. Hasta ahora DGENESIS ha sido probado exitosamente en los siguientes plataformas:

- DECstation 5000 / 25 con Ultrix 4.2
- MicroVax 3100 con Ultrix 4.1
- PC con 386BSD 0.1
- Sun con SunOS 4.1

En general, no deben haber muchos problemas para portar DGENESIS a otros sistemas UNIX que soporten los protocolos TCP/IP.

Se pueden utilizar librerías que implementan TCP/IP en DOS y en Windows para portar DGENESIS a estos ambientes y así liberar el requerimiento actual de utilizar sistemas UNIX que no siempre están disponibles o que (muy) ocasionalmente tienen una carga de trabajo pesada.

Como DGENESIS está planeado para ser ejecutarse en una red local, no se establece ningún protocolo que asegure que los mensajes que se intercambien entre los procesos lleguen correctamente a su destino. En un futuro se puede utilizar otro protocolo de comunicación que asegure la entrega correcta de mensajes (e.g. TCP) o se puede construir un protocolo propio sobre UDP.

A largo plazo, tal vez resulte conveniente una recodificación de DGENESIS en algún lenguaje orientado a objetos. Es bastante natural pensar en objetos para representar a los individuos, a la población y a un AG.

También a largo plazo, sería sumamente interesante ejecutar AGPs en una arquitectura paralela real (e.g. una red de transputers) para poder comprobar los resultados obtenidos en las simulaciones y para poder recolectar estadísticas confiables sobre el tiempo de ejecución.

#### 6.2.1.5. Optimización

Hay dos aspectos que se pueden cambiar en DGENESIS para mejorar el desempeño del sistema:

1. Tratar de reducir la relación entre el tiempo dedicado a la comunicación entre procesos y el tiempo dedicado al procesamiento.

2. La distribución de trabajo entre los nodos participantes.

Durante la migración se envía a un individuo a la vez y esto permite mucha flexibilidad en el tamaño de los cromosomas, pero normalmente se utilizan cromosomas pequeños (menores a 200 caracteres) y la red se inunda de mensajes pequeños. Esto no permite aprovechar adecuadamente el ancho de banda de la red, porque cada paquete contiene información de direccionamiento, de detección de errores, etc. Si se envían mensajes más grandes probablemente se aprovecharía mejor la capacidad de la red.

En una red generalmente hay máquinas más rápidas que otras y para obtener un aprovechamiento óptimo de los recursos se debe equilibrar la carga de trabajo de manera que las máquinas más rápidas no pasen un tiempo significativo esperando sincronizarse con las máquinas lentas. En DGENESIS el problema de distribuir la carga de trabajo se resuelve de una manera bastante primitiva dejando en manos del usuario la decisión del número de procesos que se ejecutan en cada máquina.

Existen bastantes propuestas para atacar este problema de distribución de procesos (ver [SKS92]) y una versión futura podría utilizar alguna de ellas para intentar equilibrar mejor el trabajo entre las máquinas participantes.

### 6.2.2. Experimentación

Para esta tesis se realizó un número bastante grande de experimentos que permitieron observar el efecto de los parámetros de migración y de la topología sobre el número de evaluaciones necesarias para alcanzar el óptimo. A pesar de que los resultados obtenidos permitieron alcanzar conclusiones interesantes, aún hay muchos aspectos de los AGPs que deben ser investigados. Esta sección discute algunos de ellos.

#### 6.2.2.1. Otros parámetros y operadores

Los experimentos de esta tesis se hicieron con un conjunto bastante estandarizado de parámetros para el algoritmo genético. Los únicos parámetros que se variaron fueron los relativos a la migración. Se pueden realizar experimentos utilizando un subconjunto de los mejores parámetros de migración que se encontraron cambiando otros parámetros del AG como el tamaño de la población o la brecha generacional.

La experimentación también se puede ampliar utilizando otros operadores genéticos.

#### 6.2.2.2. Otras topologías

Es importante hacer experimentos con otras topologías para tratar de obtener datos que permitan mejorar la comprensión del efecto de la topología empleada en el desempeño del AGP. Realmente es necesario saber qué característica de la topología es la que impacta más al AGP: el número de nodos totales, el número de vecinos o el diámetro de la gráfica que describe a la topología.

A corto plazo se proponen experimentos con topologías de estrella, con árboles y con variaciones de las topologías usadas en esta tesis: hipercubos de diferentes grados, anillos bidireccionales (chicos y grandes), y matrices de diferentes tamaños.

#### 6.2.2.3. Otras funciones

En el capítulo anterior se mencionó la hipótesis que formuló Mühlenbein acerca de que quizás no es conveniente usar AGPs en funciones discontinuas. `f3` es una función discontinua y los resultados obtenidos en los experimentos de esta tesis parecen corroborar esta hipótesis.

Es necesario realizar experimentos con otras funciones discontinuas para intentar mostrar la validez de la hipótesis. Si los experimentos comprobaran la hipótesis se alcanzaría un resultado muy importante: se tendría alguna idea de la clase de problemas en la que es preferible no usar AGPs.

También es importante probar DGENESIS con funciones muy complejas que requieran un tiempo de evaluación considerable. Con funciones de este tipo se pueden realizar experimentos que permitan medir más confiablemente el tiempo de ejecución del AGP y se podría comparar con el tiempo necesario por un AG serial.

La finalidad de los AGs es resolver problemas, y aunque las funciones con las que se prueban los algoritmos son representativas de una variedad grande de problemas, sería muy conveniente hacer una aplicación con un problema real.

### 6.2.3. Investigación en otros campos

Existen muchos otros métodos de optimización de funciones y es importante comparar a los AGPs con otros algoritmos de computación evolutiva y de optimización tradicional. Así se tendría una comparación más amplia que ayudaría a situar correctamente a los AGPs dentro de los algoritmos de optimización existentes.

El funcionamiento de los otros métodos de computación evolutiva es diferente al de los AGs en varios sentidos. Por ejemplo, en programación evolutiva los individuos son máquinas de estados finitos y el único operador es la mutación. Este funcionamiento diferente hace que las poblaciones evolucionen de distinta manera (e.g. más rápido o más lento) y el efecto de la migración puede afectar esta evolución de una manera diferente a como lo hace en un AG.

También sería interesante ampliar la investigación a otras áreas de algoritmos genéticos como sistemas clasificadores o programación genética para comprobar si se obtienen mejoras similares a las que se obtienen en optimización.

Es importante recordar que los AGs son el resultado del trabajo de Holland que pretende explicar el fenómeno de adaptación en sistemas complejos. Se podría utilizar un AGP como un modelo de un sistema complejo natural y estudiar su comportamiento con herramientas adecuadas; quizás se podrían observar fenómenos como competencia y/o cooperación entre especies.

### 6.2.4. Teoría

El trabajo más importante que es necesario hacer en el futuro es la descripción formal de los fenómenos que ocurren en un AGP. Existen algunos trabajos importantes en esta área, pero su alcance es muy limitado. Goldberg [Gol89c] extendió su teoría del tamaño óptimo de poblaciones de AGs seriales a AGPs, su conclusión más importante se usó en este trabajo: usar el mayor tamaño posible de población total para muestrear el espacio más grande posible. Existen también trabajos [PeLe89] que se concentran en el efecto de la migración sobre el muestreo de esquemas (ver el Teorema de Esquemas en el capítulo 2), pero sólo consideran esquemas de migración extremadamente simples.

Para explotar adecuadamente el poder de estos algoritmos, es necesario tener una base formal que permita predecir y controlar su comportamiento. También es muy importante caracterizar la clase de problemas que se pueden enfrentar con estos algoritmos. Si no se conoce la frontera es imposible ampliarla.

# Referencias


[Bäc92] Bäck, Thomas, *A User´s Guide to GENEsYs 1.0*, University of Dortmund, Department of Computer Science, Systems Analysis Research Group, July 1st, 1992.

[Bäc93] Bäck, Thomas, *A Guide to Literature and Software on Evolution Strategies*, Spotlighted Technologies Tutorial Notes, ICGA-93, 1993.

[Bal93] Baluja, Shumeet, Structure and Performance of Fine-Grain Parallelism in Genetic Search, *In Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.

[BaRo93] Ballinger, Geoffrey H, Ross, Peter, *PGA 2.4 Documentation*, University of Edinburgh, 1993.

[BeBo91] Belew, R.K., Booker, L.B. (eds), *Proceedings of the Fourth International Conference on Genetic Algortihms*, Morgan Kaufmann, 1991.

[BiBr92] Bianchini, Ricardo, Brown, Christopher, *Parallel Genetic Algorithms on Distributed-Memory Arquitectures*, The University of Rochester, Computer Science Department, Technical Report 436, August 1992.

[Can94] Cantú Paz, Erick, *Manual de Usuario de Distributed Genesis, Versión 1.0*, ITAM, 1994.

[Cav70] Cavicchio, D.J., *Adaptive search using simulated evolution*, PhD Thesis, University of Michigan at Ann Arbor, 1970.

[CMR91] Cohoon, J.P., Martin, W.N., Richards, D.S., A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes, *In Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991.

[Daw86] Dawkins, Richard, *The Blind Watchmaker*, W.W. Norton, 1986.

[DeJ75] DeJong, Kenneth A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan, Ann Arbor, 1975.

[DeSp93] De Jong, Kenneth, Spears, William, On The State of Evolutionary Computation, *In Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.

[Dun90] Duncan, Ralph, *A Survey of Parallel Computer Architectures*, IEEE Computer, February, 1990.

[deTi93] de la Maza, Michael, Tidor, Bruce, Analysis of Selection Procedures with Particular Attention to Proportional and Boltzmann Selection, *In Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.

[EAV91] Eiben, A.E., Aarts, E.H., Van Hee, K.M., *Global Convergence of Genetic Algorithms:An Infinite Markov Chain Analysis*, Parallel Problem Solving from Nature, Springer Verlag, 1991.

[ESG91] Earickson, Jeff A., Smith, Robert E., Goldberg David E., *SGA-Cube: A Simple Genetic Algortihm for nCUBE 2 Hypercube Parallel Computers*, TCGA Report No. 91005, The Clearing House for Genetic Algorithms, The University of Alabama, Department of Engineering Mechanics, 1991.

[Est92] Estivill-Castro, Vladimir, *Algoritmos Genéticos. Seminario Tutelar (Notas)*, IX Reunión Nacional de Inteligencia Artificial, 1992.

[FOW66] Fogel, L.J., Owens, A.J., Walsh, M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley and Sons, 1966.

[For93] Forrest, Stephanie (ed), *Proceedings of the Fifth International Conference on Genetic Algortihms*, Morgan Kaufmann, 1993.

[Gol89] Goldberg, David E., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison Wesley, 1989.

[Gol89b] Goldberg, David E., Zen and the Art of Genetic Algorithms, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[Gol89c] Goldberg, David E., Optimal Population Sizes for Serial and Parallel Genetic Algorithms, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[GoWh93] Gordon, Scott, Whitley, Darrell, Serial and Parallel Genetic Algorithms as Function Optimizers, *In Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.

[Gre81] Grefenstette, John J., *Parallel adaptive algorithms for function optimization*, Technical Report CS-81-19, Vanderbilt University, Computer Science Department, 1981.

[Gre90] Grefenstette, John J., *A Users Guide to GENESIS Version 5.0*, Navy Center for Applied Research in Artificial Intelligence, Wasington D.C., 1990.

[Hei94] Heitkoetter, Joerg, *The Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ)*, USENET:comp.ai.genetic, 1994.

[Hol92a] Holland, John H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, 2nd edition, 1992.

[Hol92b] Holland, John H., *Genetic Algorithms*, Scientific American, July 1992.

[Hug89] Hughes, Mark, *Genetic Algorithm Workbench Documentation*, November 1989.

[Koz92] Koza, John R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[LeFa] Leffler, Samuel J., Fabry, Robert S., et al., *An Advanced 4.3 BSD Interprocess Communication Tutorial*, Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

[MTS93] Munetomo, Masaharu; Takai, Yoshiaki; Sato, Yoshiharu; *An Efficient Migration Scheme for Subpopulation-Based Asynchronous Parallel Genetic Algorithms*, Information and Graphics Sciences, Faculty of Engineering, Hokkaido University (Poster at ICGA-93), 1993.

[Müh89] Mühlenbein, H., Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[MSB91] Mühlenbein, H., Schomisch, M., Born, J., The Parallel Genetic Algorithm as Function Optimizer, *In Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991.

[PeLe89] Pettey, Chrisila C., Leuze, Michael R., A Theoretical Investigation of a Parallel Genetic Algorithm, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[Rio92] Riolo, Rick, *Survival of the Fittest Bits,* Scientific American, July 1992.

[Ros67] Rosenberg, R.S., *Simulation of genetic populations with biochemical properties*, PhD Thesis, University of Michigan, 1967.

[Rub75] Rubinstein, Moshe, *Patterns of Problem Solving*, Prentice Hall, 1975.

[ScBe92] Schraudolph, Nicol N., Belew, Richard K., *Dynamic Parameter Encoding for Genetic Algorithms*, Machine Learning, Sept. 1992.

[ScGr92] Schraudolph, Nicol N., Grefenstette, John J., *A User´s Guide to GAucsd 1.4*, Technical Report CS92-249, University of California at San Diego, July 7, 1992.

[Sch93] Schraudolph, Nicol N., *Genetic Algorithm Software Survey*, University of California at San Diego, February 1993.

[SKS92] Shivaratri, Niranjan G., Krueger, Phillip, Singhal, Mukesh, *Load Distributing for Locally Distributed Systems*, IEEE Computer, December 1992.

[Smi90] Smith, Joshua R., *BUGS: A Non-Technical Report*, Department of Computer Science, Williams College, August 14, 1990.

[Ste90] Stevens, W. Richard, *UNIX Network Programing*, Prentice Hall, 1990.

[Sys89] Syswerda, G., Uniform Crossover in genetic algorithms, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[Tan89] Tanese, Reiko, Distributed Genetic Algorithms, *In Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, 1989.

[van92] van Lent, Michael C., *Parallel Genesis User´s Guide*, Naval Research Laboratory, August 1992.

[Wei70] Weinberg R., *Computer simulation of a living cell*, PhD Thesis, University of Michigan, 1970.

[Whi93] Whitley, Darrell, Cellular Genetic Algorithms, *In Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1993.

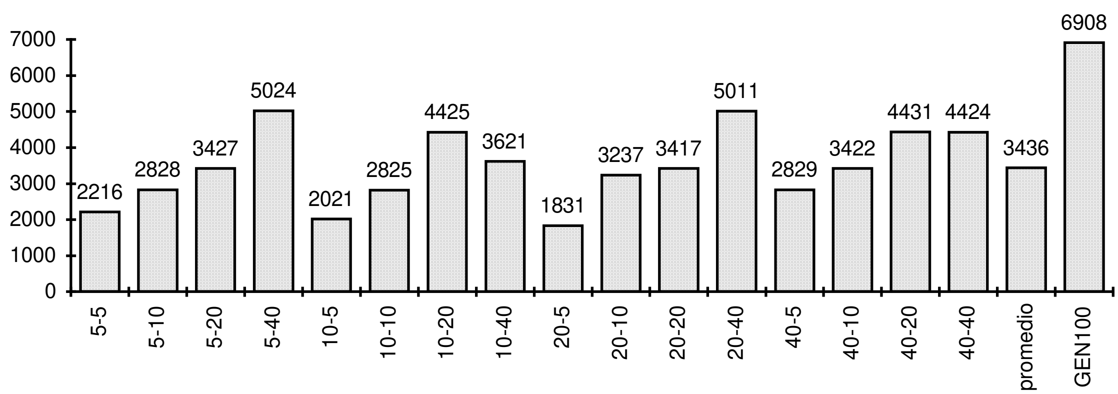
[ZeKi93] Zeigler, Bernard P., Kim, Jinwoo, *Asynchronous Genetic Algorithms on Parallel Computers*, AI and Simulation Group, Department of Electrical and Computer Engineering, University of Arizona (Resumen en Proceedings of the Fifth International Conference on Genetic Algorithms), 1993.

# Apéndice I

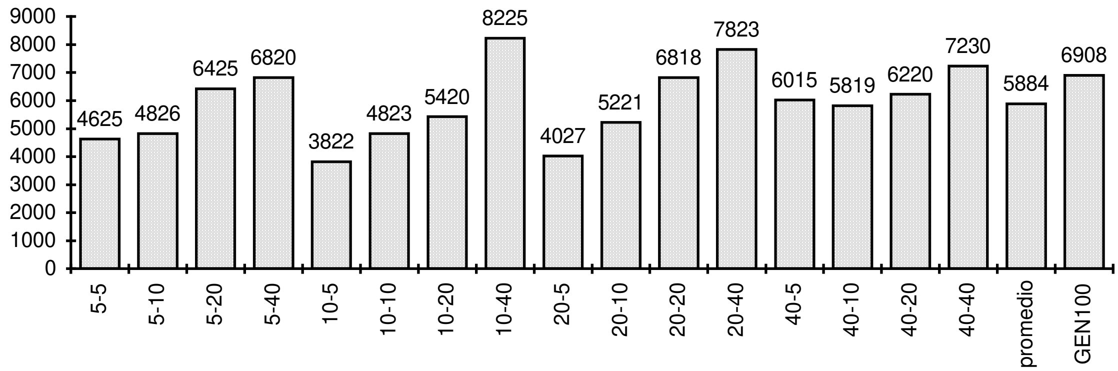
Los resultados completos de los experimentos se presentan en este apéndice. Cada gráfica representa el número de **evaluaciones por nodo** necesarias para alcanzar el óptimo de la función con 4 decimales de precisión. Hay una gráfica para cada topología y en cada gráfica se presentan los resultados de 16 combinaciones de parámetros de migración. El primer número en las etiquetas del eje horizontal es la tasa de migración, el segundo número es el intervalo de migración. También en cada gráfica se presenta el promedio de todas las combinaciones de parámetros y las evaluaciones que requiere un AG serial con una población de 100 individuos para alcanzar el óptimo.

## Función f1

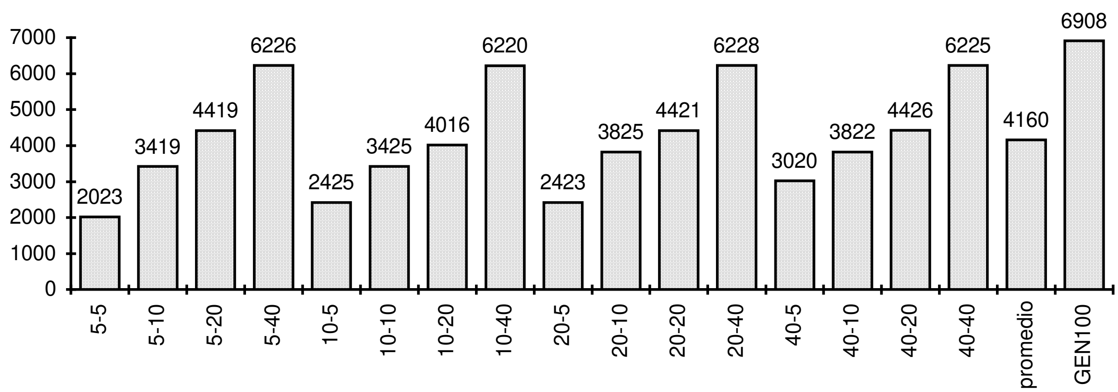
### *Topología completa*



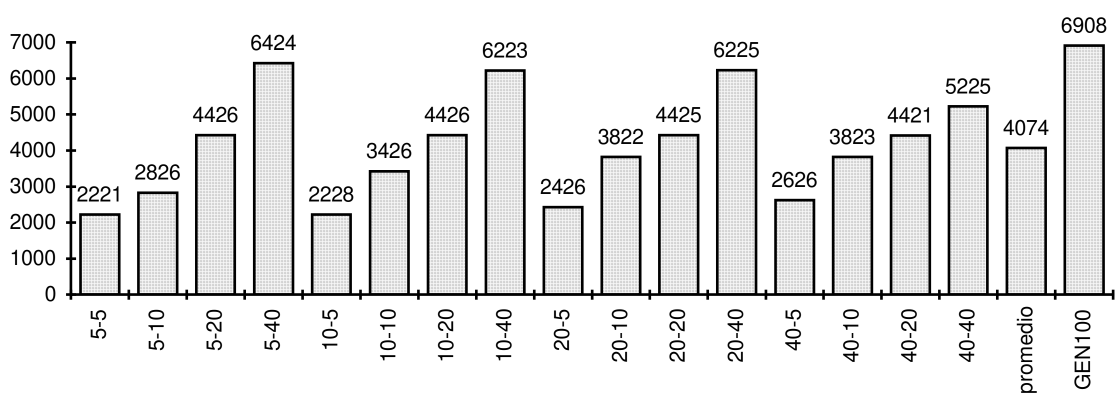
*Anillo*




*Hipercubo*



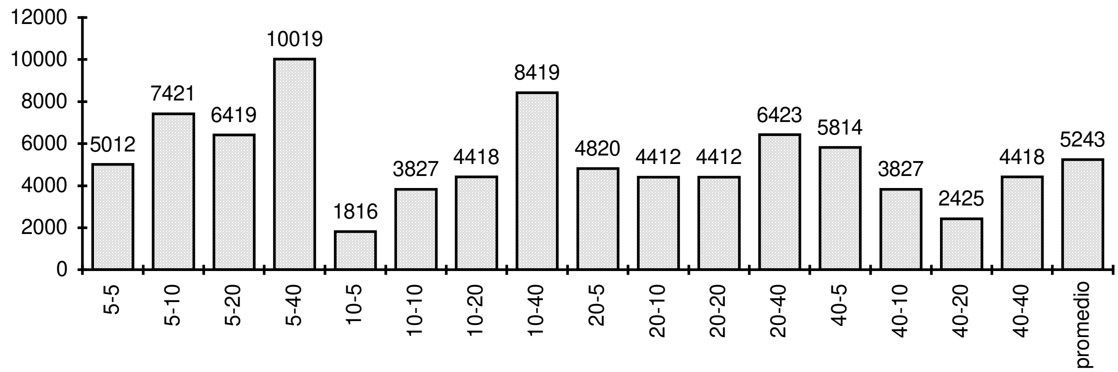

*Malla*



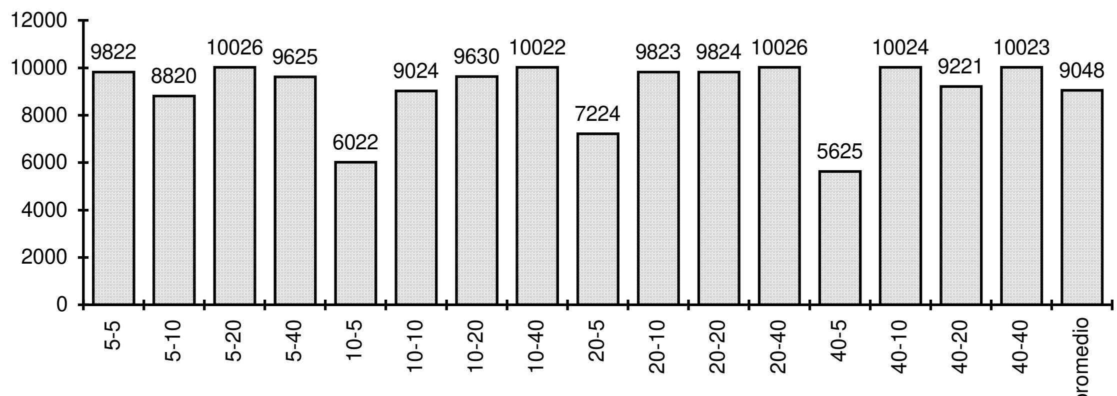
## *Función f2*

En esta función se omite de la gráfica el número de evaluaciones requeridas por el AG serial con 100 individuos (68814) porque la escala de la gráfica no permitiría ver claramente las diferencias entre los demás valores.

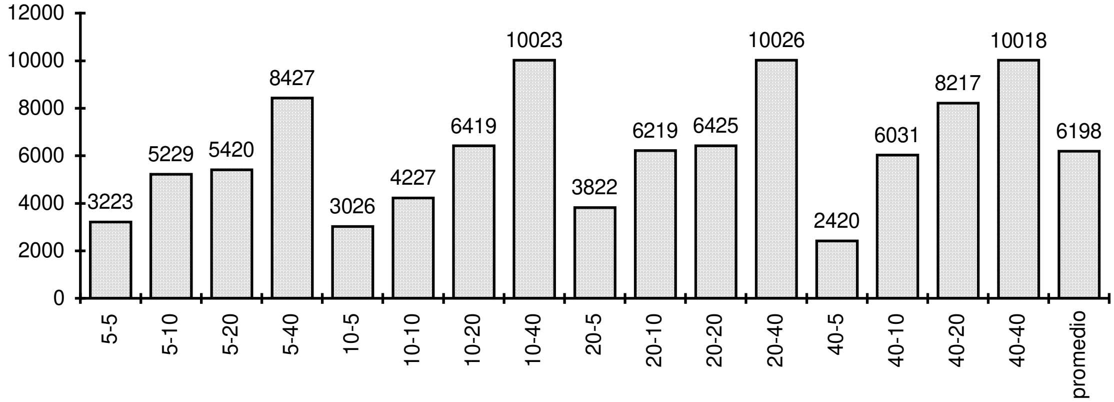
### *Topología completa*



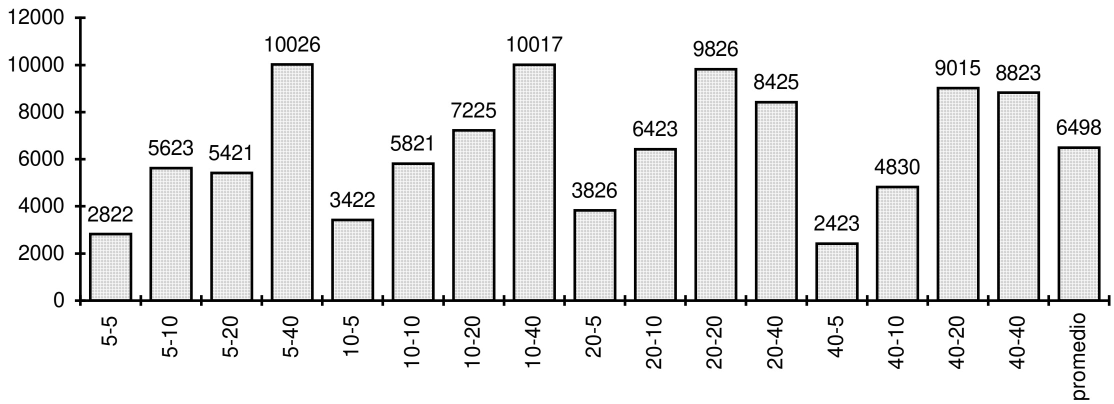

### *Anillo*



## Hipercubo

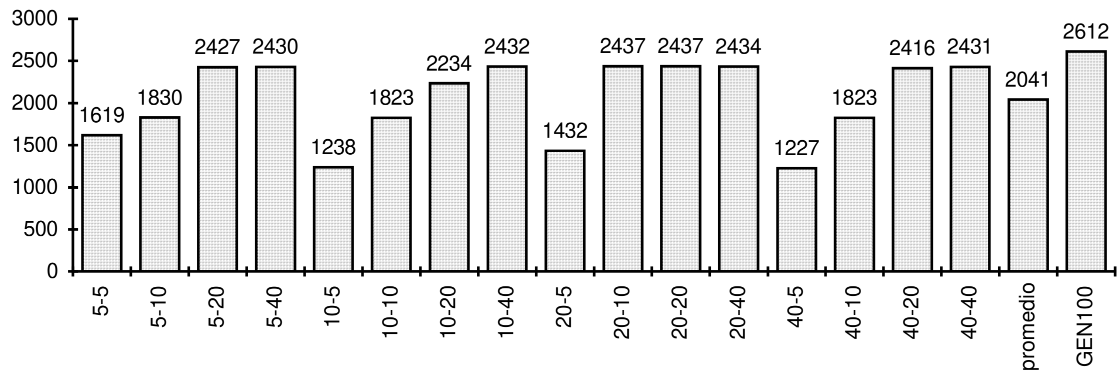



## Malla

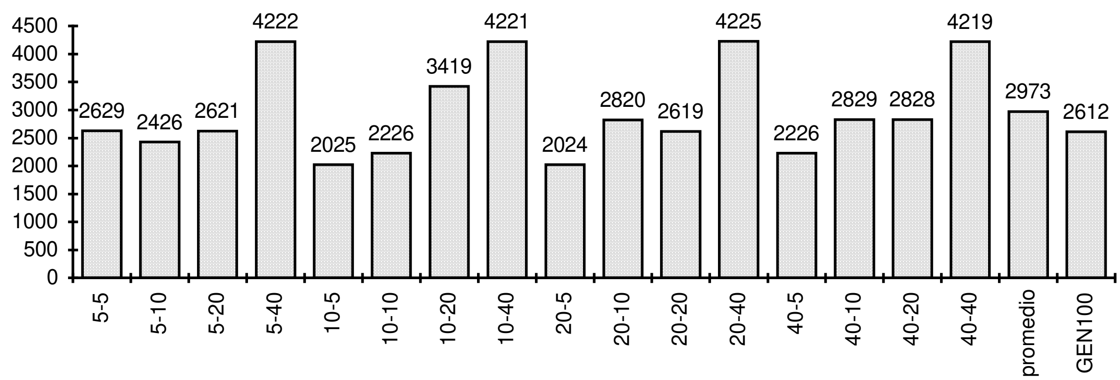


## Función f3

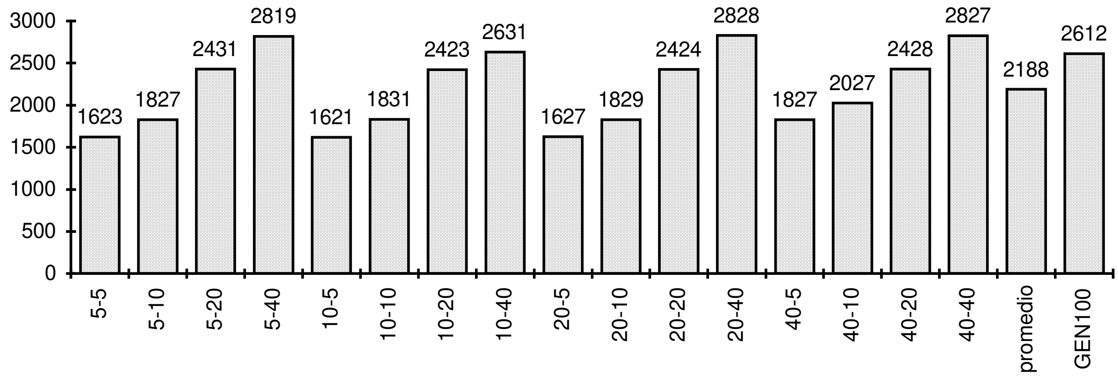
### Topología completa



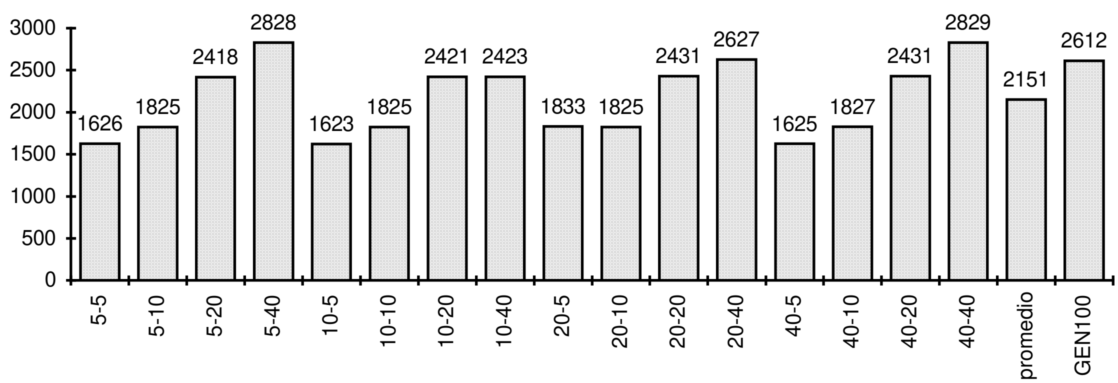

### Anillo



## *Hipercubo*

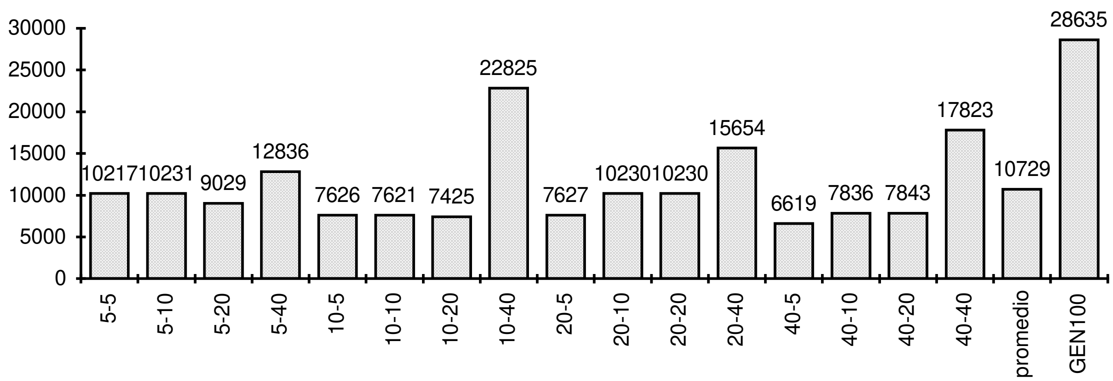



## *Malla*

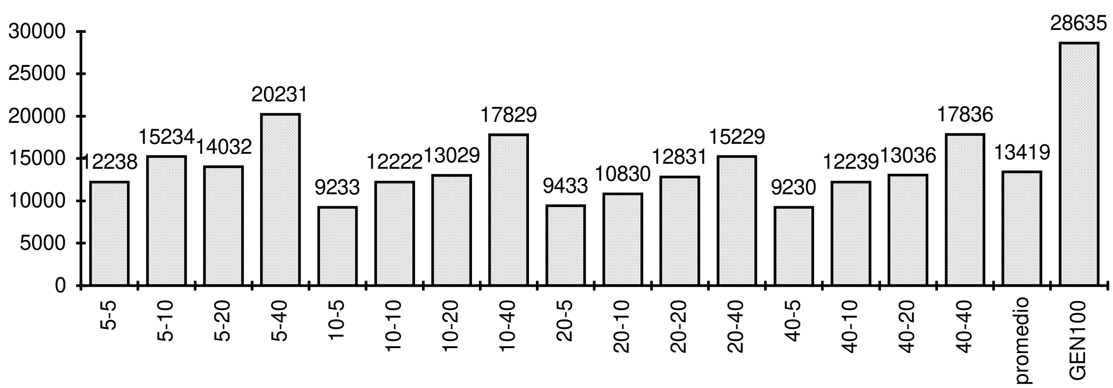


## ***Función f4***

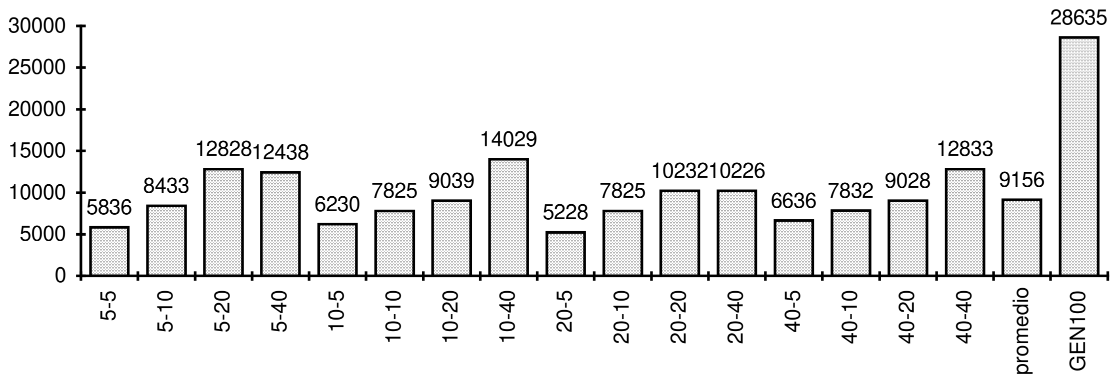
*Topología Completa*



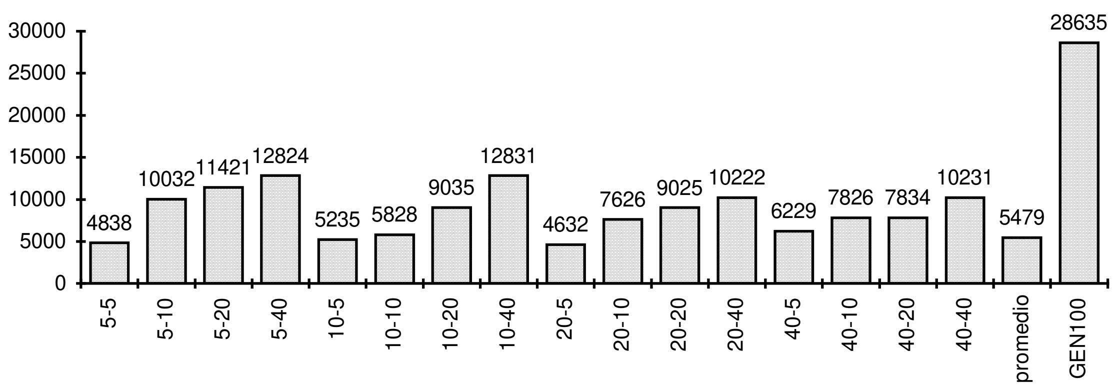

*Anillo*



*Hipercubo*

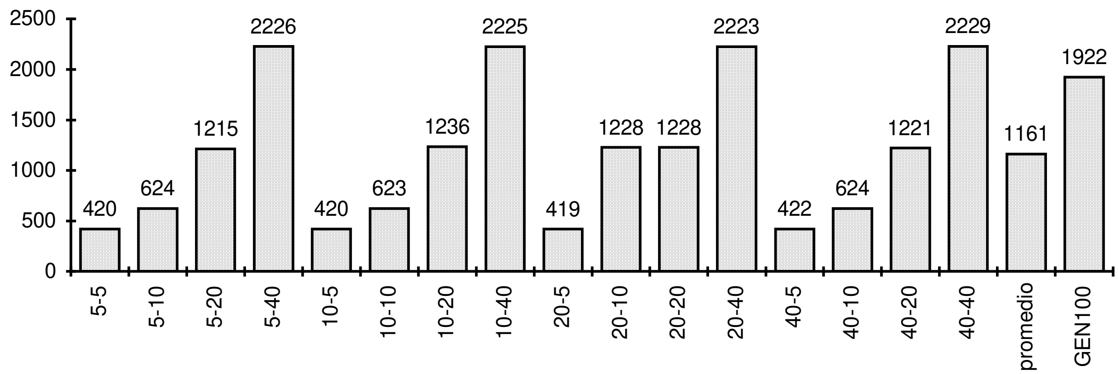



*Malla*

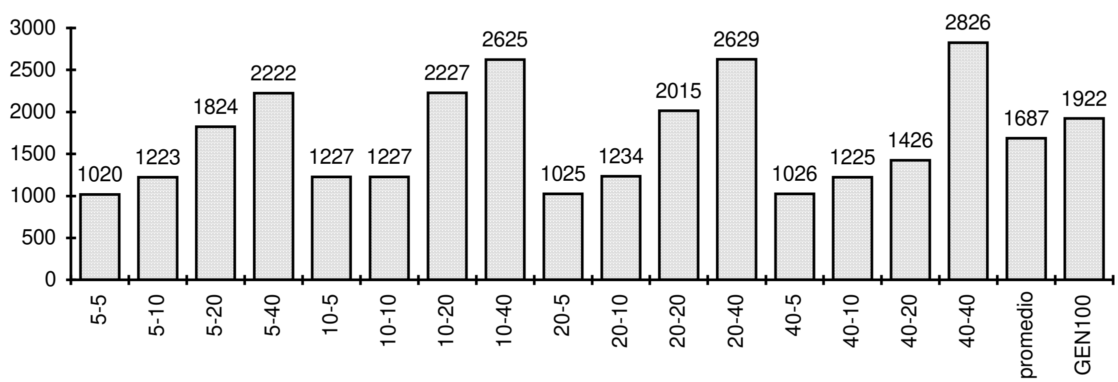


# *Función f5*

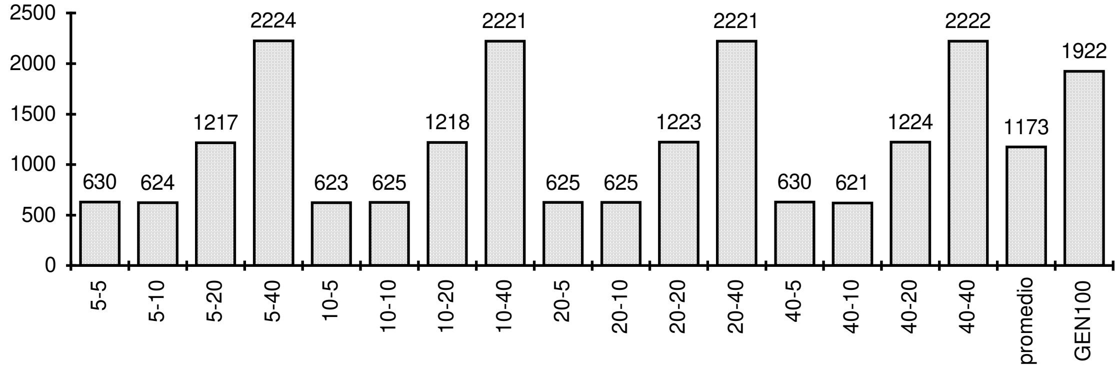
## *Topología completa*



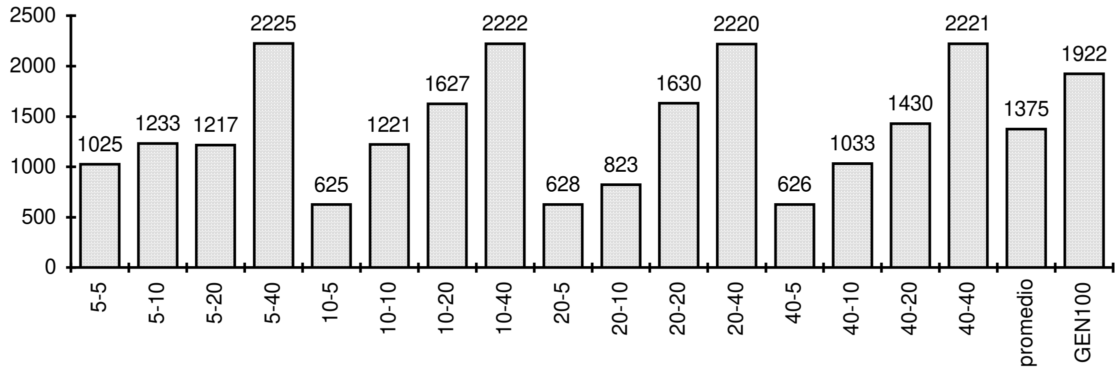

## *Anillo*



*Hipercubo*



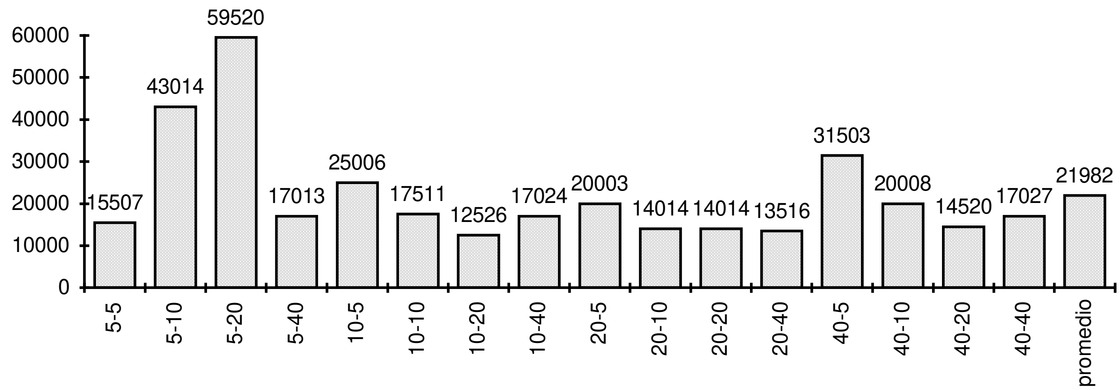

*Malla*



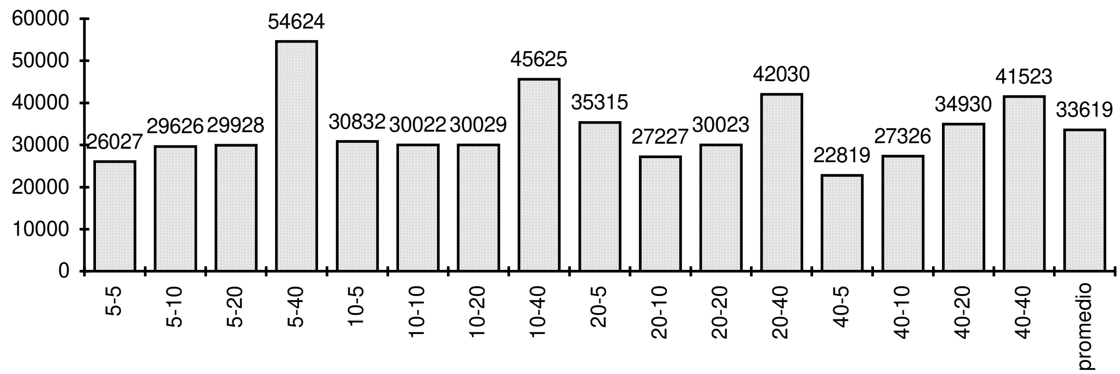

## *Función f6*

En esta función también se omite de la gráfica el número de evaluaciones requeridas por el AG serial con 100 individuos (265019) porque la escala de la gráfica no permitiría ver claramente las diferencias entre los demás valores. De cualquier forma, el AG serial no encuentra el óptimo global, sólo uno de los subóptimos.

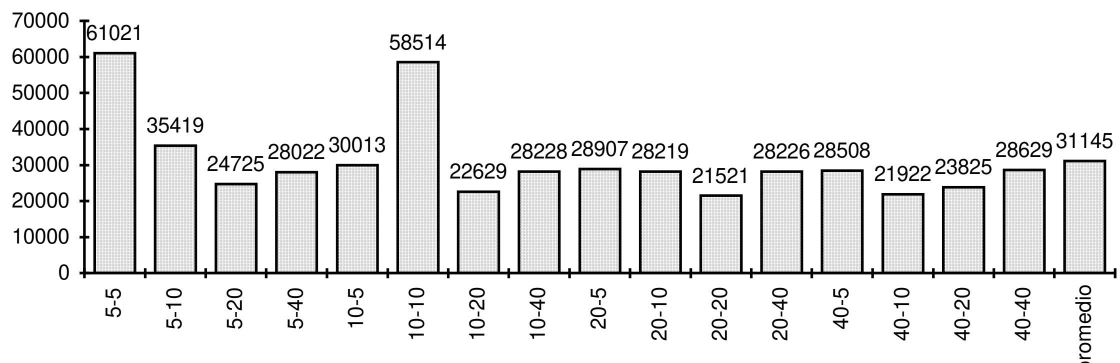
## Topología completa



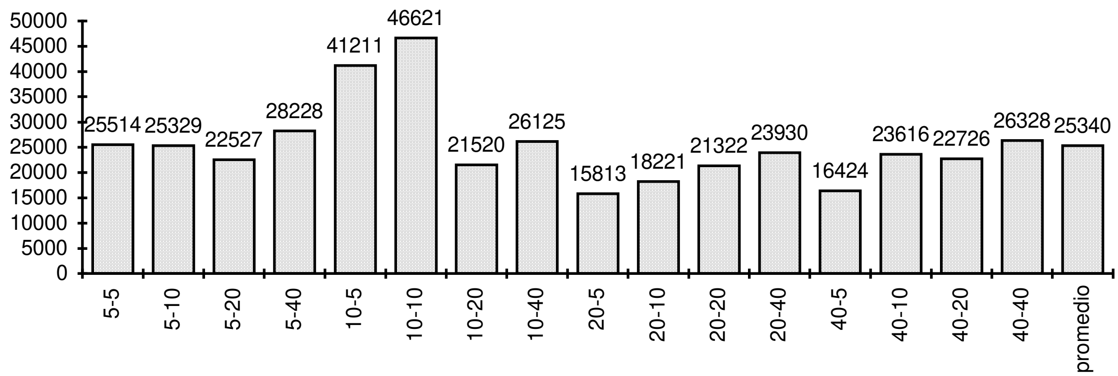

## Anillo




## Hipercubo

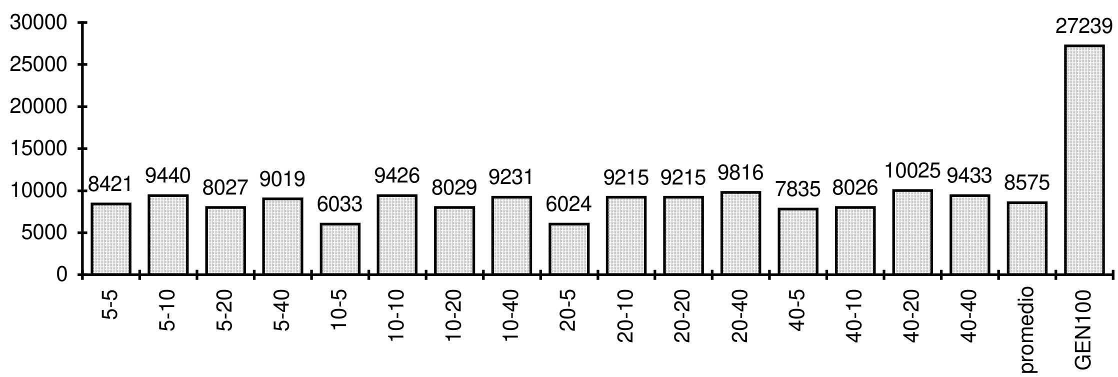


*Malla*

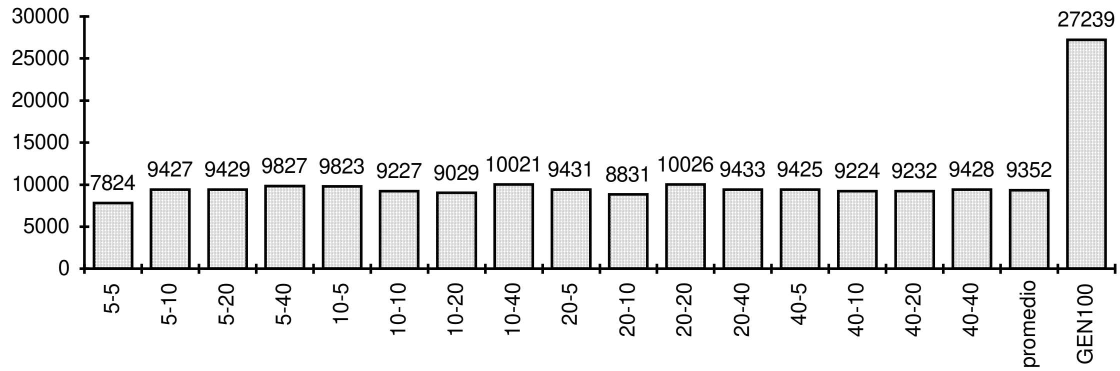



## ***Función f7***

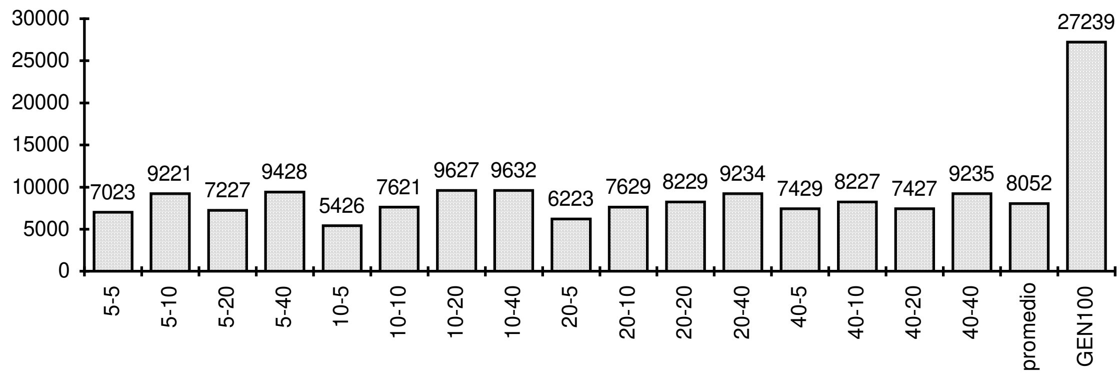
*Topología completa*



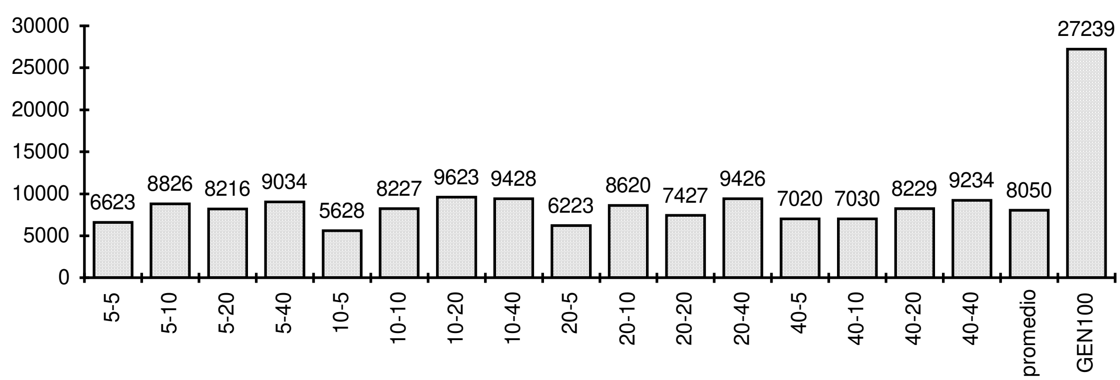
*Anillo*




*Hipercubo*




*Malla*



## Función f8

Esta es la función más difícil de optimizar y solo algunos experimentos alcanzaron el óptimo global después de 100,000 evaluaciones por nodo. A continuación se presentan los experimentos exitosos y el número de evaluaciones por nodo que necesitó cada uno. Los experimentos con AGs seriales no alcanzaron el óptimo después de 750,000 evaluaciones, sus resultados se presentaron en el capítulo 5.

### *Topología completa*

| tasa - intervalo | evaluaciones |
|---|---|
| 5 - 10 | 96215 |
| 20 - 40 | 61222 |

### *Anillo*

| tasa - intervalo | evaluaciones |
|---|---|
| 10 - 20 | 58619 |
| 20 - 10 | 31825 |
| 20 - 40 | 75229 |
| 40 - 10 | 34622 |
| 40 - 20 | 70627 |
| 40 - 40 | 88224 |

### *Hipercubo*

| tasa - intervalo | evaluaciones |
|---|---|
| 5 - 20 | 92021 |
| 40 - 5 | 15525 |

### *Malla*

| tasa - intervalo | evaluaciones |
|---|---|
| 20 - 5 | 16008 |
| 20 - 10 | 13519 |
| 20 - 20 | 23030 |
| 40 - 5 | 19024 |
| 40 - 40 | 75532 |

# Índice Temático

## *D*



## *E*



## *F*



## *G*



## *H*



## *I*



## *L*



## *M*

## *U*



## *V*