UNIVERSIDAD DE MÁLAGA

E. T. S. DE INGENIERÍA INFORMÁTICA

Departamento de Lenguajes y Ciencias de la Computación

Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos

TESIS DOCTORAL

Enrique Alba Torres

Málaga, Febrero de 1999

Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos

TESIS DOCTORAL

Presentada por
D. Enrique Alba Torres
para optar al grado de
Doctor en Informática

Dirigida por el
Dr. José María Troya Linero
Catedrático de Universidad del
Área de Lenguajes y Sistemas Informáticos

Málaga, Febrero de 1999

D. *José María Troya Linero*, Catedrático de Universidad del Área de Lenguajes y Sistemas Informáticos de la E. T. S. de Ingeniería Informática de la Universidad de Málaga

Certifica que

D. *Enrique Alba Torres*, Licenciado en Informática, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo mi dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulada

Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos

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

Málaga, 1 de Febrero de 1999

Fdo.: *José María Troya Linero*
Catedrático de Universidad del
Área de Lenguajes y Sistemas Informáticos

Agradecimientos

Este trabajo representa el esfuerzo conjunto de muchas personas que me han ayudado a lo largo de muchos años, y no sólo el mío propio. En primer lugar, me gustaría agradecer a mi director José Mª Troya su constante apoyo y seguimiento en este campo de investigación concreto.

De manera sobresaliente y definitiva quisiera agradecer a mis padres y mi hermana sus apoyos continuados que, a lo largo de mis años de formación, me han permitido realizarme como profesional y como persona. Asimismo, quiero dar las gracias a mis compañeros de despacho, Carlos y Antonio, y a los compañeros del departamento por su ayuda, discusiones, y aliento constante.

También quiero mencionar la ayuda de mis amigos del coro rociero, la chirigota, y de las distintas actividades culturales que me han ayudado a despejar las ideas y a conocer otros valores.

Por último, pero no menos importante, quiero agradecer a mi mujer, Ana, y a mi hijo, Enrique, la ayuda, apoyo y esfuerzos que han hecho sin una queja. Espero poderos compensar en el futuro por tantos años de amor desinteresado, porque sin vosotros ni este trabajo ni yo seríamos.

ÍNDICE

Introducción

Planteamiento

En la actualidad uno de los puntos débiles en la aplicación de algoritmos evolutivos (AEs) [BFM97] es su moderado/alto consumo de recursos computacionales (memoria y/o tiempo del procesador). La inclusión de técnicas paralelas en la definición de algoritmos ha sido muy importante en los recientes diseños de este tipo de mecanismos de búsqueda y optimización [Sten93] [Adam94] [LPG94] [Cant97a]. De esta forma, la especial adecuación de estos heurísticos para trabajar sobre problemas de elevada complejidad puede verse mejorada si el diseño y la implementación son paralelos.

Entre los algoritmos evolutivos, tanto secuenciales como paralelos, podemos distinguir varios modelos de búsqueda diferenciados en la actualidad pero con raíces comunes [BHS97]:

- Algoritmo Genético (AG), la variante más popular.
- Estrategia Evolutiva (EE), típica en ingeniería de sistemas.
- Programación Evolutiva (PE), típica en aplicaciones de inteligencia artificial.
- Programación Genética (PG) con una gran diversidad de aplicaciones a problemas reales.
- Sistema Clasificador (SC), normalmente empleado en el aprendizaje máquina.

El contenido de este trabajo se centra en el campo de los algoritmos genéticos, tanto secuenciales como paralelos (AGPs), por varias razones. En primer lugar, por su amplia aceptación en numerosas áreas de investigación y aplicación; en segundo lugar, debido a que muchas de las aportaciones que se realicen serán extensibles a algunos de los restantes tipos de algoritmos evolutivos y, finalmente, porque la demanda de eficiencia tanto en tiempo real como numérica es, en este campo, muy acusada debido a que son muy utilizados.

Tradicionalmente se ha distinguido entre algoritmos genéticos paralelos de *grano grueso* y de *grano fino*, según que la relación computación/comunicación fuera alta o baja, respectivamente. Sin embargo, las aportaciones de los investigadores en este terreno son confusas debido a que pocas veces se los compara directamente, y en muy contadas ocasiones se los combina. De hecho, las comparaciones existentes entre modelos secuenciales y paralelos están a menudo oscurecidas por la inclusión de técnicas híbridas de búsqueda en los algoritmos estudiados. Asimismo, la distinción respecto al grano de los modelos paralelos los ha presentado con frecuencia como alternativos, cuando en realidad son dos expresiones paralelas que podrían combinarse para obtener sus ventajas respectivas evitando en lo posible sus inconvenientes.

Aunque la existencia de algoritmos genéticos paralelos híbridos no es nueva [Gorg89] [WS90] [MSB91] [VSB92] [GW93] [Grua94] [Loza96] [Pedr96], sí lo es el estudio unificado de modelos canónicos en los que se evite la presencia de técnicas adicionales. Estas técnicas, aunque mejoran el algoritmo para un problema concreto, no ayudan al conocimiento de sus características respectivas. De hecho, cualquier técnica que mejore el comportamiento básico de un AG secuencial puede aplicarse a los que presentamos y por tanto dicha mejora podría también conseguirse en ellos.

Por tanto, un estudio canónico del comportamiento secuencial y paralelo presenta múltiples ventajas, tanto en el terreno teórico como en el práctico. La utilización de modelos paralelos está notablemente avalada por un conjunto muy numeroso de aplicaciones a problemas reales y estudios sobre su comportamiento. Sin embargo, es manifiesta la falta de uniformidad de estos modelos paralelos que, de existir, ayudaría a crear una metodología y extracción de características comunes que permitieran estimar su comportamiento en otros problemas.

En consecuencia, es importante la presentación de los distintos modelos paralelos como subclases de una formalización común de algoritmo evolutivo paralelo que permita una comparación de sus componentes, una identificación de similitudes/diferencias, y una transferencia de conceptos y a veces de resultados teóricos entre ellos. Por otro lado, algunos resultados no están suficientemente contrastados, sobre todo en lo concerniente a parámetros clave del diseño de algoritmos genéticos paralelos.

En particular, el diseño de versiones distribuidas es muy interesante por la amplia disposición de redes de estaciones de trabajo en la actualidad. Existen algunos resultados en este sentido, pero no evalúan las prestaciones de diferentes modos de evolución en los sub-algoritmos componentes. Este es otro de los huecos que pretendemos cubrir. En particular, podemos distinguir como modos de evolución básicos los siguientes:

- Evolución en estado estacionario (*steady state*) [Sysw91].
- Evolución generacional (tradicional) [Gold89a].
- Evolución ajustable (*generation gap*), que unifica las dos anteriores [Gold89a].
- Evolución por difusión (grano fino), en la que se utiliza un esquema de vecindad [SM91].

En realidad demostraremos cómo todos estos modos de selección, reproducción y reemplazo en poblaciones con o sin disposición espacial de las estructuras-solución son expresiones de un sistema subyacente común. Esta visión permite explicar, clasificar y estudiar algoritmos existentes así como derivar nuevos algoritmos de comportamiento más predecible.

Debido a estas razones, y puesto que cualquiera de estos algoritmos es un sistema *software* de mayor o menor complejidad, es interesante dar un diseño e implementación usando una metodología orientada a objetos [Rumb91]. Actualmente, los sistemas existentes aparentan usar una colección de recursos separados con los que resulta muy difícil realizar prototipos rápidos o hacer extensiones del algoritmo paralelo. Además, el conjunto de parámetros que en algunos casos se utiliza para problemas complejos parece indicar que el uso de un sistema orientado a objetos es la decisión más segura, extensible y beneficiosa a medio y largo plazo.

Para terminar este planteamiento inicial resta añadir algunas consideraciones sobre la evaluación de algoritmos paralelos. En este caso, y debido a la naturaleza estocástica de los sistemas que manejamos, las condiciones tradicionales de parada no son válidas. Los algoritmos deben ejecutarse hasta alcanzar soluciones de adecuación similares, pues es este el único criterio justo y realista para su comparación [HBBK97] [CG97]. Por otro lado, algunos estudios que versan sobre aspectos concretos de estos algoritmos se realizan sobre problemas muy simples. Esto permite comprender su comportamiento, pero al mismo tiempo desautoriza en cierta medida las conclusiones alcanzadas. En consecuencia, resulta imperativo un estudio sobre problemas de complejidad no trivial y tan alta como sea posible para poner de manifiesto los puntos fuertes y débiles de cada modelo y obtener conclusiones correctas con una seguridad razonable.

Objetivos

El objetivo global del presente trabajo es proporcionar un estudio significativo de modelos secuenciales y paralelos de algoritmos genéticos que sea útil por cubrir las variantes más usadas y que también sirva de base para proponer modelos concretos que presenten una mayor eficiencia. Este objetivo genérico se descompone en otros sub-objetivos particulares:

1. Descripción unificada de las clases de algoritmos genéticos secuenciales y paralelos.
2. Construcción y evaluación de un modelo paralelo distribuido de evolución básica secuencial o de población con estructura bidimensional.
3. Estudio de las prestaciones de los modelos de población única, estructurada o distribuida (evaluados en mono/multi-procesador). Análisis sobre problemas de dificultad elevada, determinación de su utilidad práctica, y estudio de las características más sobresalientes.
4. Determinación de decisiones *software* conducentes a una implementación eficiente.

Aportaciones Originales

En esta sección listamos las principales aportaciones que incluimos a lo largo de esta memoria y las relacionamos con los trabajos que más puedan asemejarse a cada una de ellas. En general, las aportaciones pueden clasificarse como *extensiones* de trabajos existentes o bien como *nuevos resultados*. En resumen, las aportaciones son:

1. *Caracterización formal unificada de los algoritmos evolutivos secuenciales*. Se trata de una extensión del trabajo de [Bäck96] junto con algunas definiciones nuevas.
2. *Estudio y clasificación unificados de los modelos paralelos de algoritmos genéticos*. Para ello partimos de la clasificación proporcionada en [Sten93] e introducimos modelos actuales incluyendo un mayor número de detalles sobre ellos. Como aportaciones originales se incluyen los resúmenes ordenados por año y los detalles de los modelos paralelos.
3. *Incorporación del paradigma de orientación a objetos en el diseño de software evolutivo*. Se trata de una aportación nueva, aunque existen algunos intentos con igual objetivo pero usando medios distintos.
4. *Formalización unificada de modelos paralelos de algoritmos evolutivos y genéticos*. Se pretende abrir con ello la línea de trasvase de conocimientos entre modelos paralelos. La derivación unificada permitirá entender los modelos existentes y derivar nuevos modelos paralelos, muchos de ellos evaluados en este trabajo. Las bases genéricas pueden encontrarse en el desarrollo de un sistema adaptativo [DRH95].
5. *Estudio unificado de algunos fundamentos de funcionamiento como las operaciones de selección secuencial, distribuida y descentralizada*. Para ello extendemos los estudios en [Sysw91] sobre estado estacionario secuencial al campo paralelo; directamente relacionados están los trabajos sobre AGPs distribuidos de [Tane89], [Gold89b], [Gold95], y mejoramos los desarrollos disponibles en [SD96] y [GW93] sobre selección en algoritmos genéticos celulares.

6. *Propuesta de un modelo distribuido con islas secuenciales o celulares y migración entre ellas*. Estudio sobre un conjunto de casos representativos de dominios de aplicación típicos de computación evolutiva. Algunos resultados existentes en cierta medida similares son [VSB92] y [Pedr96] sobre combinación de jerarquías de niveles y también [Grua94] [RS94] como ejemplos casi únicos de islas de algoritmos celulares (orientados a la aplicación).
7. *Estudio detallado teórico y práctico de la influencia de la forma de la malla sobre un algoritmo genético celular*. Algunos conceptos son extensiones de los trabajos [Davi91b] y [SD96]. Las aportaciones se basan en considerar la malla como rectangular en vez de cuadrada y estudiar el impacto de este hecho usando un vecindario estático.
8. *Estudio de la influencia de la frecuencia y política de migración y reemplazo*. Algunos estudios comparables pueden encontrarse en [DG89], [ME90] y [MC94], si bien nuestros algoritmos demuestran ser más eficientes en esfuerzo de evaluación y en tiempo real.
9. *Estudio en el espacio de los esquemas del trabajo de estos algoritmos*. Realizamos un trabajo nuevo en este sentido atendiendo a los esquemas en sí y a la diversidad [ER96]. Otros parámetros como la epistasis, mérito de los modelos, u otros conceptos, son interesantes, pero escapan por su dimensión a este trabajo.
10. *Resultados sobre la influencia que ciertas decisiones de implementación tienen sobre sistemas paralelos de algoritmos genéticos*. En particular, son novedosas las ideas sobre cómo implementar el vecindario, el plan reproductor, los ficheros de configuración, el protocolo de comunicaciones y los modos síncrono/asíncrono de evolución. Estas características no aparecen conjuntamente en ningún sistema existente, aunque las ventajas de usar un modelo paralelo asíncrono sí que se presentan directamente en [MTS93] y [HBBK97]. Igualmente la capacidad de ejecución heterogénea a nivel de búsqueda es distintiva y nueva. En [AP96] podemos encontrar un sistema heterogéneo atendiendo a parámetros simples como son las probabilidades de mutación y cruce, mientras que en otros modelos como [Loza96] existen planos de búsqueda para exploración y explotación con distintos tipos de cruce.
11. *Las aportaciones presentadas permiten aumentar la potencia de las versiones canónicas de AGPs, luego presentan como ventaja añadida la capacidad de sufrir mejoras ya desarrolladas para modelos más simples*. Nuestras aportaciones siguen siendo susceptibles de mejoras ante un problema concreto debido a hibridación [Cott98], manipulación de la adecuación [GR87], nuevos genotipos [GDK91], etc.

Estructura de la Memoria

La presente memoria está organizada en cinco capítulos, cuatro apéndices y una bibliografía. Los cuatro primeros capítulos describen el trabajo que compone la aportación, mientras que el quinto presenta un resumen de las conclusiones más sobresalientes y el trabajo futuro relacionado con el realizado. A continuación describimos brevemente el contenido.

En el Capítulo 1 se enmarcan los estudios de este trabajo en el contexto de las técnicas de optimización, se presenta el problema de optimización global como la super-clase de problemas que engloba a las aplicaciones a las que frecuentemente se enfrentan los algoritmos estudiados.

Introduciremos en este primer capítulo conceptos importantes que se van a utilizar en el resto de la presentación, tanto para describir técnicas como para estudiar los resultados. Adicionalmente, se discute de manera unificada los algoritmos evolutivos, en particular los algoritmos genéticos secuenciales y paralelos. También se justifica la necesidad de este trabajo, así como la significación de las aportaciones conseguidas. Finalmente, se proporciona la descripción formal de las técnicas que utilizamos y algunas otras relacionadas en términos de codificación, operadores de variación, homogeneidad, migración o difusión y otros conceptos.

El Capítulo 2 describe progresivamente la aportación concreta de funcionamiento paralelo que permite abarcar algoritmos existentes así como concretar modelos paralelos de prestaciones equivalentes o superiores a ellos. Se discute su eficiencia respecto a otras técnicas y se caracterizan los parámetros que más influyen en su comportamiento. Se sugiere la necesidad de los modelos distribuidos en base a trabajos prácticos concretos y, para terminar, se describen las bases del funcionamiento y la presión selectiva que inducen los algoritmos discutidos.

El Capítulo 3 contiene los detalles sobre nuestra propuesta de familia de algoritmos paralelos. También se incluye una descripción de alternativas de implementación usando un lenguaje orientado a objetos como C++ (y también *marcos de programación*) y se ejemplifica la comunicación de un sistema de búsqueda de grano grueso con una descripción del protocolo de comunicación usado en una de las implementaciones discutidas en esta memoria.

El Capítulo 4 contiene los resultados de los estudios realizados en términos de su capacidad de encontrar una solución, eficiencia de la búsqueda (tiempo real y algorítmico), fidelidad a las predicciones teóricas de funcionamiento, comportamiento de cada modelo ante problemas progresivamente más complejos, ganancias en la velocidad de ejecución y relación entre los resultados obtenidos.

En el Capítulo 5 se resumen las conclusiones alcanzadas y se discute su validez relativa. Además, se analizan algunas de las decisiones de implementación y diseño más significativas en el campo de los algoritmos genéticos, y se resaltan los puntos fuertes y débiles de cada técnica estudiada en términos prácticos. Para terminar, se describe el trabajo futuro que tiene sentido e interés a la vista de los resultados, así como algunas ideas preliminares basadas en trabajos en curso.

Los cuatro apéndices finales contienen, respectivamente, (A) la descripción de los problemas usados en este estudio, (B) la sintaxis de los ficheros de configuración y mensajes intercambiados por el sistema paralelo, (C) un glosario de términos y acrónimos, y (D) un resumen orientativo que compara múltiples algoritmos (evolutivos y no evolutivos) entre sí atendiendo a varios criterios.

1 Introducción

Entre el conjunto de técnicas de optimización y búsqueda, en la última década ha sido especialmente importante el desarrollo de los algoritmos evolutivos [BS93] [BFM97]. Los algoritmos evolutivos son un conjunto de metaheurísticos modernos [Reev93a] utilizados con éxito en un número elevado de aplicaciones reales de gran complejidad. Su éxito resolviendo problemas difíciles ha sido el motor de un campo conocido como computación evolutiva (CE) en el que se encuadran estudios sobre aplicabilidad, complejidad algorítmica, fundamentos de funcionamiento y muchos otros aspectos, con el objetivo de facilitar la promoción y compresión de los algoritmos evolutivos a nuevos campos de trabajo.

Aunque los orígenes de la computación evolutiva pueden seguirse hasta finales de los cincuenta (por ejemplo [Box57], [Frie58], [Brem62]) sólo después de tres décadas la comunidad científica ha comenzado a beneficiarse del uso de estas técnicas. Este período de trabajos aislados sobre computadoras de poca potencia y con numerosas lagunas metodológicas comenzó a convertirse en seria investigación de base tras las aportaciones definitivas de Fogel [FOW66], Holland [Holl75], Rechenberg [Rech73] y Schwefel [Schw81]. El cambio durante los años setenta es el responsable de la actual proliferación de estudios teóricos y prácticos sobre este tema (extensas y detalladas referencias de interés son [Gold89a], [Alan96] y [BFM97]).

Los beneficios de las técnicas de computación evolutiva provienen en su mayoría de las ganancias en flexibilidad y adaptación a la tarea objetivo, en combinación con su alta aplicabilidad y características globales de la búsqueda que llevan a cabo. En la actualidad, se entiende la computación evolutiva como un concepto adaptable para resolución de problemas, especialmente apropiado para problemas de optimización complejos. Esta visión [BHS97] es la alternativa a algunas descripciones en desuso que muestran a la computación evolutiva como una colección de algoritmos parecidos listos para usar en cualquier problema.

La mayoría de las implementaciones actuales de algoritmos evolutivos son descendientes de alguno de entre tres tipos básicos (fuertemente relacionados aunque fueron desarrollados independientemente): *algoritmos genéticos*, *programación evolutiva* y *estrategias evolutivas*. Los algoritmos genéticos fueron propuestos inicialmente por Holland [Holl75] y estudiados por muchos autores como un modelo genérico de adaptación, pero con un claro acento en aplicaciones de optimización. La programación evolutiva fue introducida por Fogel [FOW66] y los trabajos posteriores han seguido su línea inicial de aplicación en dominios donde se intenta crear inteligencia artificial. Esta propuesta inicial se basaba en evolucionar máquinas de estados finitos (MEFs). Finalmente, las estrategias evolutivas, desarrolladas por Rechenberg [Rech73] y Schwefel [Schw81] y objeto actualmente de numerosas extensiones y estudios, fueron diseñadas en un principio para solucionar problemas difíciles de optimización con parámetros discretos y/o continuos (sobre todo en ingeniería). Consulte [Foge98] para una detallada disposición histórica de los avances en el campo de los algoritmos evolutivos.

De entre dicho conjunto de técnicas, este trabajo gira principalmente entorno a los algoritmos genéticos y a los procedimientos, consideraciones, conceptos, ventajas e inconvenientes de su paralelización. En aras de la completitud debemos mencionar en esta exposición introductoria dos grupos de algoritmos evolutivos que han tomado especial relevancia por sí mismos.

El primero de ellos es la *programación genética* que recibió el impulso definitivo como campo de trabajo separado gracias a las aportaciones de Koza [Koza92] [Koza94], y que en la actualidad tiene un enorme abanico de aplicaciones a problemas reales [Kinn94] [KGFR96]. Esta técnica evoluciona árboles sintácticos de símbolos frente a las estructuras de datos algo más simples que tradicionalmente manejan los algoritmos genéticos (de donde inicialmente se desgajaron como algoritmo de búsqueda). En segundo y último lugar encontramos los *sistemas de clasificación* (SC), objetivo inicial de los desarrollos de Holland, enfocados al aprendizaje máquina y al descubrimiento por evolución de las reglas de un sistema basado en reglas (SBR).

En todo este campo compuesto por numerosos procedimientos de búsqueda, la mayoría inspirados en procesos biológicos naturales, la inclusión del paralelismo ha sido siempre presentada como una forma de extender su aplicabilidad a problemas más complejos debido a las mejoras conseguidas en eficiencia. Esto se debe principalmente a dos razones. En primer lugar, la ejecución paralela permite un uso de mayores recursos de computación como son el tiempo de procesador y la memoria. En segundo lugar, la ejecución paralela de un algoritmo evolutivo cambia, a veces sustancialmente, su comportamiento, como consecuencia de la estructuración de la población en forma de grupos interconectados de individuos. Normalmente, el balance final es una mejora en el resultado obtenido y en la calidad de la búsqueda. Este segundo punto es importante y los distingue de otros algoritmos en los que la ejecución paralela únicamente provee un tiempo de espera menor antes de obtener la solución al problema.

En este capítulo centraremos nuestro trabajo en el campo de los algoritmos evolutivos. Para ello presentaremos una formalización de los algoritmos genéticos secuenciales y paralelos (Sección 1.1) y del tipo de problemas que pretendemos resolver (Sección 1.2). En la Sección 1.3 describimos las técnicas básicas consideradas de los algoritmos genéticos utilizados.

1.1. Computación Evolutiva

Este trabajo se enmarca en el campo de la computación evolutiva y es por tanto interesante localizar primero los algoritmos de interés entre el conjunto de algoritmos de búsqueda más conocidos. La Figura 1.1 presenta una taxonomía de técnicas de búsqueda en la que se incluyen los algoritmos evolutivos y otras técnicas de búsqueda. Puede observarse cómo se trata de heurísticos (no aseguran que el resultado de la búsqueda será un óptimo global) cuyo comportamiento es no determinista [AA92].

Además de ser heurísticos estocásticos, otra característica diferenciadora es la de trabajar con múltiples puntos del espacio de búsqueda a la vez. Esto contrasta con otras técnicas de seguimiento del gradiente descendente o similares que únicamente mantienen el último punto visitado. Adicionalmente, un algoritmo evolutivo no trabaja directamente sobre los parámetros del problema, sino sobre una representación de los mismos. En general, estos algoritmos no imponen restricciones sobre convexidad, continuidad, o similares como precondiciones para su aplicación.

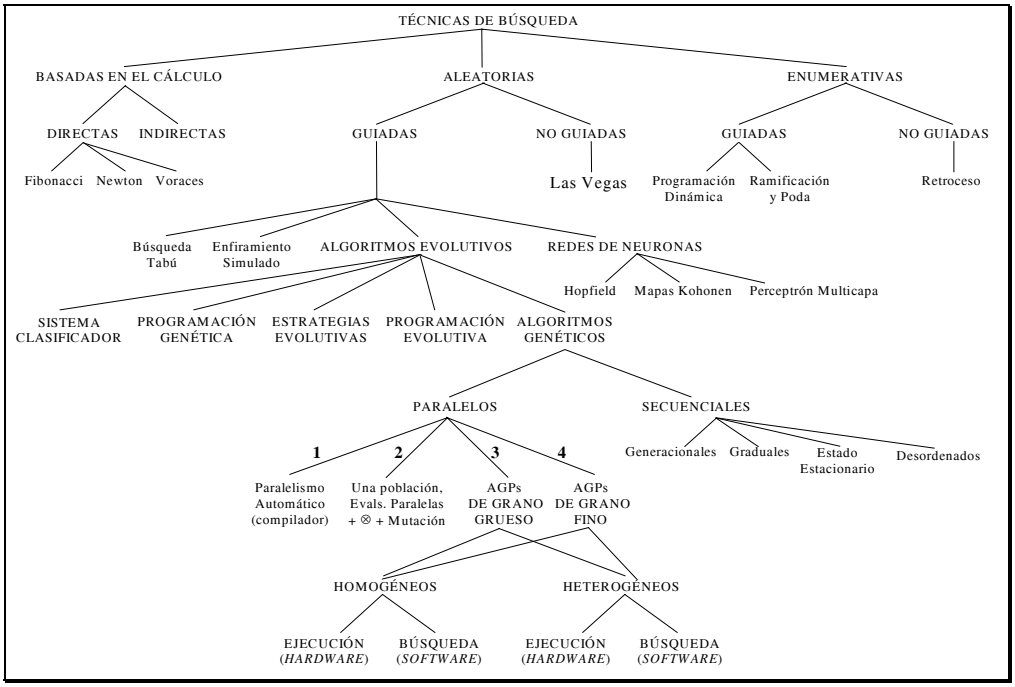


Figura 1.1. *Taxonomía de técnicas de búsqueda.*

Siguiendo una presentación de arriba-abajo podemos dar ahora otro contexto en el que típicamente se encuadra a la computación evolutiva. Nos referimos a la clasificación que describe a los AEs como una subclase de las técnicas conocidas como *Soft Computing* o *inteligencia computacional* [Bezd94] [Boni97] (Figura 1.2). Según esta clasificación muy reciente, la *computación evolutiva* (CE) junto con las *redes de neuronas artificiales* (RNs) y la *lógica difusa* (LD, borrosa o *fuzzy logic*), forman la tripleta de campos de trabajo en los que se busca solucionar problemas a través de técnicas algorítmicas basadas en la representación numérica -no simbólica- del conocimiento, al contrario que en el campo de la inteligencia artificial tradicional.

***Soft Computing* = CE + RN + LD**

Figura 1.2. *Los algoritmos evolutivos son considerados también técnicas de soft computing.*

Finalmente, disponemos ya de los elementos necesarios para presentar la computación evolutiva como suma de estudios, conceptos y resultados pertenecientes a los cinco campos mencionados en lo que daremos en llamar la *ecuación evolutiva* (Figura 1.3) [AC98].

CE = AG + PE + EE + PG + SC

[BFM97] [Gold89a] [FOW66] [Rech73] [Koza92] [Holl75]

Figura 1.3. *Ecuación evolutiva: familias de algoritmos en computación evolutiva.*

A continuación presentamos una primera definición genérica de problema de optimización que servirá para ir concretando más el tipo de tareas a las que se aplica un algoritmo evolutivo.

Definición 1.1. (Problema de Optimización Global). *De forma simple, podemos establecer que, un* problema de optimización global *como el que permite resolver un algoritmo evolutivo, requiere encontrar un vector de parámetros libres tal que se optimice (maximice o minimice) con él un cierto criterio de calidad conocido como* función objetivo

$$f(\vec{x}) \rightarrow max \tag{1.1}$$

La función objetivo puede venir definida por un sistema del mundo real de complejidad arbitraria. La solución a este problema de optimización global normalmente requiere encontrar un vector $\vec{x}^*$ *tal que* $\forall \vec{x} \in M : f(\vec{x}) \leq f(\vec{x}^*) := f^*$. ■

Esta definición de problema de optimización puede mejorarse (véase la Sección 1.2), pero por el momento nos permite conocer el ámbito de trabajo de los algoritmos evolutivos. Conceptos tales como multimodalidad, restricciones, óptimos locales, alta dimensionalidad, fuertes no-linealidades, imposible diferenciación, ruido y funciones multiobjetivo o cambiantes en el tiempo dan lugar en la práctica a problemas de optimización de difícil, cuando no imposible, solución por otros mecanismos. La eficiencia demostrada de los algoritmos evolutivos en estos campos los hace muy atractivos y merecedores de estudio.

Los problemas de optimización aparecen en disciplinas tan distintas como aplicaciones técnicas, en economía, minimización del tiempo o coste de proyectos, o maximización de beneficios. En todas ellas el desarrollo de estrategias de búsqueda global es de gran valor.

En la práctica, la función objetivo *f* es normalmente intratable analíticamente o incluso se desconoce su forma cerrada (por lo que se suelen usar modelos de simulación). En un enfoque tradicional de programación matemática deberíamos desarrollar un modelo formal lo suficientemente relajado para poder someterlo a técnicas típicas de optimización. Aunque estas técnicas son de indudable utilidad presentan el grave problema de que necesitan normalmente una excesiva simplificación, lo que conduce en ocasiones a que la solución encontrada no resuelva el problema real original.

De hecho, la función objetivo que admite un AE es a priori de dificultad arbitraria. De ahí el enorme abanico de aplicaciones existentes. En particular, con algoritmos evolutivos podemos solucionar problemas tan distintos como el diseño de redes de neuronas [AAT93a], asignación de procesos a procesadores [AAT95], diseño de controladores difusos [ACT96] o incluso estudios sobre validación de protocolos de comunicación [AT96]. La principal ventaja de los AEs radica en la flexibilidad de adaptar el método de búsqueda al problema.

Para completar la revisión de estos algoritmos las dos subsecciones siguientes presentan el funcionamiento de los AEs tanto secuenciales como paralelos, para después revisar en una tercera subsección los modelos más importantes de AGPs existentes en la actualidad.

1.1.1. Algoritmos Evolutivos Secuenciales

Los algoritmos evolutivos imitan el proceso de evolución natural, el principal mecanismo que guía la aparición de estructuras orgánicas complejas y bien adaptadas. De forma muy simplificada, la evolución es el resultado de las relaciones entre la *creación* de nueva información genética y los procesos de *evaluación*+*selección*.

Cada individuo en una población se ve afectado por el resto (compitiendo por recursos, emparejándose para procrear, huyendo de los depredadores, ...) y también por el entorno (disponibilidad de comida, clima, ...). Los individuos mejor adaptados son los que tienen mayores posibilidades de vivir más tiempo y reproducirse, generando así una progenie con su información genética (posiblemente modificada).

En el transcurso de la evolución se generan poblaciones sucesivas con información genética de los individuos cuya adecuación es superior a la de la media. La naturaleza no determinista de la reproducción provoca una creación permanente de información genética nueva, y por tanto la aparición de distintos individuos.

Este modelo *neo-Darwiniano* de la evolución orgánica se refleja en la estructura genérica de un algoritmo evolutivo (Algoritmo 1.1). El Algoritmo 1.2 es una definición en pseudocódigo un poco más detallada y cercana a la que usaremos para un algoritmo genético, quien centrará nuestro trabajo. Podemos observar que se trata de un algoritmo estocástico en el que es posible distinguir las tres etapas clásicas [WC97]: *generación de la muestra inicial*, *paso de optimización* y *comprobación de la condición de parada.*

ALGORITMO 1.1. (ALGORITMO EVOLUTIVO)

```
t := 0;
inicializar [P(t)];
evaluar [P(t)];
mientras no terminar hacer
    P'(t) := variación [P(t)];
    evaluar [P'(t)];
    P(t+1) := seleccionar [P'(t) ∪ Q];
    t := t +1 ;
fin mientras
```

ALGORITMO 1.2. (ALGORITMO EVOLUTIVO)

```
t := 0;
inicializar [P(t)];
evaluar [P(t)];
mientras no terminar hacer
    P'(t) := selección_pareja [P(t)];
    P'(t) := recombinación [P'(t)];
    P'(t) := mutación [P'(t)];
    evaluar [P'(t)];
    P(t+1) : =selecc_entorno [P'(t) ∪ P(t)];
    t := t +1 ;
fin mientras
```

En el Algoritmo 1.1 $P(t)$ denota una población de μ individuos en la generación t. Q es un conjunto de individuos adicional que podemos considerar para la selección. En el caso más genérico y normal $Q=P(t)$ como presentamos en el Algoritmo 1.2, aunque $Q=\varnothing$ es igualmente posible. El algoritmo genera una nueva población $P'(t)$ de λ individuos aplicando a $P(t)$ un conjunto de operadores de variación. Típicamente tenemos el caso del algoritmo 1.2 en el que dichos operadores son la selección y recombinación de parejas junto con la mutación de los nuevos individuos generados. Los operadores de variación son en general no deterministas, viniendo por tanto su comportamiento regulado por reglas de transición probabilísticas.

La evaluación sirve para asociar un valor de calidad, calculado por la función objetivo $f(\vec{x}_k)$, para cada solución $\vec{x}_k$ representada por el individuo k-ésimo de $P'(t)$, k:1, ..., λ. El proceso de selección en un AE consta de dos etapas: la primera decide quiénes compiten por la reproducción (emparejamiento) y la segunda decide cuáles de entre todos los individuos (nuevos y viejos) van a sobrevivir, simulando así el proceso de selección del entorno (o ambiental). Ambas etapas de la selección utilizan de alguna manera los valores de adecuación (*fitness*) calculados en la evaluación y asociados a cada individuo para guiar el proceso hacia soluciones mejores.

Ya estamos en disposición de identificar los valores de algunos parámetros que dan lugar a los distintos algoritmos secuenciales que vamos a estudiar. En concreto, podemos definir tipos distintos de esquemas de selección como sigue:

Definición 1.2. (Esquemas de Selección). *Dada la descripción abstracta de algoritmo evolutivo presentada en el Algoritmo* 1.1*, sea Q=P(t). Entonces podemos distinguir los tipos* de esquemas de selección genéricos *de las definiciones* **1.2.1***,* **1.2.2***,* **1.2.3** *y* **1.2.4** *variando el valor del parámetro* λ*.* ■

Definición 1.2.1. (Selección por Estado Estacionario). *Cuando* λ=1 *tenemos una* selección por estado estacionario *(steady-state)* [Whit89] [Swys91] *en la que únicamente se genera un hijo en cada paso de la evolución.* ■

Definición 1.2.2. (Selección Generacional). *Cuando* $\lambda=\mu$ *tenemos una* selección por generaciones [Gold89a] *en la que se genera una nueva población completa de individuos en cada paso de la evolución.* ■

Definición 1.2.3. (Selección Ajustable). *Cuando* $1 \leq \lambda \leq \mu$ *tenemos una* selección intermedia *en la que se calcula un número ajustable (generation gap) de individuos en cada paso de la evolución. Por tanto las definiciones* 1.2.1 *y* 1.2.2 *son casos particulares de ésta para* λ=1 *y* $\lambda=\mu$*, respectivamente.* ■

Definición 1.2.4. (Selección por Exceso). *Cuando* $\lambda>\mu$ *tenemos una* selección por exceso *típica de los procesos naturales reales.* ■

En general el uso de selección generacional pura era usual tras la aparición de los algoritmos evolutivos, aunque cada vez más se tiende a utilizar evolución intermedia ajustable y de estado estacionario, teniendo en cuenta de esta forma a las estructuras presentes en la población anterior para decidir quién compone la nueva generación (Figura 1.4).

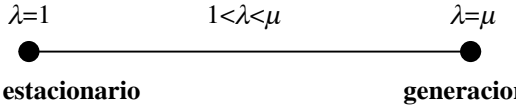


Figura 1.4. *Diferencias entre evolución en estado estacionario y generacional.*

Es normal en los algoritmos evolutivos que el tamaño de la población permanezca constante a lo largo del proceso de optimización. Asimismo, existe un elevado número de operadores de variación, sobre todo versiones de los más populares operadores de selección, recombinación y mutación. Incluimos los detalles técnicos relevantes para este trabajo en la Sección 1.3.

Sin embargo, podemos mejorar más la descripción de los distintos tipos de algoritmos evolutivos si estudiamos en profundidad el operador de selección. Este operador es quizás el que más caracteriza a un algoritmo evolutivo. De forma genérica, el bucle de trabajo de un AE puede visualizarse como presentamos en la Figura 1.5. En ella podemos observar los tres elementos más importantes que caracterizan la selección. Las diferentes ternas de valores para estos tres parámetros permiten diferenciar el funcionamiento de los operadores de selección más conocidos. Los parámetros más influyentes son: el *tamaño de la población* μ, el *número de descendientes* generados λ y finalmente el *tiempo de vida máximo* (edad) k de los individuos de la población. Los tres parámetros suelen ser constantes durante toda la evolución del AE.

La Figura 1.5 permite ver cómo en una población de μ miembros existe una porción que ya habrán alcanzado la edad máxima, a quienes no se les permitirá ya producir descendientes en el futuro. Sólo los individuos de edad menor a la máxima participan en los procesos de selección. De entre ellos la selección determina parejas formando un subconjunto llamado multiconjunto (*multiset*) que sufrirán los efectos de los operadores de variación.

Tras la generación de λ descendientes, la selección del entorno (*enviromental selection*) determina quiénes formarán la nueva población. Primero debe decidirse quiénes serán los candidatos: sólo los descendientes o también los miembros de la población antigua. La primera posibilidad se denomina selección (μ, λ), siendo $\mu \leq \lambda$, y la segunda posibilidad se denomina selección $(\mu + \lambda)$.

Por tanto, de los λ $(\mu + \lambda)$ candidatos se seleccionan los μ individuos que formarán la nueva población. La edad de los miembros que sobreviven de la población antigua se incrementa en uno y los descendientes recién creados elegidos para sobrevivir reciben una edad de 0. Típicamente esta selección del entorno determinísticamente decide que sobrevivan los mejores μ individuos (*elitismo*), pero existen numerosas variantes útiles. En principio, el elitismo permite que uno o más individuos bien adecuados al problema puedan llegar a existir durante toda la vida del algoritmo.

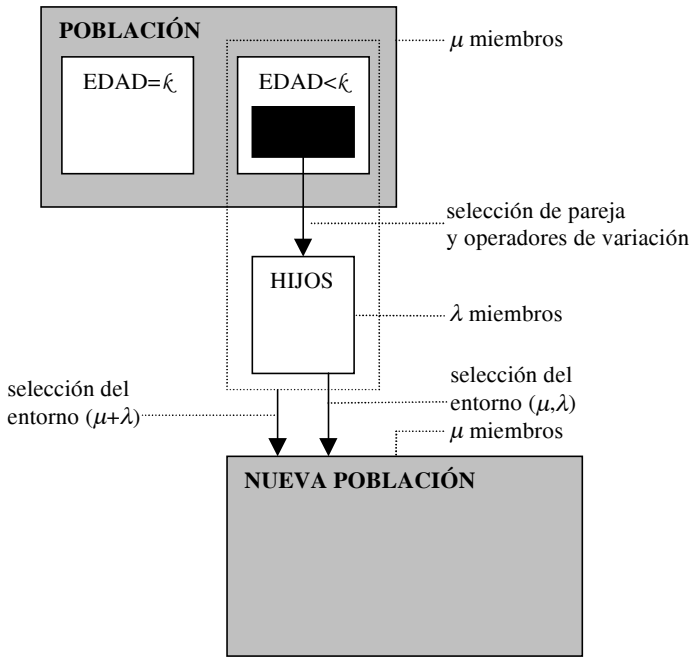


Figura 1.5. *La próxima generación.*

Con la intención de introducir las principales diferencias entre familias de algoritmos evolutivos podemos visualizar el diagrama conocido como cubo evolutivo [BGKK97]. Un AE cualquiera puede visualizarse en él como un punto representado por las coordenadas (μ, λ, κ), con $\mu, \kappa \in [1..\infty)$ y $\lambda \in [0..\infty)$.

En la Figura 1.6 los puntos negros indican tipos genéricos de AEs con diferentes parámetros. Nótese por ejemplo que, ya que la selección (μ, λ) reemplaza la población antigua, el parámetro k vale siempre uno, mientras que para la selección $(\mu + \lambda)$, k tiende a infinito. Además, en la selección (μ, λ) debe cumplirse que $\mu \leq \lambda$. Esta condición suele incluso respetarse (aunque no es obligatoria) en las estrategias evolutivas, donde es normal utilizar selección $(\mu + \lambda)$.

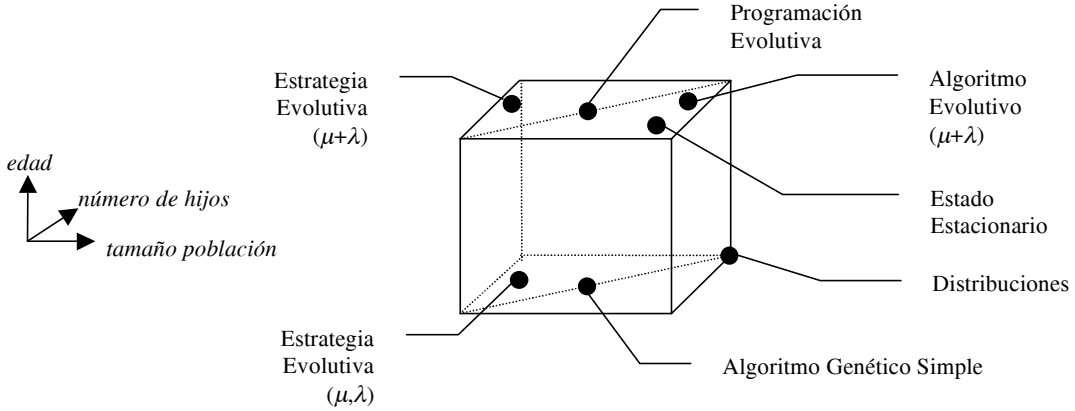


Figura 1.6. *El cubo evolutivo.*

Los algoritmos genéticos tradicionales pueden verse como un tipo de AE que utiliza una selección (μ, μ) como mecanismo de selección ambiental (del entorno). En un AE de estado estacionario realmente se utiliza selección $(\mu + 1)$, pues se crea un único descendiente en cada paso. Un algoritmo evolutivo (1 + 1) es realmente un proceso de seguimiento del gradiente descendente. Finalmente, un algoritmo de programación evolutiva tradicional [FOW66] produce un descendiente a partir de cada padre, y tanto padres como hijos se utilizan para generar la próxima población. Esto puede verse como la utilización de un esquema de selección $(\mu + \mu)$.

Aunque hemos mejorado la caracterización de los AEs, el cubo evolutivo no proporciona todos los detalles sobre el mecanismo de selección. Los tipos de selección serán objeto de estudio en la sección 1.3.1.2 (representan el 'interior' de las flechas de la Figura 1.5).

Presentamos a continuación los principales ingredientes formales necesarios para definir un algoritmo evolutivo secuencial. La descripción es una extensión y modificación del trabajo de [Bäck96] que permite el trasvase de conocimientos debido a la descripción y estudio unificados de los algoritmos evolutivos secuenciales. De la misma manera, se pretende proporcionar en el presente trabajo una descripción y estudio unificados de modelos paralelos y distribuidos de algoritmos genéticos con los que se espera obtener las mismas ventajas que en el campo de los modelos secuenciales.

Definición 1.3. (Algoritmo Evolutivo Secuencial). *Un* algoritmo evolutivo secuencial *(AE) se define como una tupla de 7 elementos:*

$$AE = (I, \Phi, \Omega, \Psi, \iota, \mu, \lambda) \tag{1.2}$$

donde $I = \mathcal{G} \times \mathbb{R}$ *es el* espacio de individuos, *siendo* $\mathcal{G}$ *el* genotipo *al que se asocia un* valor de adecuación *perteneciente al conjunto de los reales* $\mathbb{R}$.

Para asignar a cada individuo un valor de adecuación se utiliza la función de adecuación $\Phi : \mathcal{G} \rightarrow \mathbb{R}$. *A su vez, definimos el* fenotipo $\mathcal{P}$ *como el conjunto de valores resultantes de expresar el genotipo* $\mathcal{G}$ *en parámetros con significado para el problema.*

De esta forma $\Gamma : \mathcal{P} \rightarrow \mathcal{G}$ *es la* función de codificación *que permite representar los parámetros del problema como puntos del espacio de genotipos. La función inversa* Γ^{-1} *es la que se utiliza siempre sobre un individuo antes de someterlo a evaluación.*

La función de adecuación puede someter a cierta transformación el valor devuelto por la función objetivo antes de considerarlo como valor de adecuación del individuo. Para ello se utiliza una función $\varphi : \mathbb{R} \rightarrow \mathbb{R}$. *El objetivo es asegurar valores positivos de adecuación y separar dichos valores entre sí de forma que se facilite el trabajo del operador de selección.*

Por tanto, dada una estructura $\vec{a} \in \mathcal{G}$ el proceso de evaluación implica:

$$\Phi(\vec{a}) = \varphi\left(f\left(\Gamma^{-1}(\vec{a})\right)\right) \tag{1.3}$$

La componente Ω *representa un* conjunto de operadores genéticos *de aplicación no determinista* ω_{Θ_i} *cada uno de ellos controlado por un conjunto de parámetros* $\Theta_i \subset \mathbb{R}^{\phi_i}$. *El valor* ϕ_i *es el* número de parámetros *que controlan el funcionamiento del operador i-ésimo,* $s = \omega_{\Theta_0}$ *es el operador de* selección de parejas *y* $r = \omega_{\Theta_{z-1}}$ *es la* selección ambiental *(reemplazo). Si notamos como z el* número de operadores *entonces el conjunto de operadores se define como sigue*:

$$\Omega = \{\omega_{\Theta_1},...,\omega_{\Theta_{z-2}} \mid \omega_{\Theta_i} : I^{\lambda} \rightarrow I^{\lambda}\} \cup \{\omega_{\Theta_0} : I^{\mu} \rightarrow I^{\lambda}\} \cup \{\omega_{\Theta_{z-1}} : I^{\lambda} \cup I^{\lambda+\mu} \rightarrow I^{\mu}\} \tag{1.4}$$

También se permite $\mu = \lambda$. *Por otro lado,* siguiendo con la descripción de los elementos de la *tupla, los valores* $\mu, \lambda \in \mathbb{N}$ *son valores naturales, indicando el* tamaño de la población *y el* tamaño del conjunto de descendientes *generados por los operadores, respectivamente*.

Finalmente $\iota : I^{\mu} \rightarrow \{\textbf{true}, \textbf{false}\}$ *es el* criterio de terminación *del AE. La* transición *entre dos generaciones viene gobernada por la función* $\Psi : I^{\mu} \rightarrow I^{\mu}$ *quien describe el proceso completo de transformación de una población P(t) en la siguiente por aplicación del* plan adaptativo*, es decir, la composición ordenada de operadores desde la selección de parejas hasta la ambiental:*

$$\begin{aligned} \Psi &= \omega_{\Theta_{z-1}} \circ \omega_{\Theta_{i_1}} \circ ... \circ \omega_{\Theta_{i_j}} \circ \omega_{\Theta_0} \\ \Psi(P(t)) &= \omega_{\Theta_{z-1}}(Q \cup \omega_{\Theta_{i_1}}(...(\omega_{\Theta_{i_j}}(\omega_{\Theta_0}(P(t)))))...)) \end{aligned} \tag{1.5}$$

donde $\{i_1, ..., i_j\} \subseteq \{1, ..., z\text{-}2\}$ y $Q \in \{\varnothing, P(t)\}$. ■

La definición anterior admite numerosas variantes consecuencia de modificaciones en la definición de operadores, representación de los individuos, tipo de la evaluación, criterios de parada, etc. Aunque el espacio de búsqueda es arbitrariamente complejo, tradicionalmente los AGs se han ceñido a un espacio vectorial en el conjunto de los reales. Los parámetros de cada operador utilizado determinan comportamientos distintos del mismo algoritmo. El operador de selección puede ser tanto determinista como no determinista. El criterio de terminación puede ser tan simple como alcanzar un número preestablecido de generaciones o tan complejo como una condición dependiente de alguna métrica (Definición 1.10) sobre los individuos de la población.

Un algoritmo evolutivo itera un cierto número de veces generando una secuencia de poblaciones (Definición 1.4) e identificando como solución (Definición 1.5) una de las estructuras de la población aparecida durante el tiempo de ejecución (Definición 1.6) transcurrido hasta satisfacer la condición de terminación.

Definición 1.4. (Secuencia de Poblaciones). *Dado un AE con una función de transición entre generaciones* $\Psi : I^{\mu} \rightarrow I^{\mu}$ *y una población inicial* $P(0) \in I^{\mu}$, *la secuencia* $P(0)$, $P(1)$, $P(2)$, *... se denomina* secuencia de poblaciones -o evolución de $P(0)$- $\Leftrightarrow$

$$\forall t \geq 0 \quad : \quad P(t+1) = \Psi(P(t)) \quad . \tag{1.6}$$

■

La creación de la población inicial $P(0)$ se suele realizar de forma aleatoria aunque en algunas aplicaciones prácticas puede utilizarse información dependiente del problema para generar un conjunto de puntos de búsqueda iniciales que presenten a priori ciertas ventajas [Reev93b].

Típicamente se distinguen dos modelos de algoritmos evolutivos atendiendo a si se determina como solución una única estructura o bien una población completa [Gold89a] [DeJo92]. En este trabajo utilizaremos el primer criterio conocido como modelo *Pittsburg*, frente al segundo criterio (típico de sistemas para aprendizaje máquina) conocido como modelo *Michigan*.

Por tanto, podemos definir como *resultado* la estructura, de entre todas las generadas durante el proceso de búsqueda, que provoque un mayor valor de la función objetivo:

Definición 1.5. (Resultado de un AE). *Dado un AE con una función de transición entre generaciones* Ψ *y una población inicial* $P(0) \in I^{\mu}$ *se define como* resultado *al individuo* $\vec{a}$ *sii*

$$\vec{a} \in \bigcup_{t=0}^{\tau_{AE}} \Psi^t(P(0)) \tag{1.7}$$

y además se cumple que $f(\vec{a}) = \max\left\{ f(\vec{a}') \mid \vec{a}' \in \bigcup_{t=0}^{\tau_{AE}} \Psi^t(P(0)) \right\}$. (1.8)

■

En la definición anterior utilizamos el concepto de *tiempo de ejecución* del AE y lo notamos como τ_{AE}, quedando definido éste como el menor número de generaciones que satisface el criterio de terminación (vea la siguiente definición).

Definición 1.6. (Tiempo de Ejecución). *Dado un AE con una función de transición entre generaciones* Ψ *y una población inicial* $P(0) \in I^{\mu}$ *se define como* tiempo de ejecución *a* τ_{AE} *sii*

$$\tau_{AE} = \min\left\{ t \in IN \mid \iota\left(\Psi^t(P(0))\right) = \textbf{true} \right\}. \tag{1.9}$$

■

La definición de tiempo de ejecución en término de generaciones es diferente cuando se comparan algoritmos de distintos gránulos de generación, es decir, dependiente del número de descendientes generados λ en cada paso. De esta forma, ya que en este trabajo realizamos un estudio comparativo entre algoritmos cuyo paso evolutivo es diferente, es importante determinar un criterio inequívoco de comparación como es el *número de evaluaciones* de la función objetivo que se ha realizado en cada uno de los algoritmos comparados.

Para terminar esta sección, y a modo de resumen, vamos a expresar con estos formalismos el funcionamiento de un AE secuencial en el algoritmo siguiente (Algoritmo 1.3).

ALGORITMO 1.3. (ALGORITMO EVOLUTIVO SECUENCIAL)

$t := 0;$
inicializar: $P(0) := \{\vec{a}_1(0),...,\vec{a}_\mu(0)\} \in \mathcal{G}^\mu$;
evaluar: $P(0) : \{\Phi(\vec{a}_1(0)),...,\Phi(\vec{a}_\mu(0))\};$
mientras $\iota(P(t)) \neq$ **true hacer**
  seleccionar: $P'(t) := s_{\Theta_s}(P(t));$
  recombinar: $P''(t) := \otimes_{\Theta_c}(P'(t));$
  mutar: $P'''(t) := m_{\Theta_m}(P''(t));$
  evaluar: $P'''(t) : \{\Phi(\vec{a}_1'''(0)),...,\Phi(\vec{a}_\lambda'''(0))\};$
  reemplazar: $P(t+1) := r_{\Theta_r}(P'''(t) \cup Q);$
  $t := t+1$;
fin mientras

La similitud de este algoritmo con su pseudocódigo (Algoritmo 1.2) atendiendo a las definiciones proporcionadas hacen que sea un claro representante de la mayoría de algoritmos evolutivos. Los operadores de selección de pareja, cruce, mutación y reemplazo (s, $\otimes$, m, r) utilizan cada uno de ellos su propio conjunto de parámetros Θ_i. Los operadores de cruce y mutación se han descrito en su versión de alto nivel, es decir, aplicados sobre el conjunto de la población. En la siguiente sección se refina su comportamiento en términos de su *aridad* y *panmixia*.

1.1.2. Algoritmos Evolutivos Paralelos

La ejecución real de un algoritmo evolutivo secuencial requiere de una población de soluciones tentativas que representan puntos del espacio de búsqueda, sobre las que se realizan operaciones para guiar la búsqueda. En la práctica, esto supone que un AE utiliza recursos de computación como memoria física y tiempo del procesador de forma considerable.

Existen básicamente dos enfoques para disminuir el tiempo de ejecución real: (1) disminuir de alguna manera el número de evaluaciones necesario para alcanzar una solución y (2) ejecutar el algoritmo en una máquina paralela. Como veremos, los modelos paralelos de estos algoritmos son interesantes porque consiguen ambos objetivos, ya que modifican el comportamiento típico del algoritmo secuencial equivalente mediante el uso de una población estructurada.

La inclusión del paralelismo en las operaciones de un AE da lugar a un AE paralelo (AEP) [BB93] [CF96]. Un AEP permite pues utilizar poblaciones de mayor tamaño, resolver problemas de mayor dimensión y complejidad, y reducir el tiempo real de espera por una solución. Además de estas ventajas planteadas globalmente, veremos más adelante cómo algunos tipos de AEPs pueden cambiar su comportamiento debido a la forma especial de inclusión del paralelismo. Por el momento, ya disponemos de los elementos mínimos para delinear el funcionamiento de un algoritmo evolutivo paralelo (Algoritmo 1.4) a partir de la versión secuencial.

ALGORITMO 1.4. (ALGORITMO EVOLUTIVO PARALELO)

AEP^i
$t^i := 0$;
`inicializar` $[P^i(t^i)]$;
`evaluar` $[P^i(t^i)]$;
mientras `no terminar` **hacer**
$P^i{}'(t^i)$:= `selección_pareja` $[P^i(t^i)]$;
$P^i{}'(t^i)$:= `recombinación` $[P^i{}'(t^i)]$;
$P^i{}'(t^i)$:= `mutación` $[P^i{}'(t^i)]$;
`evaluar` $[P^i{}'(t^i)]$;
$P^i{}'(t^i)$:= `comunicación` $[P^i{}'(t^i) \cup \mathbf{AEP}^i{::}P^j(t^j)]$; *// Interacción con los vecinos*
$P^i(t^i+1)$:= `selecc_entorno` $[P^i{}'(t^i) \cup P^i(t^i)]$;
$t^i := t^i + 1$;
fin mientras

Como puede observarse, un algoritmo evolutivo paralelo extiende la versión secuencial incluyendo una fase de *comunicación* con un vecindario de otros algoritmos evolutivos. La forma de realizar esta comunicación, el tipo de operaciones del resto de algoritmos en paralelo y otros detalles de funcionamiento determinan su comportamiento global. Debido a la existencia de numerosas variantes trasladamos al Capítulo 2 los detalles sobre AEPs. Sin embargo, podemos observar que este modelo permite también una fácil composición con algoritmos no evolutivos, lo que constituye otra rama interesante de trabajo. Obsérvese en el Algoritmo 1.4 que notamos la generación de cada subalgoritmo con un superíndice t^i para constatar el hecho de que distintos subalgoritmos pueden estar en distintas etapas de su evolución.

De entre las operaciones de un algoritmo evolutivo secuencial el mecanismo de selección es el único que obligatoriamente involucra a toda la población. Este hecho es muy relevante para la paralelización, y por tanto es merecedor de un estudio detenido. Puede caracterizarse formalmente al operador de selección clasificándolo en tres tipos: operadores *unarios*, *binarios* y operadores que utilizan toda la población (*panmixia*):

Definición 1.7. (Operadores Genéticos Unarios, Binarios y que usan Panmixia). *Un operador genético* $\omega_\Theta : I^p \to I^q$ *se denomina*

$$\begin{aligned}
&\text{unario} \quad :\Leftrightarrow \quad \exists \omega'_\Theta : I \to I : \\
&\qquad \omega_\Theta(\vec{a}_1,...,\vec{a}_p) = (\omega'_\Theta(\vec{a}_1),...,\omega'_\Theta(\vec{a}_p)) \wedge p = q \\
&\text{binario} \quad :\Leftrightarrow \quad \exists \omega'_\Theta : I^2 \to I : \\
&\qquad \omega_\Theta(\vec{a}_1,...,\vec{a}_p) = (\omega'_\Theta(\vec{a}_{i_1},\vec{a}_{j_1}),...,\omega'_\Theta(\vec{a}_{i_q},\vec{a}_{j_q})) \\
&\qquad \textit{donde} \quad \forall k \in \{1,...,q\} \quad \textit{se eligen aleatoriamente} \\
&\qquad i_k, j_k \in \{1,...,p\} \\
&\text{panmixia} \quad :\Leftrightarrow \quad \exists \omega'_\Theta : I^p \to I : \\
&\qquad \omega_\Theta(\vec{a}_1,...,\vec{a}_p) = \underbrace{(\omega'_\Theta(\vec{a}_1,...,\vec{a}_p),...,\omega'_\Theta(\vec{a}_1,...,\vec{a}_p))}_{q}
\end{aligned} \tag{1.10}$$

■

Los operadores unarios como la mutación, y los binarios como el cruce, no presentan problemas en la definición de un modelo paralelo. Sin embargo, el operador de selección, (parejas y ambiental), representa un cuello de botella en el sistema paralelo ya que usa panmixia.

Más adelante presentaremos la evolución y los tipos de algoritmos evolutivos paralelos para encuadrar nuestra propuesta. Por el momento debemos centrar nuestra atención en los aspectos que son susceptibles de paralelizar. De esta forma, encontramos que los *operadores que no usan panmixia* pueden ejecutarse a la vez sobre diferentes porciones de la población. Por otro lado, las *operaciones de evaluación* de la adecuación de los descendientes pueden ser igualmente paralelizadas sin provocar ningún cambio en el modelo secuencial básico. Con estos dos tipos de paralelización, y manteniendo una selección centralizada, damos lugar a un algoritmo más rápido en tiempo real pero con las mismas características de búsqueda que si no estuviese trabajando en paralelo. A los modelos resultantes se les denomina de **paralelización global** (Figura 1.1, tipo 2) [Cant97b]. Cuando la paralelización no la establece el usuario sino que ocurre a nivel de compilación de las instrucciones del programa se le denomina **automática** (Figura 1.1, tipo 1).

La otra gran vertiente que conduce a la paralelización de AGs es la de abandonar la visión de una población única y separarla en *subpoblaciones* de cierto tamaño que se comuniquen entre sí de alguna forma. Es en este sentido en el que la paralelización se ha mostrado con frecuencia más efectiva que su contrapartida secuencial. Los algoritmos así paralelizados demuestran comportarse de forma distinta a la secuencial y, en general, mejoran la búsqueda. Este hecho es lógico, ya que, continuando con la idea de copiar el funcionamiento de la evolución natural, no deberíamos permitir que cualquier individuo se empareje con cualquier otro de la población global (panmixia). Por el contrario, es normal la evolución en grupos separados, dando lugar a *especies* (*demes*) o *nichos*. De esta manera, una población de individuos separada en grupos de reproducción casi-aislada se conoce como *politípica*, y es la visión natural equivalente a la ejecución paralela en islas de individuos típica de los algoritmos evolutivos distribuidos.

Atendiendo a esta última visión encontramos que podemos distinguir entre **modelos de grano grueso** [Tane89] [Beld95] [Gold95] y **grano fino** [MS89] [SM91] [MSB91], según que la relación entre el tiempo de la fase de computación y la fase de comunicación sea alta o baja, respectivamente (tipos 3 y 4 de la Figura 1.1). En un algoritmo de grano grueso el tamaño de la población es apreciable $|P^i(t^i)|=\mu/d$, donde d es el número de subalgoritmos, mientras que en uno de grano fino cada subalgoritmo contiene una sola estructura $|P^i(t^i)|=1$. En este trabajo aportamos la visión de considerar que la principal diferencia frente a los algoritmos tradicionalmente llamados secuenciales es esencialmente que utilizan una *población estructurada espacialmente*.

La Figura 1.7 muestra lo que nosotros hemos dado en llamar el *cubo de poblaciones estructuradas* en el que puede observarse que realmente las diferencias deben establecerse entre los modelos, y no entre sus implementaciones, ya que cada modelo admite múltiples. El modelo celular consta de un elevado número de subalgoritmos con frecuentemente comunicación entre ellos, cada uno con unos pocos individuos (usualmente sólo uno). Por el contrario, un modelo distribuido consta de pocos subalgoritmos de gran población y realiza intercambios esporádicos de información. En medio existe un "continuo" de posibles modelos, a veces de difícil clasificación. Respecto a la implementación, parece evidente que los modelos celulares encajan fácilmente en un sistema SIMD, mientras que los distribuidos lo hacen en uno MIMD, pero eso no significa que sean los únicos tipos de implementación posibles.

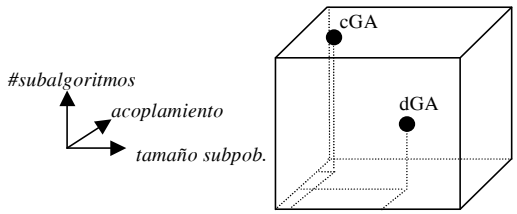


Figura 1.7. *Cubo explicativo de algoritmos con población estructurada.*

Podríamos decir que la nomenclatura antigua "secuencial/paralelo" puede ser en ocasiones equívoca. Proponemos hablar de algoritmos "celulares" frente a "distribuidos" en vez de algoritmos "de grano fino" frente a algoritmos "de grano grueso". Sin embargo, en muchas ocasiones las descripciones de modelos existentes hablan de implementaciones sobre cierto sistema *hardware*, porque en muchas ocasiones la relación es de uno a uno.

La formalización de estos conceptos se proporciona en el capítulo siguiente, pero parece claro que el proceso de crear subpoblaciones interconectadas no implica que estas subpoblaciones deban trabajar atendiendo a un AE de población única. Con esta visión del paralelismo en que grano fino y grueso no son considerados como alternativas y hablamos de algoritmos de población estructurada, se abren nuevas expectativas [AT99a]. Un AGP tradicional de granularidad combinada es un modelo AG estructurado en dos o más niveles. Esto puede ser ventajoso debido a que es posible combinar algunas de las ventajas relativas de cada modelo sin, al menos, acentuar sus respectivos inconvenientes (Figura 1.8).

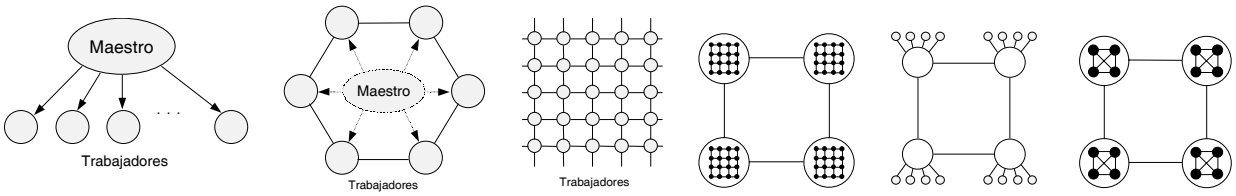


Figura 1.8. *Esquemas de un AG* (a) *con paralelización global,* (b) *paralelo de grano grueso, y* (c) *paralelo de grano fino. Existen numerosos modelos paralelos híbridos que combinan a alto y bajo nivel* (d) *grano grueso y grano fino,* (e) *grano grueso y paralelización global y* (f) *grano grueso tanto a alto como a bajo nivel (nomenclatura tradicional).*

Es importante resaltar que en los modelos distribuidos se introduce el concepto de **homogeneidad**. Este concepto nos permite distinguir dos niveles: *nivel de búsqueda y nivel de ejecución*. A nivel de búsqueda (*software*), un algoritmo de población estructurada es homogéneo si todas los subalgoritmos que lo componen aplican el mismo plan adaptativo sobre sus respectivos datos (paralelismo de datos y simetría textual) y es heterogéneo si no se cumple esta condición. A nivel de ejecución (*hardware*), si los distintos subalgoritmos se ejecutan sobre plataformas del mismo tipo, entonces el algoritmo es *homogéneo*. En otro caso, se trata de un algoritmo evolutivo *heterogéneo*.

En nuestro caso trabajamos con algoritmos homogéneos tanto a nivel de búsqueda como a nivel de ejecución. La heterogeneidad en ejecución es casi inexistente en este campo aunque la heterogeneidad a nivel de búsqueda está progresando con algunos resultados interesantes [LPG94] [AP96] [HL97].

Debemos tener en cuenta que los modelos que surgen por paralelización *no restringen el modelo genérico de algoritmo evolutivo*. Por un lado, esto significa que los desarrollos existentes sobre algoritmos evolutivos pueden extenderse a los modelos paralelos estructurados. Por otro lado, esto implica también que podemos mejorar la eficiencia en problemas donde ya se aplican los algoritmos evolutivos secuenciales o de población única, además de ser posible abordar nuevos problemas casi intratables hasta el momento.

En consecuencia, cualquier modificación del comportamiento de un AE secuencial tal como introducir conocimiento dependiente del problema es también susceptible de aplicarse al modelo paralelo, con la importante diferencia de que el modelo paralelo es de partida más genérico y potente en la mayoría de los casos que el secuencial.

1.1.3. Revisión y Clasificación de Trabajos Relacionados

En esta sección presentaremos una revisión de los principales algoritmos, clasificados como secuenciales o paralelos, encontrados en la bibliografía asociada a este campo. Pretendemos con ello dar un cauce lógico que plantee la necesidad de resolver problemas complejos usando algoritmos paralelos y estructurados. Igualmente, se discuten las principales ventajas de usar un algoritmo genético de población estructurada (Tabla 1.1) y un resumen de sus aplicaciones que ayude a justificar la utilidad de nuestro trabajo en este campo. Esta sección será intencionadamente introductoria y descriptiva debido a que nuestra aportación se centra en este terreno y dedicaremos los siguientes capítulos a refinar y formalizar los conceptos que aquí presentamos intuitivamente. En muchos puntos nos referimos a algoritmos paralelos por ser tradicional describir dichos trabajos de esta manera.

TABLA 1.1. VENTAJAS DE USAR UN AG DE POBLACIÓN ESTRUCTURADA

Ventajas de Utilizar un Algoritmo Genético de Población Estructurada	
❶ Trabaja sobre una codificación del problema	❺ Fácil paralelización como islas o vecindarios
❷ Básicamente independiente del problema (robusto)	❻ Mejores resultados incluso sin *hardware* paralelo
❸ Proporciona múltiples soluciones al mismo tiempo	❼ Mayor eficiencia e incluso porcentaje de éxitos
❹ Búsqueda paralela desde distintos puntos del espacio	❽ Fácil cooperación con métodos no evolutivos

Existen numerosos modelos de AGPs [Gorg89] [MS89] [Tane89] [WS90] [Davi91b]. Todos ellos pueden clasificarse en los cuatro tipos básicos de la Figura 1.1 ya mencionados. Los modelos de paralelización global, aunque son interesantes, tienen una aplicación reducida en la actualidad, ya que, para que merezca la pena distribuir así, la función objetivo debe ser muy costosa de evaluar. Además, queda el problema del cuello de botella en la selección del maestro, aunque nada impide la paralelización de la selección, mientras siga utilizando los valores de adecuación de toda la población. Consulte recientes trabajos están en la línea en [CG97].

El segundo tipo de modelos paralelos se basa en aplicar a un AG secuencial un compilador que paralelice automáticamente allí donde el código lo permita. Este tipo de paralelismo es poco usual en computación evolutiva.

El resto de modelos, quienes centran actualmente la atención de la comunidad científica, son AGPs de *grano fino* o *grano grueso*, o más correctamente, distribuidos y celulares.

Los modelos de distribuidos son con mucho los más populares [FAT94] debido a que de forma clara permiten extender la ejecución secuencial a un grupo de islas AG que se ejecutan en paralelo e intercambian con cierta frecuencia entre sí un conjunto de individuos u otra información, atendiendo a cierta topología como un anillo, árbol, estrella, ...

Los modelos celulares típicamente sitúan cada individuo de la población en una disposición bidimensional (como por ejemplo una malla toroidal) y hacen interaccionar a cada individuo exclusivamente con sus vecinos atendiendo a cierta definición de *vecindario*. La migración en este caso se da de forma continua en la rejilla (malla) y además transparentemente, debido al solapamiento de vecindarios.

Los modelos distribuidos están débilmente acoplados (de hecho la evolución separada durante un alto número de pasos es lo más indicado, sin llegar al aislamiento total) mientras que los algoritmos celulares están fuertemente acoplados, debido a las interacciones de cada individuo con su vecindario. Típicamente los modelos distribuidos envían al subalgoritmo destino una copia del mejor o de un individuo aleatorio, mientras que los modelos celulares deben utilizar vecindarios reducidos que permitan un trabajo descentralizado. La siguiente tabla resume las características de ambos tipos de modelos y su paralelización.

TABLA 1.2. RESUMEN DE CARACTERÍSTICAS DE LOS MODELOS CELULARES Y DISTRIBUIDOS

AGPs Distribuidos	AGPs Celulares
Débilmente acoplados (fácil implementación distribuida)	Fuertemente acoplados (paralelismo físico masivo)
Típicamente islas en un anillo con migración esporádica	Interacción de cada individuo sólo con su vecindario
Típicamente implementados en redes locales (*clusters*)	Típicamente implementados en máquinas SIMD
Múltiples subpoblaciones evolucionan a distintas solucs.	Múltiples soluciones intermedias se difunden y compiten
Fácil cooperación con otras técnicas no evolutivas	Alta eficacia en problemas complejos

La Tabla 1.3 resume de forma ordenada en el tiempo la aparición y algunas características básicas de implementaciones paralelas de modelos distribuidos o celulares que permitan obtener una idea clara de la diversidad existente, así como de los factores comunes compartidos por todas ellas.

En la Tabla 1.3 presentamos algunos modelos como dGA, DGENESIS o GALOPPS que siguen fielmente el estándar tradicional de grano grueso haciendo hincapié en presentar un sistema flexible. Otros modelos como PGA (ambos modelos), SGA-cube, SP1-GA o PARAGENESIS están fuertemente atados a la máquina para la que fueron desarrollados. En este aspecto son más implementaciones paralelas que ejemplos de modelos de ejecución paralela.

El resto de modelos se diseñó pensando en potenciar en particular una o varias características de la búsqueda. Este es el caso de GENITOR II y dGANN que utilizan una evolución de estado estacionario en cada isla en lugar de una generacional. PeGAsuS y GALOPPS están pensados para ser fácilmente transportables entre máquinas de distintos tipos. GAMAS potencia el uso de diferentes especies de codificación entre las distintas islas, aunque de forma distinta a iiGA en donde el flujo de estructuras está jerarquizado y se potencia la heterogeneidad y trabajo asíncrono sobre el resto de sus características. Finalmente, GDGA, CoPDEB y dcGA están pensados para introducir de forma explícita diferentes niveles de exploración/explotación en el algoritmo atendiendo al uso de distintos operadores o modelos de evolución.

TABLA 1.3. RESUMEN ORDENADO POR AÑO DE ALGUNAS IMPLEMENTACIONES DE AGPS

Nombre	Ref.	Año	Características Generales
PGA	[PLG87]	1987	Islas generacs. sobre Intel iPSC (8 CPUs). Migra el mejor. Topología dinámica
dGA	[Tane89]	1989	Poblaciones distribuidas. Migra el 20% de cada población cada 20 generaciones
GENITOR II	[WS90]	1990	Estado estacionario con selección por *ranking* y cruce específico propio
PGA	[MSB91]	1991	Subpoblaciones en una escalera circular 2-D. Migra el mejor y hace escalada local
SGA-cube	[ESG91]	1991	Hecho para el nCUBE2. Extensión paralela del SGA básico de Goldberg
PARAGENESIS	---	1992	Hecho para la CM-200. Sitúa un individuo en cada CPU
PeGAsuS	[RAT93]	1993	Pensado para máquinas MIMD y escrito en un lenguaje de alto nivel muy flexible
GAMAS	[PGY94]	1994	Usa 4 especies (islas) heterogéneas y migración y codificación muy especiales
iiGA	[LPG94]	1994	Islas de inyección con nodos heterogéneos jerarquizados y migración asíncrona
SP1-GA	[Levi94]	1994	128 islas de estado estacionario sobre un IBM SP1 de 128 nodos. Cuadrícula 2-D
DGENESIS	[MC94]	1994	Topología flexible por migración. Implementado usando *sockets* sobre UDP
GALOPPS	[Good96]	1996	Muy flexible. Implementado con PVM. Dispone un gran número de operadores
GDGA	[HL97]	1996	Síncrono. Paralelismo simulado (generacional). Genotipo real y cruce difuso
CoPDEB	[AP96]	1996	Cada isla usa sus propios operadores y distintas probabilidades para ellos
dGANN	[AC97]	1997	Islas de estado estacionario en anillo con intercambios síncronos de un individuo
dcGA	[CAT98a]	1998	Islas celulares o de estado estacionario con migración en anillo

Las actuales tendencias están provocando por ejemplo que GENITOR II evolucione hasta convertirse en una biblioteca incluso conteniendo modelos celulares. Como otro ejemplo de las nuevas tendencias podemos señalar a GDGA, quien utiliza un sistema de exploración/explotación explícitas en distintas islas, y que en la actualidad está evolucionando a un sistema jerárquico multinivel, todo ello ejecutado sobre un único procesador.

Aunque la lista está lejos de ser completa, demuestra el interés en realizar estos algoritmos como *software* flexible y paralelo. Igualmente se detecta una tendencia a utilizar implementaciones dependientes de una máquina en concreto, sobre todo en el caso de los modelos de difusión existentes, ya que el uso de poblaciones estructuradas en cuadrícula ha estado siempre asociado al uso de computadores SIMD de igual disposición bidimensional.

La tendencia a utilizar lenguajes modernos como C++ [Stro91] o JAVA [Pew96] es cada vez más acusada en la actualidad. La expansión del uso de modelos paralelos es importante y abarca numerosos dominios entre los que destacamos los contenidos en la Tabla 1.4.

TABLA 1.4. ALGUNOS DOMINIOS DE APLICACIÓN DE LOS ALGORITMOS GENÉTICOS PARALELOS

Referencia	Dominio de Aplicación
[AC97]	Entrenamiento en paralelo de redes de neuronas, lógica difusa,... Algoritmos puros e hibridados
[CPRS96]	Síntesis de circuitos VLSI
[GW93]	Optimización Funcional
[Levi94]	Problemas de partición de conjuntos
[LPG94]	Problemas de partición de grafos
[Mich92]	Optimización con restricciones, problemas de reordenación, ...
[MSB91]	Viajante de comercio (TSP), optimización funcional
[MT91]	Distribución de la carga de computación sobre un conjunto de procesadores
[PGY94]	Problema de asignación de ficheros, red de neuronas XOR, funciones senoidales
[Sten93]	Modelado de sistemas, estructura de las proteínas y problemas de empaquetado bidimensional
[Tane89]	Polinomios Walsh
[WS90]	Optimización de pesos en redes de neuronas (XOR, sumador binario, ...) y optimización funcional

Puede observarse en la tabla anterior la multiplicidad de trabajos en dominios de optimización funcional, optimización combinatoria, inteligencia artificial o ingeniería. Existen muchas otras aplicaciones interesantes en otros dominios que justifican el valor del trabajo hecho en algoritmos genéticos (y evolutivos en general) paralelos: asignación de frecuencias a enlaces radio, predicción de mercado, teoría de juegos, gráficos, ...

En la Tabla 1.5 detallamos el tipo de paralelismo que se ha introducido en diferentes implementaciones de algoritmos genéticos con población estructurada, así como un resumen sobre el tipo de topología que interconecta a los subalgoritmos y sus aplicaciones más conocidas.

TABLA 1.5. DETALLES INTERESANTES SOBRE ALGORITMOS GENÉTICOS PARALELOS

AGP	Tipo de Paralelismo	Topología	Ha sido Aplicado a...
ASPARAGOS	Grano fino. Aplica escalada si no detecta mejora	Escalera	TSP
CoPDEB	Grano grueso. Cada subpoblación aplica dif. operadores	Totalmente conect.	Opt. func. y RNs
DGENESIS 1.0	Grano grueso con migración entre subpoblaciones	Cualquiera deseada	Optimización funcional
ECO-GA	Grano fino (uno de los primeros de este tipo)	Malla	Optimización funcional
EnGENEer	Paralelización global (evaluaciones paralelas)	Maestro / Esclavo	Muy variadas aplics.
GALOPPS 3.1	Grano grueso. Sistema muy portable	Cualquiera deseada	Opt. func. y transporte
GAMAS	Grano grueso. Utiliza cuatro especies de genotipo	Jerarquía fija	Opt. func., RNs, ...
GAME	Orientado a objetos. Versión paralela no disponible	Cualquiera deseada	TSP, Opt. func., ...
GAucsd 1.2 / 1.4	No es paralelo, pero distribuye los experimentos sobre una LAN	<secuencial>	<como GENESIS>
GDGA	Grano grueso. Admite exploración/explotación explícita	Hipercubo	Opt. func. y (genot-FP)
GENITOR II	Grano grueso. Operador de cruce propio e interesante	Anillo	Opt. func. y RNs
HSDGA	Usa niveles de grano grueso y de grano fino para una ES	Anillo, Árbol, ...	Optimización funcional
PARAGENESIS	Paralalelización global y Grano grueso. CM-200 (1 ind→1 CPU)	Múltiple	Optimización funcional
PeGAsuS	Grano grueso o fino (MIMD). Programación de alto nivel.	Múltiple	Enseñanza y opt. func.
PGA 2.5	Selección estructur. Realiza migraciones entre subpoblaciones	Múltiple	Mochila y Opt. Func.
PGAPack	Paralelización global (evaluaciones paralelas)	Maestro / Esclavo	Optimización funcional
RPL2	Grano grueso y fino. Muy flexible y abierto a nuevos modelos	Cualquiera deseada	Investigación y Negocios
SGA-Cube	Paralelización global. Diseñado para el nCUBE 2	Hipercubo	Optimización funcional

Puede observarse en esta tabla la gran variedad de topologías usadas y la asiduidad de los algoritmos de grano grueso usando modelos distribuidos, así como el incremento reciente en la importancia de los modelos celulares, y la disminución de AGPs que usan paralelización global. Incluso es muy notable la inclusión de técnicas de búsqueda local o similares para mejorar los resultados. Esta gran diversidad de sistemas *software* es actualmente muy acusada. En general, pocos enfoques estructurados y unificados se han realizado. Sin embargo, sus características comunes permiten clasificarlos y, como veremos en breve, dar un enfoque típicamente *software* de ellos. Podemos distinguir tres categorías de algoritmos genéticos [RAT93]:

- **Orientados a la Aplicación**: Sistemas de caja negra diseñados para ocultar los detalles de los algoritmos y ayudar al usuario a desarrollar aplicaciones en dominios específicos. Normalmente presentan una interfaz dirigida por menú y son altamente flexibles.

- **Orientados al Algoritmo**: Basados en algoritmos concretos. El código fuente está disponible para incorporarlo en nuevas aplicaciones. A su vez podemos distinguir:

  - **Algoritmo Específico**: Contienen un único algoritmo genético. Sus usuarios suelen ser técnicos que generan nuevas aplicaciones o investigadores en computación evolutiva.

  - **Biblioteca de Algoritmos**: Contienen un grupo de algoritmos de forma estructurada y son muy flexibles para incorporar nuevos operadores o parámetros.

• **Herramientas**: Entornos de programación flexible para generar algoritmos o aplicaciones. Pueden subdividirse a su vez en dos tipos:

- **Educacionales**: Utilizados para explicar conceptos a usuarios noveles. Las técnicas básicas típicas para estudiar ejecuciones o resultados durante y tras la ejecución están ampliamente disponibles y son fáciles de usar.

- **Propósito General**: Útiles para desarrollar, modificar y supervisar un extenso rango de operadores, algoritmos y aplicaciones finales.

Un tipo especial de herramientas son las implementadas como *Marcos* o *Frameworks* [RP97], destinados a simplificar y extender el uso de algoritmos (u otros sistemas) paralelos entre usuarios no especializados. Cualquier modelo es susceptible de este tipo de sistema, en particular, en el Capítulo 3, proponemos un desarrollo en forma de marco inspirado y pensado para ser implementado con uno de nuestros sistemas de optimización paralelos.

Presentamos a continuación algunas tablas con detalles sobre los algoritmos concretos que más cercanos están a los propuestos y evaluados en esta memoria. Además de las etiquetas resultantes de la clasificación anterior usaremos *Prop* para identificar aplicaciones comerciales pertenecientes a una compañía propietaria de dicho algoritmo. Hemos introducido en las tablas sobre todo algoritmos paralelos distribuidos, aunque están presentes también algunos de los más relevantes entornos de celulares y de población única. Añadimos también el tipo de evolución básica distinguiendo entre generacional (*Gen*), estado estacionario (*EE*) y *ajustable*. Distinguimos además aquellos algoritmos en los que el uso intensivo de búsqueda local (*Local*) es una importante baza para el éxito del algoritmo.

En primer lugar mostramos una clasificación muy detallada *atendiendo a los algoritmos* de implementación tanto secuencial como paralela (Tabla 1.6) y explicamos los símbolos utilizados en la Tabla 1.7. La Tabla 1.8 resume el conjunto de algoritmos incluidos en la Tabla 1.6 *atendiendo a la clasificación* que acabamos de mencionar.

Como puede observarse en la Tabla 1.6, básicamente todos los modelos incluyen un único tipo de AG están implementados en C o C++ y funcionan sobre UNIX. Las bibliotecas como EM 2.1, OOGA, GAGS, GAlib, PGAPack, SUGAL 2.0, ... incluyen herramientas de programación estructurada u orientada a objetos destinadas a permitir una rápida definición de prototipos.

Las aspiraciones de GAME son realmente más altas, pues es una biblioteca más genérica y flexible que el resto. Por otro lado, TOLKIEN es un sistema bastante grande donde no sólo se consideran algoritmos y herramientas para computación evolutiva, sino también técnicas útiles en otros dominios como inteligencia artificial, aprendizaje máquina, ... En computación evolutiva RPL2 es quizás el único de los entornos paralelos que ha cumplido con las ventajas teóricas potenciales al permitir llevarlas a la práctica, proporcionando un lenguaje de definición de modelos para algoritmos evolutivos.

Algunos de los sistemas como ASPARAGOS se han remozado e incluso cambiado su orientación de grano fino por otra de grano grueso que permite un mayor impacto en los resultados obtenidos. Otros modelos como CoPDEB o GDGA son bastante nuevos y con peculiaridades muy propias, de corte muy distinto a otros poco conocidos como OMEGA o Splicer 1.0 que tienen una aplicación bastante restringida y poco estándar.

TABLA 1.6. RESUMEN DE ALGORITMOS GENÉTICOS SECUENCIALES Y PARALELOS

Nombre	Ref.	Leng.	SO/Máquina	Tipo de GA	Algoritmo	Sec/Par	Evolución	ACCESIBLE EN...
ASPARAGOS	[Gorg89]	C	64 T800 transp	Algor. Específico	Único / No pop.	PAR	Local	---
CoPDEB	[AP96]	C	UNIX	Algor. Específico	Único / No prop.	PAR	Gen.	mail to: adamidis@it.teithe.gr
DGENESIS 1.0	[MC94]	C	UNIX	Algor. Específico	Único / No Prop.	PAR	Ajustable	ftp to: ftp.aic.nrl.navy.mil
ECO-GA	[Davi91b]	C?	UNIX?	Algor. Específico	Único / No Prop.	PAR	Local	mail to: yuval@wisdom.weizmann.ac.il
EM 2.1	[VBT91]	C	DOS	Biblioteca	Biblioteca	SEC	Gen. / EE/ ES	ftp to: [130.149.192.50]/
EnGENEer	[Robb92]	C	UNIX	Propósito General	Biblioteca	SEC/par	Gen. / EE	Tel: +44 1716 379111, George Robbins
$ESC^{A}P_{A}DE$ 1.2	[Hoff91]	C	---	Biblioteca	Biblioteca	---	ES	ftp to: ls11.informatik.uni-dortmund.de
EVOLVER 2.0	---	C++	DOS & MAC	Application Orient.	Único / Prop.	SEC	---	http://www.palisade.com
GA Workbench	[Hugh89]	C	DOS & WIN	Educacional	Biblioteca	SEC	Ajustable	ftp to: camcon.co.uk
GAGA	[AT91]	C	UNIX, DOS, MAC	Algor. Específico	Único / No Prop.	SEC	como GAs	ftp to: cs.ucl.ac.uk/darpa/gaga.shar
GAGS	[MP96]	C++	UNIX & DOS	Propósito General	Biblioteca	SEC	Gen.	http://kal-el.ugr.es/GAGS
GAlib	[Wall95]	C++	UNIX	Biblioteca	Biblioteca	SEC	Gen.	ftp to: lancet.mit.edu/pub/ga/galib242.tar.gz
GALOPPS 3.1	[Good96]	C	UNIX, DOS, MAC	Algor. Específico	Único / No Prop.	PAR	Gen.	ftp to: GARAGe.cps.msu.edu
GAMAS	[PGY94]	C?	UNIX	Algor. Específico	Único / No Prop	PAR	Gen.	---
GAME	[Sten93]	C++	UNIX & DOS	Propósito General	Biblioteca	SEC-PAR	Gen. / EE	ftp to: bells.cs.ucl.ac.uk/papagena/game
GAucsd 1.2 / 1.4	[SG91]	C	UNIX, DOS ...	Algor. Específico	Único / No Prop.	SEC	Gen.	http://www.aic.nrl.navy.mil/galist/src/GAucsd14.sh.Z
GDGA	[HL97]	C	UNIX	Algor. Específico	Único / No Prop.	SEC/par	Gen.	mail to: lozano@decsai.ugr.es
GENESIS 5.0	[Gref84]	C	UNIX, DOS ...	Algor. Específico	Único / No Prop.	SEC	Ajustable	http://www.aic.nrl.navy.mil/galist/src/genesis.tar.Z
GENEsYs 1.0	---	C	UNIX	Algor. Específico	Único / No Prop.	SEC	Ajustable	[129.217.36.140]/pub/GA/src/GENEsYs-1.0.tar.Z
GENITOR I - II	[WS90]	C	UNIX	Biblioteca	Biblioteca	SEC-PAR	EE / *Rank*	*/genitor
HSDGA	[VSB92]	C	HELIOS	Algor. Específico	Único / No Prop.	PAR	Local	---
LibGA	[CW93]	C	UNIX, DOS, NeXT	Algor. Específico	Biblioteca	SEC	Gen. / EE	*/libga
OMEGA	[Sten93]	---	---	Application Orient.	Prop.	SEC	---	Tel: +44 1371 870254, KiQ Limited (business)
OOGA	[DG91]	CLOS	cualquier LISP	Biblioteca	Biblioteca	SEC	Gen. / EE	TSP, P.O. Box 991, Melrose, MA 02176-USA
PARAGENESIS	---	C*	CM-200	Algor. Específico	Único / No Prop	PAR	Ajustable	[192.26.18.74]/pub/galist/src/ga/paragenesis.tar.Z
PC/BEAGLE	[RAT93]	---	DOS	Orient. Aplicación	Prop.	SEC	EE	Tel: +44 117 942 8692, Pathway Research Ltd.
PeGAsuS	[RAT93]	C	UNIX, ...	Propósito General	Prop.	PAR	Gen. / EE	mail to: dirk.schlierkamp-voosen@gmd.de
PGA 2.5	---	C	UNIX & DOS	Biblioteca	Único / No Prop	PAR	Gen. / Local	http://www.aic.nrl.navy.mil/galist/src /pga-2.5.tar.Z
PGAPack	[Levi96]	Fort., C	UNIX, ...	Biblioteca	Biblioteca	PAR	Gen. / EE	http://www.mcs.anl.gov/pgapack.html
RPL2	[RS94]	C	UNIX, ...	Propósito General	Biblioteca Prop.	PAR	Gen. / EE	http://www.quadstone.co.uk/~rpl2
SGA	[Gold89a]	Pascal	UNIX	Algor. Específico	Único / No Prop	SEC	Gen.	ftp to: [192.26.18.56] ... sgac_94m.tgz (in C)
SGA-Cube	[ESG91]	C	nCUBE 2	Algor. Específico	Único / No Prop	PAR	Gen.	ftp to: [192.26.18.56] ... sgacub94.tgz
Splicer 1.0	[NASA91]	C	UNIX & MAC	Propósito General	Biblioteca	SEC	Gen.	ftp to: galileo.jsc.nasa.gov
SUGAL 2.0	[Hunt95]	C	UNIX, DOS, ...	Biblioteca	Biblioteca	SEC	Gen. / EE	http://osiris.sund.ac.uk/ahu/sugal/home.html
TOLKIEN	---	C++	UNIX, DOS, WIN	Propósito General	Biblioteca	SEC	Gen.?	*/tolkien
XpertRule GenAsys	---	---	---	Orient. Aplicación	Prop.	SEC	---	http://www.attar.com, Attar Software (scheduling)

Existen pocos modelos educacionales como GA Workbench (quizás SGA pudiera ser considerado también educacional por su historia), ya que normalmente no existen referencias a ellos debido a su uso local en las instituciones que los desarrollan. Algunos modelos como HSDGA, SGA-Cube o PGA 2.5 de altas prestaciones han sido casi abandonados por sus desarrolladores debido a que estaban muy asociados a sistemas operativos y máquinas que van desapareciendo del mercado en favor casi siempre de una red local de estaciones de trabajo.

Algunos de los algoritmos mencionados permiten elegir si utilizar una evolución generacional o de estado estacionario, con contadas menciones al uso de un modelo ajustable (por ejemplo DGENESIS y PARAGENESIS). En ninguno encontramos explícitamente una distribución de islas celulares aunque *potencialmente* podrían conseguirse con GAME y existe un ejemplo de aplicación usando RPL2.

En el terreno secuencial, algunos sistemas de AEs han sentado escuela, como el popular GENESIS, con numerosas extensiones como GAucsd, GENEsYs o DGENESIS. Básicamente son sistemas de algoritmo específico de libre disposición para usar en comparaciones o aplicaciones de investigadores de otros campos.

Junto a GENESIS, otro estándar importante en el terreno de los algoritmos de estado estacionario es GENITOR y su versión distribuida en islas GENITOR II. Este último, junto con dGA quizás sean los estándares más claros de evolución distribuida, aunque en la actualidad el primero está incorporando nuevos modelos y variantes. Respecto a modelos de difusión, pocos se conocen con un nombre y son un sistema *software*, con la notable excepción de ECO-GA.

Los modelos GAGA y GAMAS representan modelos poco usuales porque utilizan representaciones distintas en los subalgoritmos componentes. GALOPPS cuida más los aspectos algorítmicos y de implementación. De igual manera, LibGA es un sistema relativamente nuevo que presta atención a la definición de componentes y a la versatilidad y facilidad de extensión.

El resto de los modelos incluidos en la Tabla 1.6 se mencionan por completitud (sistemas no muy conocidos como EnGENEer, $ESC^{A}P_{A}DE$ 1.2 o PeGAsuS) o para hacer constar que existen aplicaciones comerciales y algoritmos no disponibles para el público que realmente dan beneficios en el mercado (EVOLVER 2.0, OMEGA, PC/BEAGLE o XpertRule GenAsys).

En la tabla anterior hemos incluido los enlaces hipertexto a los lugares más conocidos donde encontrar estos sistemas. Estos enlaces son bastante estables en su mayoría ya que funcionan desde hace varios años y no existen previsiones de que cambien; más bien al contrario, se refuerzan con la adición de nueva documentación, versiones, y *software* auxiliar.

La Tabla 1.7 recuerda el significado de las etiquetas para acceder a la Tabla 1.6 de forma más simple y rápida. Algunos modelos pueden trabajar indistintamente en modo secuencial o paralelo (SEC-PAR) aunque algunos de ellos no tienen aún versiones paralelas (los notamos como *par*) o están en fase de desarrollo.

Como mencionamos antes, en la Tabla 1.8 presentamos una clasificación de los modelos de la Tabla 1.6 atendiendo a los tipos de algoritmos definidos en esta misma sección. Puede observarse cómo los sistemas de algoritmo específico son los más abundantes, mientras que los puramente educacionales son minoría.

TABLA 1.7. SÍMBOLOS UTILIZADOS EN LA TABLA ANTERIOR

Símbolo	Significado
Local	Las operaciones sobre cada cadena involucran como mucho a las situadas en su vecindario
Gen.	Generacional: El paso algorítmico básico es la creación de toda una generación de individuos
EE	Estado estacionario (*Steady-State*): el paso básico de la evolución es la creación de unos pocos individuos (normalmente sólo uno)
Ajustable	(*gap*) El algoritmo reemplaza en cada paso sólo a un cierto porcentaje de los individuos de una población
E. S.	*Evolutionary Strategy*
Rank	Se ordena a los individuos por valor creciente de adecuación y la selección se efectúa atendiendo a su posición (y no a su adecuación)
par	(*minúscula*) Versión paralela aún no disponible o no es una característica fundamental de la búsqueda
SEC-PAR	El algoritmo es capaz de trabajar tanto en secuencial como en paralelo (y esto se aprovecha como característica sobresaliente)
*	`http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems`
	`pub/software/Evolution_Machine/em_tc.exe`

TABLA 1.8. OTRA PERSPECTIVA DE LA CLASIFICACIÓN ATENDIENDO AL TIPO DE ALGORITMO

Orientados a la Aplicación	Orientados al Algoritmo		Herramientas (*Tool Kits*)	
	Algoritmo Específico	**Biblioteca de Algoritmos**	**Educacionales**	**Propósito General**
EVOLVER 2.0	ASPARAGOS	EM 2.1	GA Workbench	EnGENEer
OMEGA	CoPDEB	SUGAL 2.0		GAGS
PC/BEAGLE	DGENESIS 1.0	$ESC^{A}P_{A}DE$ 1.2		GAME
XpertRule GenAsys	ECO-GA	GAlib		PeGAsuS
	GAGA	GENITOR I - II		RPL2
	GALOPPS 3.1	libGA		Splicer 1.0
	GAMAS	OOGA		TOLKIEN
	GAucsd 1.2 / 1.4	PGA 2.5		
	GDGA	PGAPack		
	GENESIS 5.0 - GENEsYs 1.0			
	HSDGA			
	PARAGENESIS			
	SGA			
	SGA-Cube			

1.2. Establecimiento del Problema

De una forma genérica, un algoritmo evolutivo resuelve problemas de optimización global (Definición 1.1). En este tipo de problemas el objetivo es encontrar un conjunto de parámetros que maximicen un cierto criterio de calidad. Los problemas de este tipo tienen una enorme importancia en muchos campos de investigación tales como producción industrial, diseño asistido por ordenador, construcción, biología, química, sistemas eléctricos, medicina, ...

El objetivo de un *problema de optimización global* se resume en la siguiente definición de *máximo global*.

Definición 1.8. (Máximo Global). *Dada una función* $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $M \neq \emptyset$, *para* $\vec{x}^* \in M$ *el valor* $f^* := f(\vec{x}^*) < +\infty$ *se denomina* máximo global *sii*

$$\forall \vec{x} \in M : \quad f(\vec{x}^*) \geq f(\vec{x}) \ , \tag{1.11}$$

donde $\vec{x}^*$ *es un* máximo global*, f es la* función objetivo, *y el conjunto M es la* región de soluciones admisibles*. El problema de determinar un máximo global se denomina* problema de optimización global *(Definición* 1.1*).* ■

Se ha demostrado [Kurs92] que los algoritmos evolutivos pueden extenderse para resolver problemas de toma de decisión multi-criterio, en los que la función objetivo tiene la forma más general:

$$f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^k, \quad M \neq \emptyset, \quad k>1 \ . \tag{1.12}$$

En este tipo de problemas existe un conjunto de soluciones no-dominantes (Pareto). En dichos problemas la calidad de la solución puede mejorarse respecto de uno de los criterios sólo degradando al menos uno de los restantes criterios. Nuestro trabajo se centra en resolver problemas del caso de criterio único, es decir, k=1.

Asimismo, nos concentraremos en problemas de maximización para proporcionar una visión única estándar en la descripción de técnicas y experimentos. Esto no restringe en absoluto la generalidad de los resultados ya que puede establecerse una igualdad entre tipos de problemas de maximización y minimización [Gold89a] [Bäck96]:

$$\max\left\{f(\vec{x}) \mid \vec{x} \in M\right\} \ = \ -\min\left\{-f(\vec{x}) \mid \vec{x} \in M\right\} . \tag{1.13}$$

El problema básico de optimización global se ve a menudo dificultado por la existencia de restricciones, es decir, por una especial definición de la región de soluciones admisibles M.

Definición 1.9. (Restricciones). *Sea* $M = \left\{\vec{x} \in IR^n \mid g_i(\vec{x}) \geq 0 \quad \forall i \in [1,...q]\right\}$ *la región de soluciones admisibles de la función objetivo* $f : M \rightarrow \mathbb{R}$. *Las funciones* $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ *se denominan* restricciones*. Se denomina a* g_j *de distinta forma según el valor que toma sobre* $\vec{x} \in \mathbb{R}^n$:

$$\begin{aligned} &\text{satisfecha} && :\Leftrightarrow \quad g_j(\vec{x}) \geq 0 \ , \\ &\text{activa} && :\Leftrightarrow \quad g_j(\vec{x}) = 0 \ , \\ &\text{inactiva} && :\Leftrightarrow \quad g_j(\vec{x}) > 0 \ , \text{ y} \\ &\text{violada} && :\Leftrightarrow \quad g_j(\vec{x}) < 0 \ . \end{aligned} \tag{1.14}$$

El problema de optimización global se denomina sin restricciones *sii* $M = \mathbb{R}^n$*; en otro caso se le denomina* restringido *o* con restricciones*.* ■

Existen trabajos muy interesantes sobre cómo manejar restricciones en un algoritmo evolutivo. Vea por el ejemplo [MJ91] o [Mich92]. En resumen, el problema se ve dificultado por el hecho de que el conjunto de soluciones admisibles $F \subseteq M$ es en realidad un subconjunto del dominio de las variables (aquellos puntos que satisfacen todas las restricciones):

$$F = \{\vec{x} \in M \mid g_j(\vec{x}) \geq 0 \quad \forall j\} \ . \tag{1.15}$$

En general, en un problema complejo como los que utilizamos en este trabajo, la función objetivo presenta varios máximos de distintas alturas. La mayoría de los métodos de búsqueda suelen quedar atrapados en máximos locales una vez que caen en su región de atracción[1]. Para formalizar el concepto de *máximo local* es necesario primero especificar una *métrica* o medida de distancia de un espacio vectorial arbitrario.

Definición 1.10. (Métrica). *Sea V un espacio vectorial. Una* métrica *sobre V es una aplicación* $\rho : V^2 \rightarrow IR_0^+$, *tal que* $\forall \vec{v}, \vec{w}, \vec{x} \in V$ *:*

$$\begin{aligned} \rho(\vec{v}, \vec{w}) &= 0 \quad \Leftrightarrow \quad \vec{v} = \vec{w} \\ \rho(\vec{v}, \vec{w}) &= \rho(\vec{w}, \vec{v}) \\ \rho(\vec{v}, \vec{w}) &\leq \rho(\vec{v}, \vec{x}) + \rho(\vec{x}, \vec{w}) \ . \end{aligned} \tag{1.16}$$

■

En muchos casos es suficiente con definir una *norma* sobre el espacio de vectores reales (o incluso complejos) en lugar de una métrica. Informalmente podemos pensar en la norma de un vector como una medida de su longitud.

Definición 1.11. (Norma). *Sea V un espacio vectorial sobre* $\mathbb{R}$. *Una* norma *sobre V es una aplicación* $\| \cdot \| : V \rightarrow IR_0^+$, *tal que* $\forall \vec{v}, \vec{w}, \vec{x} \in V$, $r \in \mathbb{R}$*:*

$$\begin{aligned} \|\vec{v}\| &= 0 \quad \Leftrightarrow \quad \vec{v} = \vec{0} \\ \|r \cdot \vec{v}\| &= r \cdot \|\vec{v}\| \\ \|\vec{v} + \vec{w}\| &\leq \|\vec{v}\| + \|\vec{w}\| \ . \end{aligned} \tag{1.17}$$

■

Es trivial verificar que en cualquier espacio donde pueda definirse una norma puede también definirse una métrica, únicamente considerando que la siguiente igualdad siempre se cumple:

$$\rho(\vec{v}, \vec{w}) = \|\vec{v} - \vec{w}\| \ . \tag{1.18}$$

Por supuesto la condición inversa no se cumple. Es muy normal utilizar la *norma euclídea* cuando se trabaja con topologías en $\mathbb{R}^n$. Esta norma asegura que $\forall \vec{x} = (x_1, ..., x_n) \in \mathbb{R}^n$ ocurre que:

$$\|\vec{x}\| = \sqrt{\sum_{i=1}^{n} |x_i|^2} \ . \tag{1.19}$$

[1] Por región de atracción queremos referirnos informalmente al conjunto de todos los puntos desde los cuales una secuencia monótona de pasos de subida conducen al punto máximo (local en este caso).

En este caso, la métrica $\|\vec{x}-\vec{y}\|$ proporciona la definición estándar de *distancia euclídea*. Basándonos en estas definiciones podemos ya caracterizar los máximos locales como sigue:

Definición 1.12. (Máximo Local). *Para un cierto* $\hat{\vec{x}} \in M$ *el valor* $\hat{f} := f(\hat{\vec{x}})$ *es un* máximo local *sii*

$$\exists \varepsilon \in IR, \varepsilon > 0 : \forall \vec{x} \in M : \ \left\|\vec{x}-\hat{\vec{x}}\right\| < \varepsilon \Rightarrow \hat{f} \geq f(\vec{x}) \ . \tag{1.20}$$

■

En definitiva, queremos decir que existe un entorno-ε $U_\varepsilon(\hat{\vec{x}}) = \left\{\vec{x} \in M \mid \left\|\vec{x}-\hat{\vec{x}}\right\| < \varepsilon\right\}$ tal que $\hat{f}$ es el mayor valor admisible de la función objetivo dentro de dicho entorno. Obviamente cualquier máximo global es también un máximo local. Se dice que la función objetivo es *unimodal* si tiene exactamente un máximo local. En otro caso se denomina *multimodal*.

En nuestro trabajo estamos interesados en obtener un óptimo global, aunque a veces este objetivo debe relajarse debido a la dificultad del problema tratado. De hecho no existe un criterio genérico de identificar un máximo global [TZ89]. Además, en la práctica, tanto la discretización que introduce el trabajo numérico con un ordenador, como la demanda relativamente permisiva en relación a su nivel de precisión hacen que esto no sea un problema.

Los planteamientos hechos sobre optimización global permiten un paso suave del dominio de las variables continuas al de las variables discretas. La idea clave es pensar que estamos discretizando el espacio de búsqueda en puntos representables en el ordenador situados en una rejilla de puntos esquidistantes en los que consideramos situados cualquier punto inspeccionado por el algoritmo. La resolución de dicha rejilla (número de marcas en cada dimensión) depende del problema y del ordenador usado. En general, es admisible pensar que encontrar un óptimo en esta rejilla abstracta que impone la discretización de valores continuos es equivalente a encontrar el óptimo de la función objetivo real.

Los problemas de optimización en los que se trabaja con variables objetivo discretas se conocen como *problemas de optimización combinatoria*. Aunque el concepto de óptimo global es el mismo, en estos problemas la definición del entorno-ε sí que se transforma en un concepto equivalente pero nuevo. Este concepto es el de *vecindario*. Con ambos podemos caracterizar conjuntos de puntos cercanos a un punto dado atendiendo a la métrica usada, pero el concepto de vecindario es más general que el de entorno-ε. Según las definiciones de Papadimitriu y Steiglitz [PS82] podemos hacer las siguientes definiciones aquí:

Definición 1.13. (Vecindario). *Se define un* vecindario *sobre un conjunto S como una aplicación* $\mathcal{N} : S \to 2^S$*, donde* 2^S *denota el* conjunto de las partes (*power set*) *de S -*P(S)-. ■

Ahora podemos redefinir el concepto de máximo local en un espacio discreto.

Definición 1.14. (Máximo Local en Espacios Discretos). *Para un cierto* $\hat{\vec{x}} \in M$ *, donde M≠∅ denota una región admisible arbitraria, y un cierto vecindario* $\mathcal{N} : M \to 2^M$*, el valor* $\hat{f} := f(\hat{\vec{x}})$ *es un* máximo local respecto a $\mathcal{N}$*sii*

$$\hat{f} \geq f(\vec{x}) \quad \forall \vec{x} \in \mathcal{N}(\hat{\vec{x}}) \ . \tag{1.21}$$

■

En esta memoria también abordamos la solución de problemas de una subclase de la optimización combinatoria bastante importante denominada *problemas de optimización pseudológicos* (*pseudoboolean*). En este tipo de problemas la función objetivo transforma vectores de parámetros binarios en valores reales. Si definimos $\mathbb{B}=\{0,1\}$ como el dominio de dichas variables, entonces $f: \mathbb{B}^n \rightarrow \mathbb{R}$ tiene 2^n vectores-argumento posibles. Esto implica también que es imposible una enumeración completa de todas las soluciones debido al crecimiento exponencial con n y a las restricciones naturales de los recursos de computación.

En estos casos se suele utilizar como métrica la conocida *distancia de Hamming*, que simplemente cuenta el número de posiciones en que dos vectores se diferencian.

Definición 1.15. (Distancia de Hamming). *Para dos vectores* $\vec{a} = (a_1,...,a_n) \in \mathbb{B}^n$ *y* $\vec{b} = (b_1,...,b_n) \in \mathbb{B}^n$ *se define la métrica llamada* distancia de Hamming *como*

$$\rho_H\left(\vec{a},\vec{b}\right) \;=\; \sum_{i=1}^{n} \left|a_i - b_i\right| \;. \tag{1.22}$$

■

Con esta métrica podemos definir vecindarios $\mathcal{N}_k$ para cualquier $k \in \{0,...,n\}$ por agrupación de todos los puntos de igual distancia a un punto de referencia común:

$$\mathcal{N}_k(\vec{a}) \;=\; \{\vec{b} \in IB^n \;\mid\; \rho_H\left(\vec{a},\vec{b}\right) = k\} \;. \tag{1.23}$$

Los óptimos locales del hipercubo binario vienen caracterizados de acuerdo a su vecindario $\mathcal{N}_1$ de tal forma que, dada una función $f: IB^n \rightarrow IR$, $\hat{f} := f(\hat{\vec{a}})$ es un máximo local (respecto $\mathcal{N}_1$) de f, sii $\forall \vec{b} \in \mathcal{N}_1(\vec{a}):\;\; \hat{f} \geq f(\vec{b})$.

Una vez descrito el problema y los algoritmos usados para resolverlo, en la siguiente sección caracterizamos los operadores formalmente y de esta manera, además, concretamos aquéllos en los que estamos interesados en nuestro estudio.

1.3. Técnicas Básicas e Implementación

En esta sección se presentan las técnicas básicas que usaremos en los modelos estudiados (Sección 1.3.1). Asimismo, discutimos la necesidad de implementar estas técnicas y, en general, el *software* evolutivo, atendiendo a paradigmas de programación más modernos como es el de la programación orientada a objetos (Sección 1.3.2).

1.3.1. Codificación, Operadores y Criterios de Evaluación

De entre todos los algoritmos evolutivos presentamos un estudio esencialmente en el campo de los algoritmos genéticos. Para caracterizar un algoritmo genético secuencial frente a otro de los algoritmos de su familia presentamos la instanciación de operadores, codificaciones y técnicas que concretan la descripción de un AE secuencial (Algoritmo 1.3) para dar lugar al tipo de algoritmos con los que trabajamos.

Para facilitar la descripción presentaremos primero los genotipos que consideramos y después el operador de selección y reemplazo en su versiones más populares utilizando panmixia. El operador de mutación que utilizaremos es unario y los operadores de cruce estudiados son binarios. Utilizaremos codificaciones en genotipo binario y real explorando así las capacidades de la caracterización ortodoxa de algoritmo genético. Nuestro interés reside en versiones canónicas de las técnicas secuenciales y paralelas, aunque la extensión del estudio que presentamos a programación genética [ACT96] de codificaciones alternativas, como por ejemplo de longitud variable [AC97], son también puntos de interés que motivan la necesidad de trabajar en paralelo y que discutiremos en el siguiente capítulo.

1.3.1.1. El Genotipo $\mathcal{G}$

En contraste con las estrategias evolutivas y la programación evolutiva, los algoritmos genéticos como los que utilizaremos trabajan sobre cadenas de longitud fija l, es decir, $\mathcal{G}=\mathbb{B}^l$. En el caso de que la función objetivo sea pseudológica esta representación es directamente utilizable como fenotipo. Sin embargo, en problemas de optimización con parámetros continuos:

$$f : \prod_{i=1}^{n} [u_i, v_i] \rightarrow \mathbb{R} \ , \tag{1.24}$$

donde $u_i<v_i$. De esta forma el genotipo que codifica un vector $\Gamma^{-1}(a_1,...,a_n) = \vec{x} \in \prod_{i=1}^{n} [u_i, v_i]$ será:

binario estándar: $$\Gamma_i^{-1}(a_{i1},...,a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1} \cdot \left(\sum_{j=0}^{l_x-1} a_{i(l_x-j)} \cdot 2^j \right) \tag{1.25}$$

binario Gray: $$\Gamma_i^{-1}(a_{i1},...,a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1} \cdot \left(\sum_{j=0}^{l_x-1} \left(\bigoplus_{k=1}^{l_x-j} a_{ik} \right) \cdot 2^j \right) \tag{1.26}$$

flotante (o real): $$\Gamma_i^{-1}(a_i) = a_i \tag{1.27}$$

En el caso binario (estándar o Gray, donde $\oplus$ indica la suma módulo 2) la cadena se subdivide de forma lógica en n trozos de igual longitud $l=n \cdot l_x$ de forma que el i-ésimo segmento $a_{i1}, ..., a_{il_x} \in \mathbb{B}^{lx}$. De esta forma la representación de los parámetros puede cambiar el problema más o menos dependiendo de la codificación usada. En definitiva, una codificación binaria presenta una precisión en su representación que depende del número de bits y de la amplitud del rango de la variable parámetro:

$$\Delta x_i \ = \ \frac{v_i - u_i}{2^{l_x} - 1} \ . \tag{1.28}$$

Así, definiendo $\Gamma^{-1} = \Gamma_1^{-1} \times \Gamma_2^{-1} \times ... \times \Gamma_n^{-1}$ tenemos la mencionada *función de decodificación* $\vec{x} = \Gamma^{-1}(\vec{a})$. El caso flotante es el de más fácil decodificación, ya que el genotipo contiene directamente las variables-parámetro. El caso binario Gray tiene la propiedad añadida sobre el binario estándar de que cada par de valores adyacentes tiene distancia de Hamming 1:

$$\forall i_1, i_2 \in \{0,...,2^{l_x} - 1\} \qquad |i_1 - i_2| = 1 \qquad \Rightarrow \qquad \rho_H(\vec{a}_1, \vec{a}_2) = 1 \ . \tag{1.29}$$

Esta relación es una implicación, no una equivalencia, ya que incluso con codificación Gray el cambio de un solo bit puede causar cambios arbitrariamente grandes en el valor decimal. En esta memoria utilizaremos varios de los códigos mencionados para demostrar así la validez de los modelos propuestos sobre otros desarrollos previos en términos de codificación. La codificación que se debe utilizar depende del tipo de problema. Aunque la representación binaria estándar o Gray proporciona buenos resultados [CS88] [Salo96], existen numerosos trabajos que estudian códigos no binarios [WS90] [JM91] [AC97] [Loza96], después que se demostrara su capacidad teórica no inferior a la binaria para solucionar problemas [Anto89]. Volveremos sobre este punto en el Capítulo 2.

1.3.1.2. El Operador de Selección *s*

La selección juega un importante papel tanto en los algoritmos evolutivos secuenciales como en los paralelos, ya que guía la búsqueda y provoca la convergencia de la población de individuos. Por tanto, representa el tan importante compromiso entre exploración y explotación. Vamos a ofrecer en esta sección un resumen de los operadores de selección más comunes. En la Tabla 1.9 μ es el número de cadenas en la población y $(\vec{a}_i(t))$ es una cadena del paso de evolución (generación) *t*. Para ello describimos para cada operador la probabilidad de seleccionar a un individuo cualquiera de entre los presentes en la población. En la Tabla 1.9 resumimos los mecanismos y comentamos cada uno de ellos.

De entre los operadores de selección proporcional a la adecuación, la implementación como ruleta (RW [Gold89a]) es la más popular. Sin embargo, también es de mayor complejidad algorítmica y con tendencia a cometer errores estocásticos respecto de la implementación conocida como "muestreo estocástico universal" (SUS [Bake87]). En esta memoria compararemos ambos confirmando en sus extensiones paralelas que dichas características se mantienen. De hecho, la selección proporcional a la adecuación puede usarse para seleccionar un conjunto arbitrario de parejas y por tanto puede utilizarse en modelos secuenciales generacionales o de estado estacionario, así como en modelos paralelos distribuidos o celulares.

TABLA 1.9. ESQUEMAS DE SELECCIÓN

Operador de Selección	Probabilidad de Selección	Comentarios
Proporcional a la adecuación	$p_s(\vec{a}_i(t)) = \Phi(\vec{a}_i(t)) \Big/ \sum_{j=1}^{\mu} \Phi(\vec{a}_j(t))$	[Holl75] Implementado como ruleta (RW) o como muestreo universal estocástico (SUS) [Bake87]
Ranking Lineal	$p_s(\vec{a}_i(t)) = \frac{1}{\mu} \cdot \left(\eta_{máx} - (\eta_{máx} - \eta_{mín}) \cdot \frac{i-1}{\mu - 1} \right)$	[Bake87] $\eta_{mín} = 2 - \eta_{máx}$ y $1 \le \eta_{máx} \le 2$. Individuos ordenados (adecuación) desde 1 a μ
Ranking Lineal de Whitley	$Índice(\chi, b) = \frac{\mu}{2 \cdot (b-1)} \cdot \left(b - \sqrt{b^2 - 4 \cdot (b-1) \cdot \chi} \right)$	[Whit89] Devuelve el índice del individuo. $b \in [1,2]$ es la presión y $\chi \in [0,1]$ es una VAD-U
Ranking uniforme (λ μ)	$p_s(\vec{a}_i(t)) = \begin{cases} 1/\lambda & 1 \le i \le \lambda \\ 0 & \lambda < i \le \mu \end{cases}$	[Schw81] Típico en estrategias de evolución. Los individuos están ordenados desde 1 hasta μ

En la actualidad cualquiera de los modelos listados en esta tabla se utilizan en estudios teóricos y aplicaciones prácticas. Véase una comparativa teórica detallada en [GD91].

La selección proporcional usa el valor de adecuación para seleccionar parejas o bien el individuo que se desea reemplazar. Algunos mecanismos utilizan una ordenación creciente atendiendo a la adecuación de los individuos para después seleccionar atendiendo a su posición en dicha ordenación. Esto reduce el error estocástico debido a valores de adecuación similares (aunque introduce la sobrecarga de la ordenación).

Existen muchos otros mecanismos de selección interesantes como el torneo [GD91], y sobre todo variantes de estos modelos básicos que modifican de alguna forma la probabilidad (véase la Tabla 1.10). En cualquier caso, la suma de probabilidades siempre es 1. Algunos modelos son bastante flexibles, presentando parámetros que permiten modificar su presión selectiva (preferencia por las mejores soluciones).

TABLA 1.10. POSIBLES VERSIONES DE LOS MODELOS DE SELECCIÓN

Según la respuesta a...	**Versión 1 (SÍ)**	**Versión 2 (NO)**
¿Cambian las probabilidades a través de las generaciones?	*Dinámico* $\Leftrightarrow \exists i \in \{1,...,\mu\}, \forall t \geq 0 \mid \quad p_s(\vec{a}_i(t)) = c_i$	*Estático* $\Leftrightarrow \forall i \in \{1,...,\mu\}, \forall t \geq 0 \mid \quad p_s(\vec{a}_i(t)) = c_i$
¿Se permiten probabilidades nulas?	*Extintivo* $\Leftrightarrow \forall t \geq 0, \exists i \in \{1,...,\mu\} \mid \quad p_s(\vec{a}_i(t)) = 0$	*Preservativo* $\Leftrightarrow \forall i \in \{1,...,\mu\}, \forall t \geq 0 \mid \quad p_s(\vec{a}_i(t)) > 0$
¿Los peores individuos se reproducen?	*Extintivo por la izquierda* $\Leftrightarrow \forall t \geq 0, \exists l \in \{1,...,\mu - 1\}$ $\mid \quad i \leq l \Rightarrow p_s(\vec{a}_i(t)) = 0$	*Extintivo por la derecha* $\Leftrightarrow \forall t \geq 0, \exists l \in \{2,...,\mu\}$ $\mid \quad i \geq l \Rightarrow p_s(\vec{a}_i(t)) = 0$
¿Se mezclan los k mejores padres con su descendencia para selec.?	*k-Elitista* $\Leftrightarrow \exists k \in \{1,...,\mu\}, \forall t \geq 0, \forall i \in \{1,...,k\}$ $\mid \quad f(\vec{a}_i(t)) \geq f(\vec{a}_i(t-1))$	*Puro* <Ningún individuo satisface la condición de *k*-elitismo>
¿Se genera una población de μ individuos?	*Generacional* <Conjunto fijo de padres hasta generar μ individuos, (μ,λ)>	*Estado-Estacionario* <Caso especial de $(\mu$-1)-elitismo también conocido como $(\mu$+1)>

De entre estas versiones estudiamos en este trabajo el modelo 1-Elitista y las versiones generacional y de estado estacionario.

Por otro lado, ya hemos discutido la existencia del otro tipo de selección conocido como ambiental o política de reemplazo, encargada de generar el nuevo conjunto de estructuras sobre las que el algoritmo continuará trabajando.

A continuación presentamos las variantes más populares de este tipo de técnicas de reemplazo. En general, una variante plausible a priori sería utilizar la función complementaria de cualquiera de los métodos de selección de pareja descritos. Sin embargo, existen muchas otras (Tabla 1.11).

Puesto que se trata de un mecanismo de selección, también admite variantes similares a las de la Tabla 1.10. En el presente trabajo vamos a utilizar o estudiar todos los tipos de reemplazo excepto el primero, es decir, aleatorio, siempre, si_mejor y generacional, aunque esto no signifique que todos ellos proporcionen buenos resultados [CAT98a].

TABLA 1.11. DIFERENTES POLÍTICAS DE REEMPLAZO

Reemplazo	Probabilidad de Reemplazo	Comentarios
Proporcional Inversa	$p_r(\vec{a}_i(t)) = 1 - \Phi(\vec{a}_i(t)) \Big/ \sum_{j=1}^{\mu} \Phi(\vec{a}_j(t))$	Inversa de la selección proporcional
Aleatoria Uniforme	$p_r(\vec{a}_i(t)) = c = \frac{1}{\mu}$	Cualquier individuo tiene la misma probabilidad de ser reemplazado
Peor Siempre	$p_r(\vec{a}_i(t)) = \begin{cases} 1 & i = \mu \\ 0 & i \neq \mu \end{cases}$	Reemplazar siempre al peor con el nuevo individuo en una población ordenada de mayor a menor adecuación (la notaremos como `always`)
Peor Si mejor	$p_r(\vec{a}_i(t)) = \begin{cases} 1 & i = \mu \wedge \Phi(\vec{a}_i(t)) \leq \Phi(\vec{a}'(t)) \\ 0 & i \neq \mu \end{cases}$	Reemplazar al peor con el nuevo siempre que su adecuación sea mejor. La población está ordenada de mayor a menor adecuación (`if_better`)
Generacional	$p_r(\vec{a}_i(t)) = 1$	La nueva generación reemplaza a la antigua totalmente

1.3.1.3. El Operador de Cruce ⊗

El operador de cruce es muy importante en el campo de los algoritmos genéticos. Existen numerosas variantes de él, todas ellas clasificables en tres categorías de alto nivel:

- Operadores de cruce **puros**: aplicables en cualquier algoritmo genético.
- Operadores de cruce **híbridos**: que mezclan un operador puro con una técnica no genética.
- Operadores de cruce **dependientes del problema**: operadores que realizan operaciones basadas en el conocimiento del problema y por tanto son sólo aplicables a dicho problema.

Existen numerosos tipos de cruce. El más estándar es binario y genera dos hijos. Sin embargo no es raro encontrar aplicaciones que usan un cruce que genera un único hijo. También existen operadores de cruce típicos de problemas de reordenación y otros campos sobre todo de optimización combinatoria en los que dicho operador transmite de padres a hijos la admisibilidad como solución al problema.

El operador más tradicional [Gold89a] genera aleatoriamente el mismo punto de cruce (*Single Point Crossover* o SPX) en ambos padres e intercambia los 4 segmentos resultantes dando lugar a dos hijos nuevos. Existen numerosas extensiones a dos puntos (DPX) y *N* puntos [SD91]. Esta familia de operadores de cruce es eficiente y de aplicación bastante genérica.

Hemos desarrollado en trabajos anteriores [AT96] [AC97] una variante interesante de DPX que llamaremos DPX1 en la que se genera un único hijo de entre los dos posibles. El hijo generado es el que contiene la mayor proporción del mejor de los padres atendiendo a sus adecuaciones. El otro hijo nunca llega siquiera a generarse ni mucho menos evaluarse. Esta característica hace de DPX1 muy atractivo en algoritmos genéticos donde es crítico minimizar el número de evaluaciones [Reev93b] y por tanto resulta más eficiente generar un único hijo evitando así una evaluación que podría consumir elevados recursos computacionales durante la búsqueda (muchas aplicaciones sólo utilizan el mejor de los hijos, desperdiciando el esfuerzo dedicado a generar el otro).

Las siguientes definiciones describen formalmente los operadores que utilizaremos en este trabajo. Todas ellas presentan la versión (*bajo nivel*) de un cruce binario.

Definición 1.16. (Operador $\otimes_{SPX}$). *Dado un genotipo $\mathcal{G}$, una variable aleatoria discreta uniforme τ y un valor aleatorio α se define el operador de* un sólo punto de cruce *sobre dos vectores $\vec{a}$ y $\vec{b}$ como:*

$$\otimes_{SPX\ \{p_c\}} : \mathcal{G}\times\mathcal{G} \to \mathcal{G}\times\mathcal{G} \qquad \tau \in [0,1] \qquad \alpha \in \{1,...,l-1\}$$

$$\textbf{si } \tau \le p_c \textbf{ entonces } \otimes_{SPX\ \{p_c\}} \left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right)=((a_1,...,a_\alpha,b_{\alpha+1},...,b_l),(b_1,...,b_\alpha,a_{\alpha+1},...,a_l))$$

$$\textbf{si no} \qquad \otimes_{SPX\ \{p_c\}} \left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right)=((a_1,...,a_l)(b_1,...,b_l)) \ , \qquad (1.30)$$

donde $\Theta_{\otimes SPX}=\{p_c\}$es el conjunto de parámetros, consistente únicamente en la probabilidad p_c de realizar dicho cruce. ■

Definición 1.17. (Operador $\otimes_{DPX1}$). *Dado un genotipo $\mathcal{G}$, una variable aleatoria discreta uniforme τ y dos valores aleatorios α, σ se define el operador de* dos puntos de cruce y un sólo hijo *sobre dos vectores $\vec{a}$ y $\vec{b}$ como:*

$$\otimes_{DPX1\ \{p_c\}} : \mathcal{G}\times\mathcal{G} \to \mathcal{G} \qquad \tau \in [0,1] \qquad \alpha,\sigma \in \{1,...,l-1\}, \sigma > \alpha \qquad \boldsymbol{\Phi}(\vec{a}) \ge \boldsymbol{\Phi}(\vec{b}) \wedge (\sigma-\alpha) < l/2$$

$$\textbf{si } \tau \le p_c \textbf{ entonces } \otimes_{DPX1\ \{p_c\}} \left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right)=(a_1,...,a_\alpha,b_{\alpha+1},...,b_\sigma,a_{\sigma+1},...,a_l)$$

$$\textbf{si no} \qquad \otimes_{DPX1\ \{p_c\}} \left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right)=(a_1,...,a_l) \ , \qquad (1.31)$$

donde $\Theta_{\otimes DPX1}=\{p_c\}$es el conjunto de parámetros, consistente únicamente en la probabilidad p_c de realizar dicho cruce. ■

Definición 1.18. (Operador $\otimes_{UX}$ Sesgado). *Dado un genotipo $\mathcal{G}$, una variable aleatoria discreta uniforme τ y un conjunto de l variables aleatorias discretas α_i se define el operador de* cruce uniforme sesgado *sobre dos vectores $\vec{a}$ y $\vec{b}$ como:*

$$\otimes_{UX\ \{p_c,p_b\}} : \mathcal{G}\times\mathcal{G} \to \mathcal{G}\times\mathcal{G} \qquad \tau \in [0,1] \quad \alpha_i \in [0,1] \qquad \boldsymbol{\Phi}(\vec{a}) \ge \boldsymbol{\Phi}(\vec{b}) \quad \textit{normalmente } p_b \ge 0.5$$

$$\textbf{si } \tau \le p_c \textbf{ entonces } \otimes_{UX\ \{p_c,p_b\}} \left(\vec{a},\vec{b}\right)=\left(\left(\vec{a}',\vec{b}'\right)\right), a_i'=\begin{Bmatrix} a_i & \alpha_i \le p_b \\ b_i & \alpha_i > p_b \end{Bmatrix}, \ b_i'=\begin{Bmatrix} b_i & \alpha_i \le p_b \\ a_i & \alpha_i > p_b \end{Bmatrix} \ \forall i \in \{1,...,l\}$$

$$\textbf{si no} \qquad \otimes_{UX\ \{p_c,p_b\}} \left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right)=((a_1,...,a_l)(b_1,...,b_l)) \ , \qquad (1.32)$$

donde $\Theta_{\otimes UX}=\{p_c\ ,\ p_b\}$ es el conjunto de parámetros, consistente en la probabilidad p_c de realizar dicho cruce y la tendencia *(bias) p_b hacia utilizar cada parámetro del mejor padre.* ■

El operador estándar uniforme [Sysw89] es una particularización de nuestro operador uniforme sesgado en el que p_b=0.5, de forma que se genera una plantilla virtual equiprobable sobre las cadenas padre. Nuestro operador es más genérico y flexible. Este operador permite simular multitud de puntos de cruce y está ampliamente demostrada su utilización con éxito en problemas bastante distintos entre sí, muchas veces con resultados mejores que SPX o DPX [Sysw89] [AC97].

Definición 1.19. (Operador $\otimes_{AX}$ Sesgado). *Dado un genotipo $\mathcal{G}$ y una variable aleatoria discreta uniforme τ se define el operador de* cruce aritmético sesgado *sobre dos vectores $\vec{a}$ y $\vec{b}$ como:*

$\otimes_{AX\ \{p_c,p_b\}} : \mathcal{G}\times\mathcal{G} \to \mathcal{G}\times\mathcal{G} \quad \tau \in [0,1] \qquad \boldsymbol{\Phi}(\vec{a}) \geq \boldsymbol{\Phi}(\vec{b})$ *normalmente* $p_b \geq 0.5$

si $\tau \leq p_c$

entonces

$$\otimes_{AX\ \{p_c,p_b\}}(\vec{a},\vec{b}) = \left((\vec{a}',\vec{b}')\right), a_i' = p_b \cdot a_i + (1-p_b)\cdot b_i,\ \ b_i' = p_b \cdot b_i + (1-p_b)\cdot a_i \quad \forall i \in \{1,...,l\}$$

si no

$$\otimes_{AX\ \{p_c,p_b\}}\left(\vec{a}=(a_1,...,a_l),\vec{b}=(b_1,...,b_l)\right) = \left((a_1,...,a_l)(b_1,...,b_l)\right), \tag{1.33}$$

donde $\Theta_{\otimes UX}=\{p_c\ ,\ p_b\}$ es el conjunto de parámetros, consistente en la probabilidad p_c de realizar dicho cruce y el porcentaje p_b de cada parámetro del mejor padre. ■

El cruce aritmético tradicional es un tipo especial de nuestro operador aritmético sesgado. El sesgo p_b hacia el mejor padre de los dos últimos operadores permite introducir un componente de explotación en el comportamiento de exploración típico de cualquier cruce. Ambos operadores trabajan con independencia de la longitud de la cadena de parámetros, lo que supone una ventaja adicional en ciertos problemas. Véanse ejemplos de cómo trabaja cada operador en la Figura 1.9.

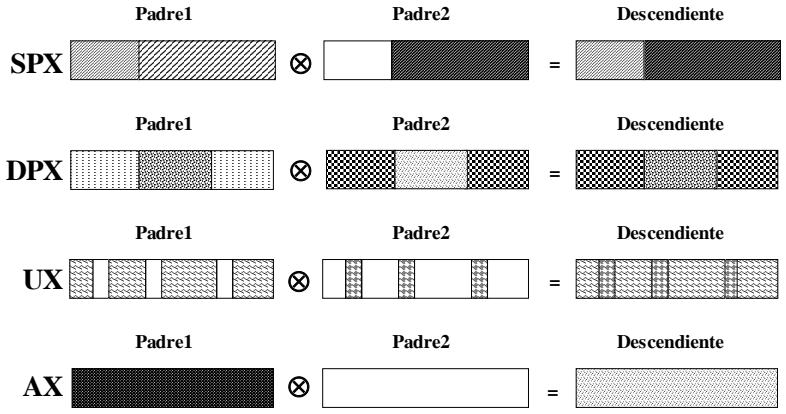


Figura 1.9. *Hijo resultante de aplicar distintos tipos de cruce.*

En general existen operadores típicos de ciertos tipos de problemas que muestran una mayor eficacia que el resto. Esto es así para UX o AX en el diseño de redes de neuronas [AC97], cruce ordenado (OX), de circuito (CX) o parcialmente especificado (PMX) para TSP [Mich92], o también cruces mejorados mediante la incorporación de conectivas de lógica difusa sobre todo para optimización de parámetros continuos [Loza96].

1.3.1.4. El Operador de Mutación *m*

El operador de mutación que utilizaremos en nuestros algoritmos genéticos se describe por la instanciación a dos operadores concretos del operador genérico mencionado para un AE. En el caso de genotipo binario se utiliza el tradicional operador de complemento de bits [Gold89a], mientras que en el caso de genotipo real utilizaremos una definición propia que se encuentra implementada de forma parecida en la literatura asociada a este tipo de codificación [Davi91a].

La mutación ha sido tradicionalmente entendida como un operador de segunda fila en los algoritmos genéticos. Sin embargo, numerosos estudios en este campo [Spea92] [TS93], así como en otros dominios donde la mutación es determinante, (ES y PE) han puesto de manifiesto la gran importancia de este operador en problemas de complejidad real, ya que es el único que introduce nueva información en una población secuencial, permitiendo así escapar de óptimos locales y ampliando la exploración del algoritmo. Las siguientes dos definiciones establecen formalmente su funcionamiento.

Definición 1.20. (Operador m_b -Binaria- y m_r -Real-). *Dado un genotipo $\mathcal{G}=\mathbb{B}$ y una variable aleatoria discreta uniforme α_j muestreada de nuevo para cada alelo binario, se define el operador de* mutación binaria *sobre un vector $\vec{s}$ como:*

$$m_{\{p_m\}}:\mathcal{G}\to\mathcal{G} \quad m_{\{p_m\}}(s_1,...,s_l)=(s'_1,...,s'_l) \;\; \forall j\in\{1,...,l\} \;\; s'_j=\begin{Bmatrix} s_j & \alpha_i > p_m \\ 1-s_j & \alpha_i \le p_m \end{Bmatrix} \quad \alpha_j\in[0,1], \qquad (1.34)$$

donde $\Theta_{m_b}=\{p_m\}$ es el conjunto de parámetros, consistente en la probabilidad p_m de mutar.

La mutación real *que utilizaremos consiste en restar o sumar un valor aleatorio al parámetro que se deba mutar, ajustando al rango de definición de la variable el resultado de dicha operación. El sumando es un porcentaje aleatorio elegido nuevamente para cada mutación del intervalo de definición de la variable que determina la perturbación introducida. Por tanto, el conjunto de parámetros es $\Theta_{m_r}=\{p_m\}$ y cada parámetro que deba ser mutado cambiará según:*

$$s'_j=\left(s_j \pm p_{m_r}\cdot s_j\right)^{u_j}_{l_j} . \qquad (1.35)$$

Otros modelos son posibles, como la mutación Normal (*Gaussiana*) [BFM97]. ■

Tanto el cruce como la mutación pueden sufrir un cambio *dinámico* durante la evolución en su conjunto de parámetros, típicamente en la probabilidad de aplicación. Esta característica ha proporcionado buenos resultados en algunos campos de aplicación [Foga89] aunque aún es un proceso relativamente caro debido a que dichos cambios surgen como consecuencia de una monitorización de la diversidad de la población (menor distancia de Hamming, mayor mutación [Davi89] [HMP96]), o de la aportación que hacen a la adecuación de los descendientes generados [Jusl95].

A veces es incluso más beneficio y menos caro predefinir un plan de cambio de dicha probabilidad antes de que empiece la evolución. Por ejemplo, modificando la probabilidad de forma lineal o exponencialmente creciente/decreciente entre la generación inicial y la última, siempre que el criterio de terminación imponga una generación tope.

1.3.1.5. Criterios Utilizados para el Análisis

Una vez establecido el tipo de operadores que nos interesan (en cada estudio se ofrecerán en concreto cuáles de ellos intervienen y cuál será la parametrización usada) pasamos a definir los criterios que usaremos para estudiar los modelos secuenciales y paralelos. En general, éstos se basan en determinar el número de evaluaciones que dos algoritmos necesitan para resolver el mismo problema y el tiempo empleado. Este tipo de medida es especialmente importante en algoritmos paralelos donde no es posible hacer una comparación útil (ni siquiera justa) si el criterio de terminación es preestablecer el mismo número de evaluaciones para cualquier algoritmo comparado.

Por tanto utilizaremos, bien directamente el número de evaluaciones de la función objetivo, o bien un concepto que podemos elaborar para ello denominado *esfuerzo de evaluación*.

Definición 1.21. (Esfuerzo de Evaluación). *Dado un AE* ($\mu+\lambda$) *con tiempo de ejecución* τ_{AE} *definimos el* esfuerzo de evaluación *necesario para resolver el problema de optimización global con función objetivo f como:*

$$\boldsymbol{E}_{AE}(f) = \lambda \cdot \tau_{AE} \ . \tag{1.36}$$

■

Parece claro que al comparar dos algoritmos (secuenciales o paralelos) podemos obtener una idea clara de la eficiencia numérica del proceso de búsqueda atendiendo al esfuerzo computacional necesario para alcanzar una solución de igual adecuación en cada algoritmo. Este tipo de medida defendida en recientes estudios [HBBK97] [CG97] es especialmente importante en trabajos con algoritmos evolutivos paralelos, en los que no se puede utilizar como medida de eficiencia ningún otro criterio, ya que dos ejecuciones del mismo algoritmo son siempre potencialmente distintas, y además el comportamiento en la búsqueda es también diferente.

Esta idea podemos formalizarla y concretar un criterio de comparación atendiendo al esfuerzo computacional que necesitan dos algoritmos evolutivos para resolver el mismo problema.

Definición 1.22. (Eficiencia Numérica). *Dados dos algoritmos evolutivos* AE_1 *y* AE_2 *utilizados para resolver un problema de optimización global sobre una función objetivo f, decimos que el primero es más* eficiente *que el segundo (numéricamente) sii*

$$\boldsymbol{E}_{AE_1}(f) < \boldsymbol{E}_{AE_2}(f) \ . \tag{1.37}$$

■

El interés de este concepto es clave en este trabajo ya que proponemos entre nuestros objetivos conseguir algoritmos evolutivos (genéticos en concreto) paralelos de elevada eficiencia numérica. Un concepto igualmente interesante es el de *eficiencia numérica marginal* (Definición 1.24). Sin embargo, necesitamos antes un criterio para establecer una relación entre dos conjuntos de valores (Definición 1.23).

Definición 1.23. (Concepto de Cubrimiento). *Dados dos conjuntos de valores reales* $A=\{a_1,...,a_n\}$ *y* $B=\{b_1,...,b_n\}$*, decimos que B* cubre *a A y lo notaremos* $A \leq B$ *sii*

$$a_i \leq b_i \qquad \forall i \in \{1,...,n\} \ . \tag{1.38}$$

■

Definición 1.24. (Eficiencia Numérica Marginal). *Dados dos algoritmos evolutivos AE_1 y AE_2 utilizados para resolver un conjunto de problemas de optimización global $\mathbf{F}=\{f_1, ..., f_k\}$ que presentan esfuerzos computacionales $\mathbf{E}_{AE_1}(f_i)$ y $\mathbf{E}_{AE_2}(f_i)$ decimos que el primero es marginalmente más eficiente que el segundo (numéricamente) sii -un conjunto cubre al otro-*

$$\forall i \in \{1,...,k\} \qquad \mathbf{E}_{AE_1}(f_i) \leq \mathbf{E}_{AE_2}(f_i) . \tag{1.39}$$

■

Demostrar que un algoritmo evolutivo es siempre más eficiente que otro dado, incluso fijando el tipo de problema, es en general una tarea difícil debido a para el mismo tipo de problemas existen instancias considerablemente distintas entre sí que provocan distintos comportamientos. De hecho, existe una familia de teoremas conocidos como NFL (*No Free Lunch*) [WM97] que demuestran la imposibilidad de que un algoritmo sea mejor que otro sobre un problema arbitrario cualquiera, ya que en término medio todos presentan iguales resultados.

Aunque estas consideraciones hagan imposible definir de forma genérica la superioridad de uno u otro algoritmo en general y con independencia del problema, sí que es posible utilizar instancias de problemas complejos y útiles a la vez en donde medir la eficiencia numérica marginal. En nuestro caso, utilizaremos problemas epistáticos, con extensos espacios de búsqueda y un gran número de óptimos que además en algunos casos requerirán un importante refinamiento de los valores codificados (Apéndice A).

Estamos interesados en estudiar modelos paralelos de mayor eficiencia que otros modelos tradicionales, tanto secuenciales como paralelos, sobre un conjunto de problemas lo suficientemente representativo, complejo y a la vez útil [Sten93] [GW93]. El objetivo es obtener una aportación interesante desde el punto de vista de la investigación teórica tanto como para permitir nuevas aplicaciones hasta ahora vedadas a estos algoritmos (Figura 1.9).

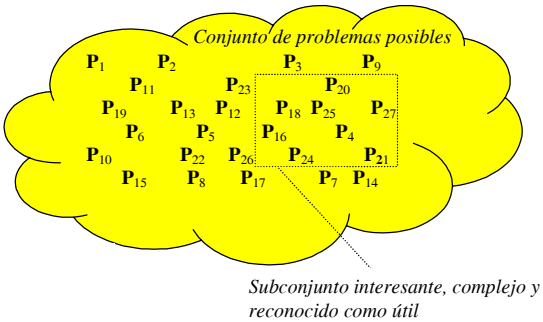


Figura 1.9. *Conjunto de problemas y subconjunto de evaluación en computación evolutiva.*

Otra medida de comparación que utilizaremos típica en el estudio de algoritmos paralelos es la ganancia en velocidad (*speedup*) medida como la relación entre el tiempo medio real en resolver un problema sobre un procesador y el tiempo en resolverlo con n_{proc} procesadores.

La definición y uso del concepto de *speedup* en el campo de los algoritmos estocásticos paralelos como es el caso de un AGP requiere algunas consideraciones especiales sobre el criterio de terminación.

Como se demuestra en [HBBK97] y [CG97], un AGP debe ejecutarse hasta encontrar una solución de calidad equivalente a la del resto de modelos comparados, sean secuenciales o paralelos, ya que detener el algoritmo cuando ha ejecutado un número de pasos predefinido no es un criterio justo ni significativo.

Definición 1.25. (Ganancia en Velocidad -*Speedup*-). *Dados dos algoritmos evolutivos AE_1 ejecutado sobre un único procesador y $AE_{n_{proc}}$ que utiliza n_{proc} procesadores, definimos el concepto de* ganancia $\mathcal{S}(n_{proc})$ *como*

$$\mathcal{S}\left(n_{nproc}\right) = \frac{T_1}{T_{n_{proc}}} \tag{1.40}$$

y también los conceptos de:

$$\begin{array}{lcl} \text{ganancia} & \Leftrightarrow & \mathcal{S}(n_{proc})>1 \\ \text{ganancia lineal} & \Leftrightarrow & \mathcal{S}(n_{proc})= n_{proc} \\ \text{ganancia superlineal} & \Leftrightarrow & \mathcal{S}(n_{proc})> n_{proc} \end{array} \tag{1.41}$$

■

Es evidente que la ganancia en velocidad únicamente tiene sentido cuando hablamos de dos algoritmos evolutivos de idéntico comportamiento. Además, debemos hacer otra consideración ya que es posible conseguir ganancias superlineales al utilizar un AGP. Esta diferencia respecto a los algoritmos deterministas [Akl92] se debe a que, como se demuestra en [Shon93]:

"para cualquier tiempo T, la probabilidad del algoritmo estocástico de encontrar una solución tras buscar durante dicho tiempo es no nula"

Las ganancias superlineales son relativamente usuales en este campo [Beld95] [Shon93] [AT99b], e incluso se ha propuesto un modelo de ganancia teórico como el descrito por la ecuación:

$$\mathcal{S}\left(n_{nproc}\right) = n_{proc} \cdot s^{n_{proc}-1} \tag{1.42}$$

donde s el *factor de aceleración* y $s>1$ produciría ganancias superlineales.

1.3.2. Necesidad del Paradigma de Orientación a Objetos

Del contenido de las secciones anteriores (diversidad de modelos, implementaciones, técnicas básicas, sistemas de comunicación, etc.) podemos extraer las siguientes conclusiones:

- Los modelos secuenciales, y especialmente los paralelos, de algoritmos genéticos representan en general sistemas *software* complejos que necesitan de un diseño unificado y estructurado.
- La frecuente extracción de prototipos concede ventaja a los modelos de AGPs que sean muy flexibles y fácilmente extensibles-aplicables a un problema dado.
- El trabajo, tanto en subpoblaciones distribuidas como en vecindarios, los cambios de topología o vecindario, el estudio del impacto de estos modos de evolución, etc. hacen necesario un sistema fácilmente parametrizable y que pueda combinar técnicas entre sí.

- La frecuente monitorización de valores y el estudio del estado de ciertos elementos del algoritmo precisan de una subdivisión del sistema resultante en módulos.
- Las distintas tareas, operadores y técnicas que combinadas proporcionan el sistema evolutivo requieren trabajar en cualquier orden entre sí y con técnicas nuevas o no evolutivas. Esta relación de combinación/competición precisa que los subsistemas componentes presenten una interfaz clara y flexible al resto de elementos del sistema.

Debido a estas razones no es de extrañar que, a pesar de que se han utilizado una gran variedad de lenguajes para implementar sistemas evolutivos como LISP, PASCAL, C, y muchos otros menos conocidos [TA91], e incluso dependientes de la máquina, las tendencias actuales conduzcan a usar lenguajes orientados a objetos donde todas las características mencionadas están presentes para permitir desarrollar sistemas complejos como los incluidos en GAGS, GAME, GAlib, TOLKIEN, EVOLVER y varios más.

En especial C++ [Stro91] es un lenguaje muy extendido que incorpora estas y otras características de forma natural. Otros lenguajes más recientes como JAVA [Pew96] permiten también un diseño muy elegante, pero en la actualidad son lentos en la ejecución de algoritmos con numerosos niveles de clases relacionadas entre sí, como demostraremos con un caso de estudio en el Capítulo 3 (en la sección dedicada a implementación).

De entre el *software* existente, quizás el intento más serio y conocido sea el entorno de manipulación genética GAME y recientemente RPL2. El primero contiene una máquina virtual que permite manipular información en forma de poblaciones de individuos de muy variado contenido y trabajar con operadores o técnicas de evolución distintas entre sí y fácilmente definibles por el usuario. Presentaremos en la sección de implementación (Capítulo 3) los diagramas de clases para GAME y otros sistemas *software* diseñados usando OMT [Rumb91].

La principal ventaja de GAME es a la vez su principal inconveniente. Se trata de un entorno tan flexible y tan genérico que la aplicación a un problema concreto es tediosa e incluso ineficiente. GAME no es muy utilizado debido además a que contiene numerosos errores de codificación internos. Sin embargo, representa un esfuerzo conjunto europeo útil en el diseño unificado de AEs, potenciando los aspectos de monitorización y paralelización. RPL2 sí que parece estar más dotado, aunque es un sistema comercial y sus aplicaciones son aún reducidas.

En general, la programación orientada a objetos dispone de características muy interesantes como la herencia, el polimorfismo, la abstracción y otros elementos que sobre el papel son todos ellos deseables. Sin embargo, una implementación real supone un *compromiso entre un buen diseño y una ejecución eficiente*. El concepto de herencia por ejemplo a niveles tan inferiores como los alelos, genes e individuos puede repercutir fácilmente en duplicar o triplicar el espacio de memoria necesario. Igualmente, la abstracción y continua creación/copia/destrucción de instancias temporales de las clases diseñadas pueden hacer muy lento el sistema final.

Los siguientes ejemplos representan algunos intentos de conseguir este compromiso entre diseño y eficiencia. El sistema dGANN [AC97] [Garc96] es un entorno distribuido de migración para diseño de redes de neuronas. El diseño e implementación de dGAME [Manj96] igualmente proporciona un sistema flexible paralelo distribuido en la práctica pero que arrastra algunas de las desventajas de su contrapartida secuencial. En esta extensión sí que se pueden obtener ventajas en cuanto a variedad de topologías y mezcla de AGPs.

Por último, el entorno jdcGA [Muño98] permite ejecutar en JAVA versiones distribuidas de algoritmos evolutivos en cierta medida similares a los que estudiamos en este trabajo. Sin embargo, su ejecución es muy lenta en comparación con implementaciones hechas en C++ y por tanto sólo su diseño resulta interesante por el momento. Volveremos a ellos con más detalle en la sección de implementación del Capítulo 3. Otras alternativas basadas por ejemplo en el uso de objetos distribuidos [OHE96] son también interesantes de evaluar para el trabajo futuro.

Existe una alternativa/extensión posible al uso en forma de diseño orientado a objetos conocida como Marcos de Programación (*Programming Frames*). En ellos la interfaz con el usuario y los niveles de abstracción se cuidan especialmente con el objetivo de facilitar el uso del *software* paralelo (un AG paralelo en nuestro caso) a personas no especialistas en paralelismo. Hay una cierta relación y un conjunto de similitudes entre el diseño orientado a objetos y dichas arquitecturas de modelado que analizaremos al final del Capítulo 3.

Con esta introducción a la importancia de la POO en estos sistemas *software* acabamos el Capítulo 1. El siguiente capítulo formaliza el concepto de sistema adaptativo granulado y deriva a partir de él los algoritmos genéticos secuenciales y paralelos, demostrando que todos ellos pertenecen al mismo tipo de plantilla de búsqueda. También en el Capítulo 2 introduciremos algunos resultados que demuestran que en el estudio de AGPs debe cuidarse el mecanismo básico de selección de parejas y ambiental si queremos obtener mayor eficiencia.

Para terminar, incluiremos en el siguiente capítulo un conjunto de trabajos que demuestren la necesidad y utilidad de los trabajos precursores de los algoritmos analizados para solucionar con éxito a la evolución de estructuras de datos complejas, incluso más allá de los simples vectores de valores binarios o reales.

2
Caracterización Unificada de los Algoritmos Genéticos Secuenciales y Paralelos

El amplio éxito de los algoritmos evolutivos en múltiples disciplinas ha dado lugar a numerosos trabajos sobre sus aplicaciones, desarrollos teóricos y estudios empíricos. Sin embargo, no se suele disponer de trabajos donde se parta de una descripción formal fundamentada seguida de una posterior derivación de algoritmos útiles en la práctica. En este trabajo pretendemos llenar este hueco y empezamos en este capítulo por mostrar una formalización genérica de sistema adaptativo y una particularización de dicho sistema para dar lugar a las subclases de los algoritmos genéticos secuenciales y paralelos.

En un análisis preliminar estudiamos su robustez, presentando brevemente su aplicación sobre problemas complejos. Concretamente, pretendemos caracterizar tanto la clase de los algoritmos evolutivos secuenciales como paralelos basándonos en el tipo de manipulaciones que realizan sobre la población. Posteriormente se derivan de esta clase genérica tres subclases que dan lugar a los conocidos como algoritmos genéticos de población única [Gold89a], celular [SM91] y distribuida [Tane89].

Tradicionalmente se distingue entre AGPs de grano grueso y de grano fino según que la relación entre computación y comunicación sea alta o no, respectivamente. En el presente trabajo no se proporciona esta visión tradicional del paralelismo, más bien los AGPs distribuidos (grano grueso) se consideran como una superclase que engloba a la clase secuencial y a la celular tanto como a cualquier otro esquema de evolución básica, definiendo de esta forma un sistema de búsqueda distribuida con migración de estructuras entre los algoritmos. La visión complementaria es también posible, ya que, como vimos en la Figura 1.7, existe un "continuo" entre los modelos descentralizados.

El contenido de este capítulo comienza en la Sección 2.1 con la formalización de un “sistema adaptativo”. La Sección 2.2 demuestra cómo un algoritmo genético secuencial o paralelo es una subclase de esta clase más genérica de plantilla de búsqueda. La Sección 2.3 está centrada en demostrar la robustez de los algoritmos genéticos secuenciales y poner de manifiesto de forma práctica las diferencias de comportamiento entre clases de mecanismos de evolución secuencial, así como justificar la necesidad de modelos paralelos para resolver problemas complejos.

Las Secciones 2.4 y 2.5 están dirigidas a plantear formalmente el trabajo de los modelos celulares y distribuidos, respectivamente. En la Sección 2.6 ofrecemos un pseudocódigo básico como primer acercamiento a la propuesta de familia de algoritmos evolutivos paralelos que concretaremos en el Capítulo 3. La Sección 2.7 contiene los fundamentos y la presión selectiva del comportamiento canónico de todos los modelos derivados; finalmente, la Sección 2.8 revisa las conclusiones de los resultados mostrados en este capítulo.

2.1. Formalización Unificada

Para estudiar el comportamiento de los algoritmos evolutivos es necesario en primer lugar caracterizarlos. En esta sección presentamos un concepto lo suficientemente general como para contener como subclases a un algoritmo evolutivo cualquiera y en particular a un AG secuencial o paralelo. Partiendo de la definición de *sistema adaptativo* (SA) hecha en [DRH95], proponemos los siguientes conceptos que nos resultarán útiles:

Definición 2.1. (Sistema Adaptativo). *Definimos un* sistema adaptativo *como una tupla*

$$\mathcal{S} = (A, \vartheta, \tau, B, \vartheta_B, \tau_B, E) \tag{2.1}$$

donde:

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

Nótese que existe una gran similitud entre los tres primeros y los tres siguientes elementos de $\mathcal{S}$. Básicamente, ambas ternas corresponden a los dominios del genotipo $\mathcal{G}$ y el fenotipo $\mathcal{P}$, respectivamente (i.e., los primeros son representaciones internas de los segundos).

El siguiente concepto que es necesario definir hace referencia a la *granularidad* del algoritmo. Un sistema adaptativo granulado está internamente estructurado en *gránulos* comunicantes. Dichos gránulos (modos de funcionamiento) son las unidades estructurales que, combinadas de manera apropiada, dan lugar a un algoritmo.

Definición 2.2. (Sistema Adaptativo Granulado). *Se define un* sistema adaptativo granulado *como una tupla*

$$\Xi = (\Delta, \Pi, \xi) \tag{2.2}$$

donde:

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

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

Está demostrado [Bäck96] [CAT98a] [Cott98] que la plantilla de un sistema adaptativo granulado es lo suficientemente potente y general como para realizar un cómputo arbitrario. Dicha demostración se basa en el hecho de que cualquier computación puede ser representada como una máquina de Turing no determinista [PS82] (asumiendo modelos computacionales discretos) y mostrando que un sistema adaptativo granulado puede simularla.

En este capítulo pretendemos derivar los algoritmos genéticos a partir de las definiciones genéricas sobre sistemas adaptativos. En particular, derivaremos los algoritmos genéticos de población única, tanto de estado estacionario como generacionales, los algoritmos celulares y los algoritmos distribuidos. Para conseguir estos objetivos debemos ir instanciando progresivamente los elementos de un sistema adaptativo.

2.2. Derivación Unificada de Modelos Des/Centralizados

Un algoritmo genético de población única centralizado es un sistema adaptativo granulado que consta de un solo gránulo (véase la definición 2.2) de forma que, si la tupla genérica la notamos como $\Xi_{cent}=(\Delta_{cent}, \Pi_{cnet}, \xi_{cent})$, cada componente se define como:

- $\Delta_{cent} = \{\Delta_1\}$
- Π_{cent} y ξ_{cent} son indiferentes, ya que sólo hay un gránulo.

Esta instanciación de valores para la tupla genérica basta para definir cualquier AG centralizado sugiriendo su implementación secuencial. Para distinguir las variantes de estado estacionario y generacional necesitamos especificar el contenido del gránulo.

Definición 2.3. (Algoritmo Genético Centralizado). *Definimos un* algoritmo genético centralizado *como un sistema adaptativo granulado con un único gránulo*

$$\Delta = (A, \vartheta, \tau, B, \vartheta_B, \tau_B, E) \tag{2.3}$$

- $A = \{A_1, A_2, ...\}$ *es el* conjunto de poblaciones posibles. *De forma que cualquier población generada perteneciente a la secuencia de poblaciones* $P(i) \in \bigcup_{t=0}^{\tau_{AE}} \Psi^t(P(0))$ *también pertenece a A, es decir,* $\exists i, j \quad P(i) = A_j$.

- $\vartheta = \{\omega_{cent}\}$. *Existe un único operador denominado* ciclo reproductor central. *Este operador se define por la concatenación de cuatro operadores internos*:

$$\omega_s \equiv s_{\Theta_s} \qquad \omega_c \equiv \otimes_{\Theta_c} \qquad \omega_m \equiv m_{\Theta_m} \qquad \omega_r \equiv r_{\Theta_r}$$

$$\omega_{cent}(A_i,E_i) = \omega_r(A_i \cup \{\omega_m(\vec{a}_{ij}) \mid \vec{a}_{ij} \in \omega_c(\omega_s(A_i,E_i))\})$$

- $\tau : E \to \vartheta$ *tal que* $\tau(E_i)=\omega_{cent}(A_i,E_i)$.
- $B = \{B_1, B_2, ...\}$, *tal que*, $B_i = (\vec{b}_{i_1},...,\vec{b}_{i\mu})$, *donde* $\vec{b}_{i_j} \in \mathcal{P}$ (conjunto de fenotipos). *La* función de representación $\Gamma : \mathcal{P} \to \mathcal{G}$ es tal que $\vec{b}_{ij} = \Gamma^{-1}(\vec{a}_{ij})$. Podemos extender la notación para trabajar a alto nivel sobre toda la población: $\Gamma^{-1}(A_i) = (\Gamma^{-1}(\vec{a}_{i1}),..., \Gamma^{-1}(\vec{a}_{i\mu}))$.
- $\vartheta_B=\{\omega_{Bcent}\}$ *donde* $\omega_{Bcent}(B_i,E_i) = \Gamma^{-1}\circ(\omega_{cent}(\Gamma^{-1}\circ B_i,E_i))$.
- $\tau_B : E \to \vartheta_B$ *se define como* $\tau_B(E)=\omega_{Bcent}(B_i,E)$.
- $E=\mathbb{R}^{\mu}$, *contiene los* valores de adecuación *de las estructuras en la población.* ■

El funcionamiento de un AG consiste pues en la repetida aplicación del operador de ciclo reproductor sobre una población de estructuras codificadas (A_i). Son los detalles de los operadores internos selección (ω_s) y reemplazo (ω_r) los que diferencian a los AGs generacionales ($\lambda=\mu$) de los AGs de estado estacionario (λ=1 ó 2) -véase el Capítulo 1-. Observe el uso de conceptos más genéricos como conjunto de poblaciones posibles y entorno, ambos inspirados en la filosofía inicial de Holland.

Por otro lado, un algoritmo genético descentralizado (o de población estructurada) es un sistema adaptativo granulado de μ gránulos que se define como sigue:

Definición 2.4 (Algoritmo Genético Descentralizado). *Un* algoritmo genético descentralizado *es una tupla*

$$\Xi_{descent} = (\Delta_{descent}, \Pi_{descent}, \xi_{descent}) \tag{2.4}$$

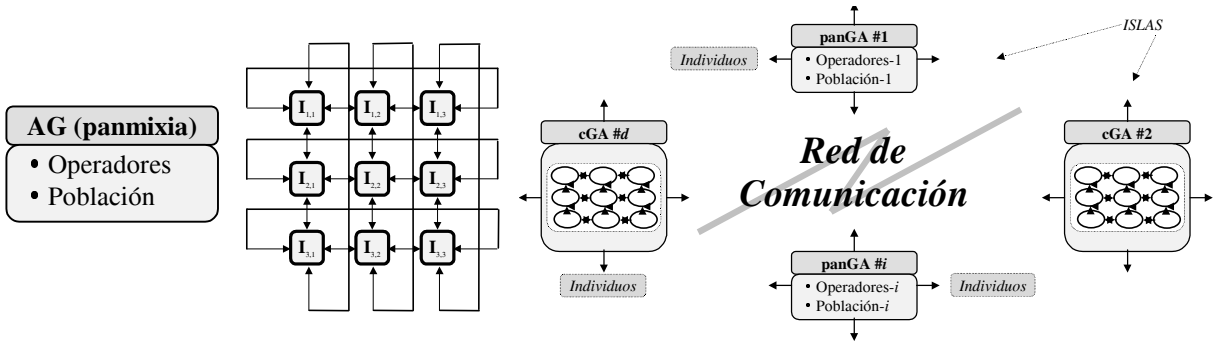
en la que:

- $\Delta_{descent} = \{\Delta_1, ..., \Delta_\mu,\}$
- $\Pi_{descent} \equiv \Delta_1 \parallel \Delta_2 \parallel ... \parallel \Delta_\mu$*, es decir, la política de activación es* concurrente *(implementada con paralelismo físico normalmente), usualmente sincronizada entre los gránulos*.
- $$\xi_{descent}(\Delta_i, \Delta_j) = \begin{cases} \varnothing & j \notin \upsilon(i) \\ \chi_{ij}(B^i) & j \in \upsilon(i) \end{cases}$$

Esta última expresión caracteriza la comunicación *entre dos gránulos. En ella, B^i representa el conjunto de estructuras en el gránulo Δ_i. La función χ_{ij} selecciona de entre éstas a las estructuras distinguidas que se comparten con el gránulo j. Usualmente $\chi_{ij} = \chi$, es decir, el mecanismo de comunicación es homogéneo. La función $\upsilon(i) : \mathbb{N} \rightarrow P(\mathbb{N})$ se denomina* función de vecindad *y determina qué gránulos comparten estructuras.* ■

Los modelos secuenciales y de paralelización global quedan caracterizados por la definición de sistema adaptativo granulado centralizado. Los dos modelos de algoritmos genéticos paralelos tradicionales, celular y distribuido, quedan caracterizados por la anterior definición de AG descentralizado. La diferencia entre ambos estriba en que, en los primeros, cada gránulo Δ_i contiene un vecindario propio $\upsilon(i)=\{ \upsilon_1, \upsilon_2, ..., \upsilon_{\mu^i} \}$, siendo $\mu^i = |\upsilon(i)|$, y la función χ_{ij} selecciona siempre al mismo elemento distinguido de este vecindario ($\chi_{ij} \neq \chi_{ik}(b_i) \mid k \neq j;\ k,j \in \upsilon(i)$). En los segundos, cada gránulo contiene una subpoblación independiente (isla) de tamaño arbitrario, consistiendo usualmente la función χ_{ij} en la selección de la mejor estructura de cada isla o de una estructura aleatoria de ésta. Adicionalmente, los AGPs celulares se sincronizan al final de cada ciclo reproductor, mientras que los AGPs distribuidos lo hacen con menor frecuencia. La función de comunicación $\xi_{descent}$ forma parte del mecanismo de comunicación señalado en el Algoritmo 1.4 del capítulo anterior (usada para seleccionar los emigrantes).

Esta definición genérica nos permite derivar nuevos modelos paralelos en los que varios algoritmos celulares evolucionan en paralelo con migración esporádica de individuos entre ellos (por ejemplo en anillo) -dcGA- o bien a la inversa -cdGA-. Además, por supuesto, podemos derivar las tradicionales islas de AGs secuenciales en panmixia, a los que llamaremos dpanGA (Figura 2.1). En definitiva, se trata de una jerarquía de sistemas adaptativos granulados.



(a) (b) (c)

Figura 2.1. *Esquema general de los modelos de población única* (a)*, celular* (b) *y una implementación en red de un AG paralelo distribuido heterogéneo* (c)*.*

2.3. La Búsqueda de Modelos Eficientes. Preliminares

Es notable en la actualidad el interés por encontrar algoritmos genéticos secuenciales cada vez más eficientes sin pérdida de capacidad de aplicación a nuevos problemas. En realidad, los algoritmos genéticos, como muchos otros mecanismos de búsqueda (divide-y-vencerás, retroceso, enfriamiento simulado, ...), mejoran su comportamiento al utilizar información sobre el problema para guiar dicha búsqueda.

Como ejemplos generales de este proceso de hibridación podemos citar la hibridación de un AG con técnicas de escalada como operadores genéticos, los trabajos que intentan afinar las funciones que acotan el coste en algoritmos divide-y-vencerás, los trabajos relacionados con afinar el valor inicial y final de la temperatura y generación de vecinos en el enfriamiento simulado, la extensión del retroceso para dar lugar a los algoritmos de Las Vegas, ...

Por tanto, los metaheurísticos hibridados suelen comportarse mejor que su contrapartida pura para un problema concreto. Sin embargo, el precio es la mayor complejidad y casi total dependencia del problema. Es nuestro objetivo proporcionar mejoras que afecten principalmente a la eficiencia de la búsqueda de un algoritmo genético, y que, además, sean susceptibles de otras mejoras ulteriores relativas a hibridación o cualquier otro tipo de proceso.

En esta sección pretendemos demostrar cómo un algoritmo genético es capaz de competir con otras técnicas en la resolución del mismo problema, incluso en su estado más estándar y simple (Sección 2.3.1). A continuación, en la Sección 2.3.2, presentamos la visión complementaria a esta, relativa a su capacidad de resolver problemas distintos entre sí, comprobando en cada caso de estudio cómo el resultado mejora no únicamente si se lo hibrida con otra técnica, sino sobre todo cuando se trabaja con un modelo paralelo distribuido (véase Figura 2.2).

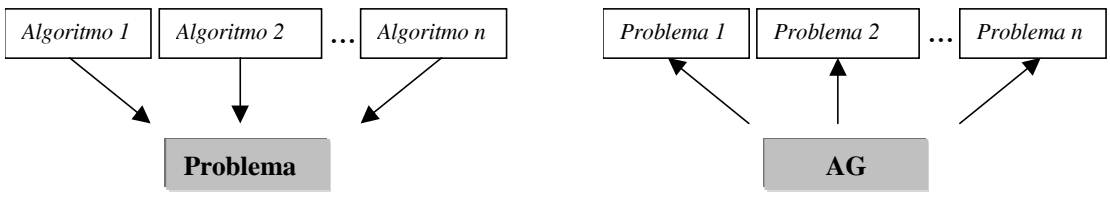


Figura 2.2. *Guión gráfico de las dos siguientes secciones.*

2.3.1. Relación con otras Técnicas

Para comprender el tipo de búsqueda que realiza un algoritmo genético es importante en primer lugar presentar su comportamiento respecto a otros algoritmos tradicionales. Aunque existen algunas comparativas sobre este tema vamos a mostrar resultados propios que nos permitan centrar la discusión y aportar conclusiones gráficas y numéricas. Naturalmente, para tomar una visión de conjunto todos ellos serán usados en su concepción más primitiva y, por tanto, de cualquiera de ellos pueden encontrarse variantes más eficientes.

En esta sección usamos quizás el modelo más estándar y básico de algoritmo genético generacional 1-elitista (incluido en el sistema GENESIS [Gref84]). Para demostrar algunas propiedades básicas de los algoritmos de búsqueda realizamos una comparativa de las técnicas naturales y tradicionales presentadas en la Tabla 2.1 (véanse las referencias en la tabla para más detalles sobre ellas).

TABLA 2.1. METÁFORA NATURAL QUE INSPIRA A LAS TÉCNICAS ESTUDIADAS

Técnica Heurística	**Proceso natural en que se basa**
Red de Neuronas	Neurofisiología del cerebro [RM89]
Algoritmo Genético	Selección natural y cruce de especies [Holl75]
Enfriamiento Simulado	Enfriamiento (recocido) de un sólido [Ingb93]
Técnica Tradicional	**Tipo de algoritmo**
Retroceso (Backtracking)	Búsqueda sistemática no guiada [BB88]
Algoritmo de Las Vegas	Búsqueda aleatoria no guiada [BB88]

Para este objetivo hemos seleccionado el bien conocido problema de situar N reinas en un tablero de ajedrez sin que se ataquen entre sí [BB88]. Se trata de un problema de optimización combinatoria muy conocido, no especialmente complejo para pequeños valores de N (Figura 2.3a) pero del que podemos extraer instancias de cualquier tamaño, lo que lo hace idóneo para comprobar características relativas entre diferentes métodos de búsqueda. El hecho de usar permutaciones (característica de todos los algoritmos comparados en esta sección) reduce el espacio de búsqueda desde N^N a ser $N!$ (Figura 2.3b) y asegura que los puntos generados son admisibles. La violación de restricciones se maneja en todos los algoritmos (si procede) como penalización de la función objetivo. Se pretende pues minimizar una función objetivo que cuenta el número de ataques entre las reinas. Para más detalles sobre los algoritmos y resultados consulte [AD97].

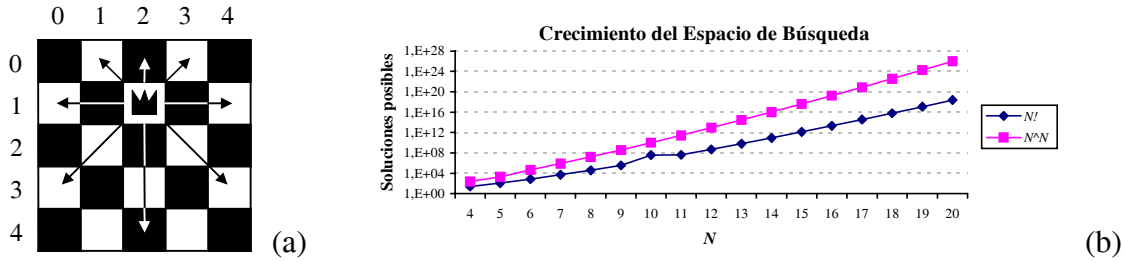


Figura 2.3. (a) *Movimientos legales;* (b) *crecimiento del espacio de búsqueda.*

Las Figuras 2.4 y 2.5 resumen la clase de resultados típicos cuando se mide el esfuerzo de evaluación y el tiempo real necesario para resolver instancias cada vez mayores de un problema (promedio de 100 ejecuciones independientes). Podemos observar cómo el AG generacional compite con una red de neuronas hecha a la medida [Take92] en cuanto a esfuerzo de evaluación. Igualmente, en [AD97] presentamos a ambos (junto con enfriamiento simulado) como los únicos que solucionan instancias altas del problema (*N*=512, 1024, ...). El esfuerzo de evaluación es uno de los mejores criterios para caracterizar la calidad de la búsqueda de un algoritmo, ya que determina el número de puntos visitados. En este aspecto otros algoritmos de inspiración natural como el enfriamiento simulado han proporcionado peores resultados.

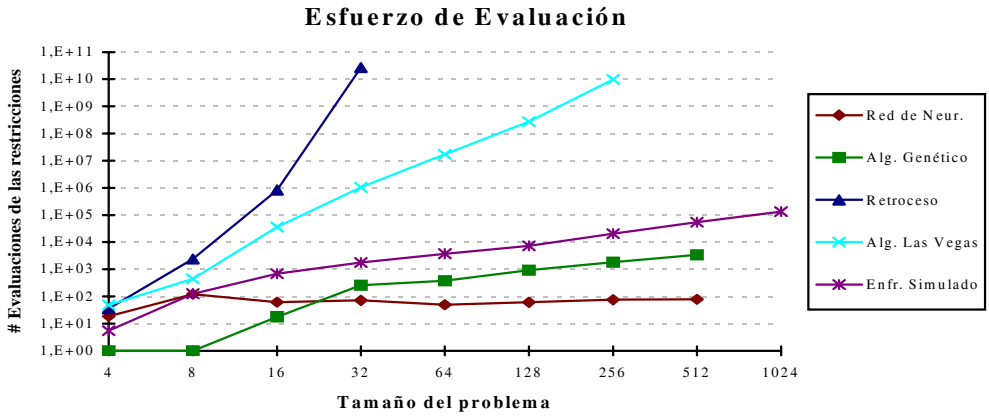


Figura 2.4. *Comparación del número medio de evaluaciones de las restricciones del problema.*

Vemos que para un crecimiento exponencial del tamaño del problema el AG es comparable a un algoritmo muy sofisticado hecho a la medida, y mejor que el resto de técnicas naturales tradicionales comparadas. En cuanto al tiempo real, el AG generacional es peor que ambas y de ahí nuestro interés en reducir el tiempo recurriendo a una ejecución paralela (que además disminuirá con frecuencia el esfuerzo de evaluación).

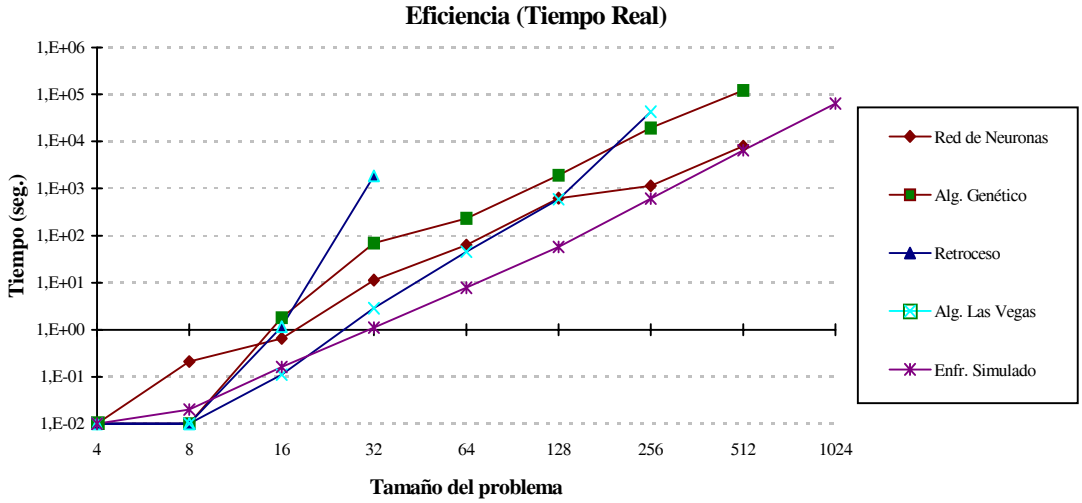


Figura 2.5. *Tiempo real (s) empleado en una estación Sparc1 por las técnicas estudiadas.*

Como detalle interesante debemos mencionar que la implementación usada para el AG es lenta debido a que GENESIS realiza operaciones internas innecesarias dada su generalidad [Gref90]. Es por ello que desde el capítulo anterior hacemos hincapié en obtener generalidad y flexibilidad, pero no a coste de eficiencia en la ejecución. Igualmente, parece necesario mejorar el modelo básico de evolución generacional para mejorar la velocidad de la búsqueda.

Como consecuencia, es evidente en este conjunto de problemas que una búsqueda adaptativa e inteligente en el espacio de soluciones posibles da lugar a un número menor de evaluaciones de las restricciones del problema de las *N*-Reinas que una búsqueda aleatoria no guiada como la que realiza el algoritmo de LV o una exhaustiva como la de la técnica de retroceso (*backtracking*).

En cuanto a las técnicas de inspiración natural, el enfriamiento simulado es muy eficaz y rápido pero resulta menos eficiente que la RN y el AG en cuanto al número de comprobaciones de ataques entre reinas. El enfriamiento simulado resulta práctico para problemas en que la función de evaluación sea ligera en tiempo de computación. Recordemos que, algorítmicamente, el único competidor del AG aquí es la red de neuronas y se trata de una red pensada específicamente para este problema. Esta y otras características de los AGs, a pesar de sus requisitos computacionales, son las que los hacen muy prometedores en problemas complejos.

2.3.2. Evolución de Estructuras de Datos Complejas

Para demostrar la robustez que acabamos de sugerir presentamos en esta sección un resumen de trabajos, todos ellos propios, en los que mostramos cómo distintas versiones del algoritmo genético básico hacen evolucionar estructuras de datos complejas que codifican soluciones a cada uno de los cuatro problemas abordados. Nuestro objetivo final es reconocer los aspectos que mejorar para la obtención de algoritmos competitivos y eficientes. La sección queda dividida en subsecciones conteniendo, respectivamente, la descripción de los problemas, la metodología general de trabajo, las representaciones utilizadas, las funciones de evaluación desarrolladas y las conclusiones sobre tan variado conjunto de problemas.

2.3.2.1. Casos de Estudio

① Asignación Óptima de Procesos a Procesadores. [AAT94] [AAT95]

Dada una consulta sobre una base de datos lógica deductiva distribuida [CGT90] el objetivo es asignar sobre un anillo de procesadores los numerosos procesos comunicantes que resuelven dicha consulta atendiendo al modelo *Datalog* [Alda93]. La función objetivo que se pretende minimizar contempla los costes de realizar la comunicación de los numerosos flujos de datos entre los procesos y/o procesadores, así como el coste computacional de las operaciones que cada uno de estos procesos lleva a cabo. El genotipo es un vector de valores enteros.

② Diseño de Controladores que usan Lógica Borrosa. [ACT96] [CAT96] [ACT99]

A través del uso del paradigma de Programación Genética [Koza92] [Koza94], inicialmente desgajado de la familia de los algoritmos genéticos, pueden diseñarse controladores borrosos para problemas de control típicos como el de la carretilla móvil con brazo. La estructura de datos que evoluciona es un árbol sintáctico que representa un conjunto de reglas `IF-THEN` sobre variables de entrada/salida y conjuntos borrosos.

③ Validación de Protocolos de Comunicación. [AT96]

Con esta aplicación se pretende demostrar que realmente existe gran diversidad respecto a las estructuras de datos que pueden evolucionar. Nuestro objetivo es determinar la validez de un protocolo de comunicación aplicando un AG sobre cadenas de longitud variable que representan las trazas de ejecución del protocolo seguidas por un extremo cliente y otro servidor en la petición de un servicio de comunicación.

④ Entrenamiento de Redes de Neuronas. [AAT93a] [AC97]

Una vez fijada la estructura de la red de neuronas [RM89] (para lo cual se puede emplear también un algoritmo genético [WSB90] [AAT93b]) el problema consiste en encontrar los pesos que hacen a la red aprender los patrones de entrada (aprendizaje supervisado). Analizamos varias versiones de un AG: secuencial, hibridado con enfriamiento simulado (ES) y/o con propagación hacia atrás del error (PAE) y una versión distribuida en islas secuenciales de estado estacionario.

2.3.2.2. Metodología

Puede parecer que aplicaciones tan dispares como las presentadas requieren metodologías de trabajo diferentes. La realidad es que el enfoque empleado en todas ellas es el mismo, concentrándose las diferencias en los puntos finales del desarrollo.

Cuando se aborda la resolución de un problema mediante el empleo de técnicas evolutivas es necesario tener en cuenta el funcionamiento cualitativo de la búsqueda que se realizará. Dicho funcionamiento está caracterizado por la *teoría de los esquemas* [Holl75] cuya principal derivación es la *hipótesis de los bloques de construcción* [Gold89a] (presentados en la sección 2.7.1). Básicamente, debe considerarse que la búsqueda se realiza mediante la combinación de pequeñas unidades que representan porciones de una solución. Por consiguiente, es necesario determinar cuáles son dichas entidades y emplear una codificación del problema que permita la expresión de éstas. Adicionalmente, los operadores evolutivos deben permitir la propagación y yuxtaposición de los bloques fundamentales, lo que deberá ser tenido en cuenta a la hora de elegir tanto la codificación como los propios operadores.

Una vez definidos los aspectos anteriores, es crucial la construcción de la función de evaluación que medirá la calidad de cada solución. Obviamente, dicha función deberá incorporar aspectos propios del problema bajo resolución (lo que hemos definido como entorno E).

En resumen, la **metodología de trabajo** común que se sigue con estos algoritmos supone llevar a cabo los siguientes pasos:

- Determinar la función que medirá la adecuación de una cadena genética. Debe implementarse de manera eficiente porque representa el grueso de la computación.
- Determinar la mejor codificación para la cadena: valores binarios, codificación Gray, números flotantes, cadena o árbol de símbolos, etc... (Sección 2.7.1)
- Decidir los operadores que se van a utilizar. Por defecto, se consideran la selección de los mejores atendiendo a su adecuación, cruce y mutación. Adicionalmente, puede considerarse la incorporación de operadores que utilicen conocimiento específico sobre el problema que se desea resolver, lo cual ayuda a refinar el resultado final y disminuir el esfuerzo de evaluación, aunque suele aumentar el tiempo de ejecución.

2.3.2.3. Representaciones

En los problemas presentados anteriormente el objetivo principal ha sido determinar los puntos fuertes y débiles de los modelos secuenciales de AGs en dominios tan dispares. Para todos ellos se ha utilizado un modelo secuencial en estado estacionario que evoluciona estructuras de datos complejas (vea la Figura 2.6).

En el problema ① se han utilizado vectores de valores binarios de longitud fija codificando en cada gen el procesador en el que situar un proceso concreto de entre los que componen la búsqueda en la base de datos distribuida. El problema ② supone la evolución de árboles sintácticos que representan reglas de decisión borrosas encaminadas a llevar a una situación de equilibrio el brazo (inicialmente desequilibrado) por movimientos de un carro móvil.

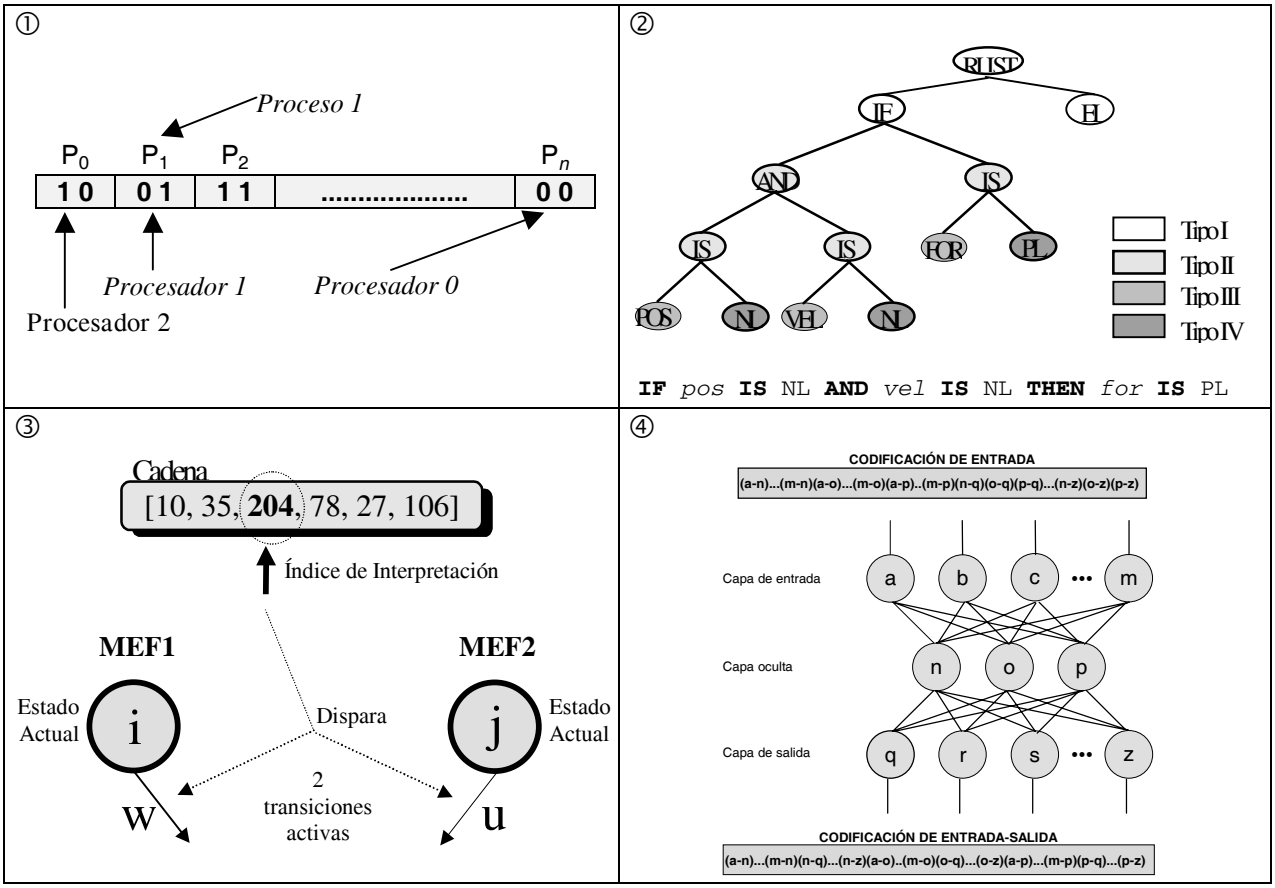


Figura 2.6. *Estructuras de datos complejas (G) usadas para demostrar que una plantilla de búsqueda tan genérica como la de un AG canónico no es sólo eficaz y competitiva con técnicas a la medida, sino también robusta ante problemas reales (no de laboratorio).*

El problema ③ utiliza como genotipo cadenas de longitud variable que representan trazas de ejecución (*protocol testing*) que permitan localizar posibles errores en el protocolo de comunicación. Por último, el problema ④ codifica en vectores reales los pesos de la red de neuronas de forma que el algoritmo busca la mejor combinación que minimiza el error de dicha red ante patrones de entrada para los que se conoce la salida deseada (aprendizaje supervisado).

2.3.2.4. Funciones de Evaluación

La Figura 2.7 muestra las funciones usadas en cada problema. En el caso del problema ① maximizamos la función complementaria respecto a una constante (*MEPV*: máximo coste esperado) en la que consideramos el coste de comunicación de la asignación representada por la cadena y el coste del desequilibrio introducido por dicha asignación en la carga computacional sobre los procesadores (pesando con sendas constantes su importancia relativa). En el problema ② minimizamos el número de pasos (tiempo simulado) hasta conseguir el equilibrio en el usando la base de reglas (SBR) representada por el árbol sintáctico codificado en la cadena.

En el problema ③ se maximiza una función que premia el descubrimiento de bloqueos, estados no visitados y transiciones no disparadas en la comunicación de las dos máquinas de estados finitos que ejecutan el protocolo de comunicación. En último lugar, el problema ④ minimiza el error cometido por una red de *n* neuronas de salida al serle presentado el conjunto de *p* patrones de entrenamiento.

① **ADECUACIÓN =** *MEPV* **- COSTE** **COSTE** = *Cw* * **Comm** + *Uw* * **Unb**	② **ADECUACIÓN**=*MAX_PASOS* – **COSTE** **COSTE** = **Pasos_Hasta_Equilibrio**
③ **ADECUACIÓN** = (*BL_ADEC* ***N_Bloqueos_Detectados**) + (*NV_ADEC**N_Estados_No_Visitados**) + (*ND_ADEC**N_Transiciones_No_Disparadas**)	④ **ADECUACIÓN =** *MEPV* **- ERROR** **ERROR** = $\sum_{i=1}^{p}\sum_{j=1}^{n}\left\|desada_j - actual_j\right\|$

Figura 2.7. *Funciones de evaluación Φ maximizadas en cada problema.*

2.3.2.5. Conclusiones

De la resolución de estos problemas se deducen varias conclusiones. En primer lugar, todos los problemas presentan una función de evaluación que es no lineal y bastante lenta en tiempo de computación. Esto indica que es imprescindible un algoritmo que minimice el esfuerzo de evaluación, ya que *cada evaluación* de un individuo supone la simulación de un sistema complejo: ① medir el coste de la asignación hecha de procesos paralelos, ② simular el sistema físico hasta encontrar el equilibrio usando el SBR definido por la cadena evaluada, ③ simular el protocolo para comprobar los posibles errores en que incurre y ④ evaluar el error de la red de neuronas para cada uno de los patrones del conjunto de aprendizaje.

Por estas razones confirmamos las ventajas de usar un AG frente a otras técnicas en los problemas presentados. Por ejemplo, en el caso del problema ④ nuestro estudio [AC97] demuestra que podemos ofrecer la ordenación de la Figura 2.8 atendiendo al error final, esfuerzo de evaluación y tiempo real crecientes.

En particular, mostramos en orden de prestaciones los resultados obtenidos por un algoritmo genético paralelo distribuido de islas estacionarias (dssGA), uno secuencial estacionario (ssGA), enfriamiento simulado (ES), enfriamiento simulado sobre una población generada usando selección proporcional (gES), propagación hacia atrás del error (PAE) y múltiples hibridaciones, incluyendo únicamente usar mutación sin cruce (MU) (véase [AC97] para más detalles).

Por tanto, comprobamos de nuevo las ventajas en cuanto a evitar óptimos locales de un AG frente a técnicas de gradiente descendente, incluso diseñadas para el problema en cuestión. Nótese la comparación entre algoritmos híbridos.

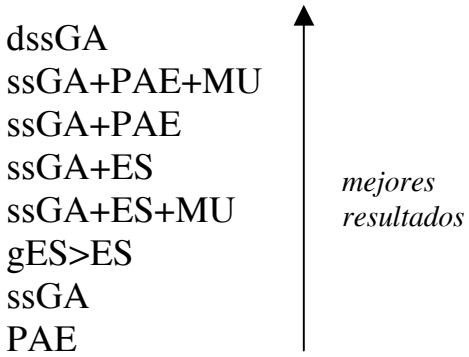


Figura 2.8. *Ordenación de técnicas en la resolución del problema ④.*

Por otro lado, todos los problemas demuestran el éxito y eficiencia en dominios complejos, como los de los problemas ① y ③, donde aún queda mucho por hacer. Incluso en el diseño de controladores borrosos podemos apreciar la flexibilidad de usar un AG en cuanto a la creación dinámica de conjuntos de pertenencia difusos.

El problema ② pone de manifiesto las ventajas de una solución usando estado estacionario frente a la tradicional generacional 1-elitista. Aunque el modelo estacionario y generacional son en el fondo derivables de un sistema adaptativo común y pueda definirse incluso un modelo generacional equivalente al estacionario, es un hecho que los modelos generacionales tradicionalmente usados son más lentos en converger a la misma solución que el modelo estacionario. Este hecho en el problema ② se presenta con una ventaja adicional en el refinamiento de la solución final y menor esfuerzo de evaluación (véanse también los resultados en el Capítulo 4).

En todas las aplicaciones mencionadas, sin embargo, queda también patente la necesidad de mejorar el tiempo de ejecución. Esta razón, junto a la necesidad de disminuir el esfuerzo de evaluación, nos conduce de forma natural al uso de algoritmos paralelos. Igualmente, retoma los trabajos de [Whit89], [Sysw91], [Juls95], [Levi97] en el sentido de comprobar que el modelo estacionario es en muchas ocasiones más eficiente que el generacional.

Puesto que hemos presentado los modelos secuenciales generacional y estacionario y una versión en islas estacionarias distribuidas, el siguiente paso natural consiste en formalizar el comportamiento celular. Con ello pretendemos conseguir un estudio preliminar completo de este modelo y unirnos a las líneas de creciente interés por la ejecución descentralizada, así como confirmar los excelentes resultados de los modelos celulares en problemas pensados para resultar difíciles a un algoritmo genético.

En la sección siguiente estudiamos el modelo celular y en la Sección 2.5 abordamos el modelo distribuido. Esta presentación se ha elegido debido a nuestra particular visión de los tres modelos (centralizados -secuenciales-, celular y distribuido) que forman el eje central de la discusión y desarrollo del presente trabajo.

2.4. Algoritmos Genéticos Celulares

La importancia de los algoritmos genéticos con estructura espacial está creciendo por varias razones. En primer lugar debido a las propiedades derivadas de su disposición bidimensional de los individuos. Esta característica le confiere al algoritmo propiedades interesantes respecto al mantenimiento de la diversidad, ajuste flexible de la presión selectiva y mayor eficacia [DS95], [SD97]. Además, existe un campo de trabajo interesante atendiendo a su paralelización en máquinas SIMD en las que se asigna un individuo por procesador (incluso existen implementaciones MIMD de ellos [HBBK97]) y también a su similitud con los autómatas celulares [Toma93] [Whit93b]. Vea el esquema típico del modelo celular frente al secuencial tradicional en la Figura 2.9.

En esta sección abordamos directamente el modelo de ejecución presente en un algoritmo genético celular en el que los individuos se sitúan en una rejilla (malla) toroidal de forma que cada uno de ellos interacciona únicamente con sus vecinos. Sin embargo, no implementaremos el modelo en máquinas SIMD, sino que usaremos el sistema de vecindarios sobre una población ejecutando todo en una misma máquina, aprovechando así las propiedades algorítmicas pero no las físicas.

De entre todos los tipos posibles de topología elegimos la rejilla y de entre todos los esquemas de vecindad posibles elegimos NEWS (*North-East-West-South*). Esto no supone pérdida alguna de generalidad puesto que, como se demuestra en [SD96], es posible con este esquema simular la presión selectiva de cualquier otro algoritmo, incluido el secuencial (Sección 2.7.3).

Figura 2.9. (a) *Algoritmo genético celular en una rejilla toroidal 2D y* (b) *una población en panmixia típica de los modelos secuenciales en los que no se define vecindario.*

Hemos mencionado la presión selectiva porque es el operador de selección, tanto de parejas como ambiental, el que principalmente se ve modificado en este modelo. Esto se debe a que en él abandonamos la definición en panmixia de la selección para redefinir su trabajo a nivel puramente de vecindario. El solapamiento de vecindarios es el principal sistema de transmisión de genotipos en el algoritmo (*difusión*) [CJ91].

En la Sección 2.4.2 definiremos formalmente la rejilla y haremos algunas hipótesis de trabajo que confirmaremos en el Capítulo 4. Por el momento, en la siguiente Sección (2.4.1) pretendemos mostrar una revisión de sus características más sobresalientes y una derivación que lo define a partir del modelo genérico de sistema adaptativo.

2.4.1. Derivación y Revisión

Para derivar un algoritmo genético celular del modelo granulado unificado únicamente debemos aclarar el funcionamiento de las operaciones de selección de vecinos y precisar que la sincronización de funcionamiento debe realizarse al final del cálculo de cada generación.

El hecho de que exista sincronización tras cada generación permite mantener a cada individuo de la topología en la misma generación en cualquier instante de la búsqueda. Los individuos generados residen en una población auxiliar hasta que se haya computado la generación completa. Después se reemplaza la generación antigua con la nueva como hace cualquier otro algoritmo evolutivo. El grano del paso evolutivo es pues la generación de μ individuos.

Aunque la sincronización sea necesaria sólo al final de cada generación (lo que no es poco en términos de una ejecución en computadores SIMD o MIMD, ya que restringe el paralelismo) sí que es necesario determinar y obtener en cada paso las estructuras vecinas de una dada residente en la población. Esto supone que una implementación masivamente paralela real en que cada estructura reside en un procesador genera una carga de comunicación muy importante al pedir copias de las estructuras designadas como vecinas (o al menos de sus adecuaciones). En el caso de una implementación MIMD el trasiego constante de vecinos entre los trozos distribuidos de la misma malla provoca una considerable merma de las prestaciones.

En nuestro caso, nos restringimos a utilizar el modelo celular pero ejecutándolo en un único procesador, de forma que consigamos sus ventajas algorítmicas aunque no una ejecución en tiempo real más rápida. Paradójicamente en muchos casos, incluso con un solo procesador, sí que se consiguen reducciones en el tiempo debido a que el esfuerzo de evaluación para resolver un problema es con frecuencia menor que en los modelos secuenciales [GW93].

La literatura [SD96] distingue entre vecindarios *lineales* o *compactos*. En los vecindarios lineales **L***r*, los vecinos de un punto dado incluyen a las *r*-1 estructuras más cercanas elegidas sobre los ejes horizontal y vertical (Figura 2.10). En el caso compacto **C***r*, el vecindario incluye a los *r*-1 individuos más cercanos. El tipo de elección puede dar lugar a vecindarios compactos *cuadrados* o *diamante*. Esta distinción inicial puede extenderse a vecindarios que *cambian* en el tiempo o en los que los individuos eligen su pareja tras un *paseo aleatorio* hasta ella [SD97].

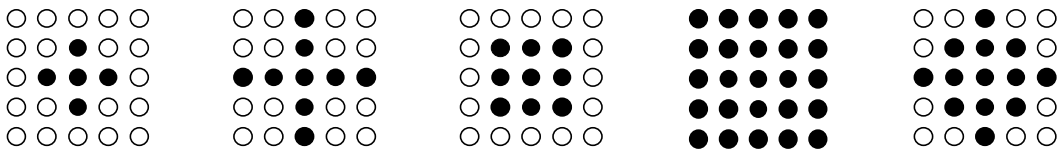


Figura 2.10. *Dos posibles tipos de vecindarios: lineales y compactos.*

Puede observarse en la Figura 2.10 que atendiendo al radio y a qué vecinos se incluyan distinguimos entre distintos vecindarios. Es de esperar que los vecindarios pequeños como LINEAL 5 sobrecarguen poco al sistema. En cuanto a los vecindarios grandes, es interesante observar, como ocurre con el vecindario COMPACTO 25 del ejemplo, que algunos pueden llegar a incluir a todos los elementos de la malla. En el caso extremo de vecindarios muy grandes y mallas pequeñas el efecto es similar al de usar una única población (panmixia).

Por estas razones parece claro que necesitamos caracterizar numéricamente los vecindarios y también la malla en sí para poder calibrar la relación entre uno y otro en un algoritmo celular. En particular, el vecindario LINEAL 5 es muy conocido y usado. También se le conoce con el nombre de NEWS y es el que vamos a utilizar debido a su sencillez teórica y también a la rapidez de la selección en vecindarios tan pequeños.

Como se deduce de su expresión formal (Definición 2.4) la única diferencia de un modelo celular respecto a otro generacional (por ejemplo) es que las operaciones se realizan sobre un vecindario de pocas (5) estructuras, necesitando así información adicional para calcular cada nueva estructura. Para vecindarios genéricos debemos: identificar las posiciones vecinas de la topología, pedirles una copia de sus individuos, seleccionar dos padres, cruzar, mutar y reemplazar la posición considerada. Esta idea intuitiva la mejoraremos en la Sección 2.6 en pseudocódigo y en el Capítulo 3 de manera más formal.

Por el momento los pasos expuestos en el párrafo anterior nos permiten comprobar que para trabajar con vecindarios genéricos necesitamos implementaciones eficientes para la población. Además, debemos reparar en que a las operaciones típicamente listadas sobre modelos celulares debemos añadir la de *identificar* cuáles son las posiciones vecinas. Esta operación no suele ser cara aunque algunos trabajos hacen de ella todo un arte [Balu93]. Podemos considerar un vecindario NEWS sobre una *lista* de individuos $A_i = \{\vec{a}_{i0},...,\vec{a}_{i\mu-1}\}$ donde cada estructura $\vec{a}_{ij}$ reside *virtualmente* en la posición (x,y) de una rejilla de tamaño $w \times h$ siendo:

$$X(j) := j \bmod w \qquad Y(j) := j \operatorname{div} w \qquad h := \mu / w \qquad j : 0..\mu - 1 \tag{2.5}$$

Inversamente, la estructura que reside en la posición (x,y) de la malla se encuentra almacenada en la lista de la población en la posición:

$$pos(x,y) := y \cdot w + x \tag{2.6}$$

y, por otro lado, el vecindario NEWS se define sobre dicha topología como sigue:

$$\upsilon(j) := news(j) := \{n, e, w', s\}$$

$$n := pos(X(j), --_{\text{mod}}(Y(j), h)) \qquad e := pos(++_{\text{mod}}(X(j), w), Y(j)) \tag{2.7}$$

$$w' := pos(--_{\text{mod}}(X(j), w), Y(j)) \qquad s := pos(X(j), ++_{\text{mod}}(Y(j), h))$$

Las funciones de *incremento y decremento acotado* tienen una definición simple:

$$++_{\text{mod}}(i,k) := (i+1) \bmod k \qquad --_{\text{mod}}(i,k) := i > 0 \,?\, (i-1) : k \tag{2.8}$$

Con esta definición de vecindario es posible caracterizar la presión selectiva de cualquier malla toroidal de dos dimensiones como se demuestra en [SD96].

Respecto a los demás componentes del sistema adaptativo celular no existen diferencias respecto a otros algoritmos. El algoritmo dispone de tantos gránulos como estructuras (μ). La estructura que un gránulo comparte con cualquier otro de entre los cuatro que pertenecen a su vecindario es siempre la única que reside en el gránulo: $\chi(B^i) = \{\vec{b}_i\}$.

La difusión de estructuras se lleva a cabo por el solapamiento de vecindarios, ya que cada una de las estructuras pertenece al vecindario de sus cuatro puntos adyacentes de la malla. De esta forma, la aparición de una solución en un punto de la malla se difunde a sus vecinos en el primer paso siguiente al de su descubrimiento y estos, a su vez, la difunden en pasos sucesivos, ya que la política de reemplazo suele requerir que la nueva estructura calculada por los operadores genéticos sea mejor que la considerada actualmente para reemplazar a ésta última. Este elitismo local (reemplazo por torneo binario) ha resultado muy útil en numerosos trabajos y presentaremos su funcionamiento relativo respecto a otras políticas en el capítulo de resultados.

En la siguiente sección caracterizamos cuantitativamente la rejilla y el vecindario atendiendo a su "radio". Esto permitirá futuros estudios sobre tipos de malla y vecindarios.

2.4.2. Caracterización Formal de la Rejilla

En esta sección definimos los parámetros que caracterizan la rejilla de un cGA. Nuestro trabajo es una extensión del trabajo en [DS95] y [SD96]. Definimos el concepto de *radio* (Ecuación 2.9), válido tanto para la topología de la rejilla como para el vecindario sobre ella. La idea básica es la de considerar a los n^* individuos implicados como puntos dispersos respecto a un centro común.

$$rad = \sqrt{\frac{\sum (x_i - \bar{x})^2 + \sum (y_i - \bar{y})^2}{n^*}} \qquad \bar{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*} \qquad \bar{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \tag{2.9}$$

Una alternativa sería usar el radio de un círculo que circunda a la rejilla (o vecindario), pero esto asignaría valores iguales a rejillas (vecindarios) distintos. Lo mismo ocurre si utilizásemos un coeficiente de asimetría. Por ejemplo, los vecindarios L5 y C9 (Figura 2.10) son distintos con la definición de radio de la Ecuación 2.9 (deseable), pero tendrían iguales valores de radio de círculo o de asimetría (indeseable).

El valor *rad* mide la dispersión de n^* puntos en un círculo centrado en $(\bar{x}, \bar{y})$. Para un número constante dado de individuos ($n=n^*$) el radio crece a medida que la rejilla se estrecha o a medida que el vecindario se extiende. Ya que el vecindario se mantiene constante en nuestros estudios, el ratio global (Ecuación 2.10) entre radio del vecindario y de la topología decrece al estrechar la rejilla (Figura 2.11).

$$ratio_{cGA} = \frac{rad_{vecindario}}{rad_{topología}} \tag{2.10}$$

Como veremos, la relación o ratio entre ambos radios influye en la búsqueda. Nuestro estudio se centra en considerar que es la forma de la rejilla la que más fácilmente puede modificarse para variar el comportamiento del algoritmo. Este punto de vista difiere de otros estudios que están enfocados en considerar variable el vecindario sobre una malla fija [Balu93] [SD97]. Nuestro acercamiento es mucho más simple de implementar y estudiar en la práctica.

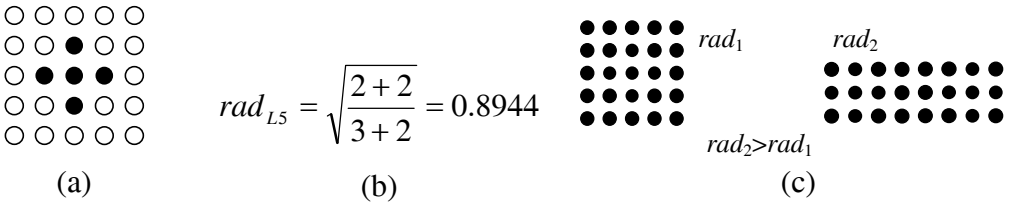


Figura 2.11. *Vecindario NEWS (conocido también como L5)* (a)*, su radio* (b)*, y un ejemplo de dos mallas con radios distintos para un mismo número de individuos* (c)*.*

La siguiente Figura (2.12) demuestra cómo, a medida que la rejilla se estrecha, el radio de la topología crece y por tanto la relación decrece proporcionalmente. Esta definición nos permite distinguir y estudiar modelos distintos de cGA de forma cuantitativa.

#	Topología	Radio	Ratio usando NEWS
a	1024x1024	4,180460501E+2	0,002139477217369000
b	2048 x 512	6,094017558E+2	0,001467668892463000
c	4096 x 256	1,184720431E+3	0,000754946041780600
d	8192 x 128	2,365115325E+3	0,000378163377720300
e	16384 x 64	4,729689472E+3	0,000189103323864000
f	32768 x 32	9,459311312E+3	0,000094552337955660
g	65536 x 16	1,891861418E+4	0,000047276190078740
h	131072 x 8	7,567445452E+4	0,000011819047863290
i	262144 x 4	1,513489090E+5	0,000005909523933205
j	524288 x 2	3,026978179E+5	0,000002954761967579
k	1048576x1	6,053956359E+5	0,000001477380983545

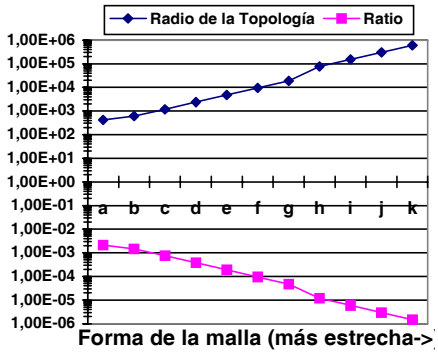


Figura 2.12. *Crecimiento (reducción) típico del radio de la topología (ratio) con NEWS.*

La presión de la selección en rejillas distintas con diferentes vecindarios pueden clasificarse en clases de equivalencia atendiendo a valores similares de ratii [SD96]. La cuantificación que acabamos de hacer es necesaria para estudiar las hipótesis de trabajo que delinearemos en este capítulo (Sección 2.7.3). Como adelanto, mencionaremos que es de esperar que la reducción de esta proporción suponga la reducción de la presión selectiva. Esto a su vez tiene un efecto doble. Por un lado incrementa el número de pasos necesarios para converger a una solución, y por otro mantiene durante más tiempo de ejecución del algoritmo la diversidad, mejorando la probabilidad de encontrar una solución óptima. El primer efecto es negativo y el segundo positivo; queda estudiar en qué condiciones el efecto positivo es el dominante.

2.5. Algoritmos Genéticos Distribuidos

En esta sección abordamos las principales políticas de trabajo que definen el comportamiento de un algoritmo genético paralelo distribuido. La principal motivación del uso de estos algoritmos es obtener ganancias en velocidad de ejecución a través del uso de islas débilmente acopladas. En las siguientes secciones concretamos múltiples mecanismos para la evolución de las islas (Sección 2.5.1), la topología de interconexión de las islas (Sección 2.5.2), la política de migración (Sección 2.5.3), los detalles sobre sincronización en la comunicación (Sección 2.5.4) y, finalmente, una visión global sobre heterogeneidad en la búsqueda y en la ejecución (Sección 2.5.5).

2.5.1. Evolución Básica de las Islas

En la mayoría de los trabajos con algoritmos genéticos distribuidos se utilizan islas cuya evolución básica es secuencial y en panmixia. Esto supone que cada isla ejecuta un número dado de generaciones antes de llegar a la fase de comunicación de individuos con su(s) vecino(s). Genéricamente, este modelo de evolución básico ha sido un AG secuencial generacional similar al usado en los trabajos de [Tane89], [Cant94], [ER96] o [CG97].

Los primeros pasos hacia algoritmos genéticos distribuidos con islas de estado estacionario podemos encontrarlos en los estudios con GENITOR II [WS90]. Sin embargo, la amplia disponibilidad de algoritmos generacionales ha condicionado la ejecución distribuida a islas generacionales. Nuestros resultados en [CAT96] en el diseño de controladores de lógica borrosa así como en [AC97], [CAT98a] y otros trabajos relacionados indican las ventajas que pueden obtenerse de una evolución en estado estacionario, tanto en dominios típicos de inteligencia artificial como en optimización funcional.

Tras considerar el modelo de ejecución de la isla con población única, el siguiente paso lógico es considerar la evolución de islas celulares. En primer lugar, esto se debe a que es difícil decidir entre las dos alternativas que plantean el uso de algoritmos celulares frente a distribuidos, ya que cada una tiene ventajas innegables y algunos inconvenientes. En segundo lugar, ya existen algunos indicios de que dicha combinación (conocida también como algoritmos genéticos paralelos híbridos [Cant97a]) proporciona muy buenos resultados (véase [Grua94], [RS94], [Pedr96]). Esta combinación, sin embargo, será caracterizada y estudiada como tal por primera vez aquí, ya que el resto de acercamientos están orientados a solucionar problemas muy concretos y no aportan ni estudios como clase de AGPs, ni indicaciones comparativas sobre su comportamiento sobre varios problemas.

Cualquiera de los modelos distribuidos mencionados puede obtenerse especificando únicamente el contenido del gránulo básico, bien instanciándolo a un trabajo en panmixia con cierto paso de evolución (Definición 1.2), o bien trabajando sobre un vecindario. Debe notarse que en el caso de islas de evolución celular cada gránulo del AG distribuido contiene un conjunto de μ gránulos interconectados entre sí situados en el mismo nodo. De esta forma, vemos cómo no es necesario ejecutar el algoritmo celular en una máquina SIMD, ya que sus ventajas algorítmicas provienen de utilizar una población estructurada bidimensionalmente.

2.5.2. Topología de Interconexión

En relación a la topología de interconexión entre las islas podemos distinguir dos características importantes en un algoritmo evolutivo paralelo distribuido en general:

- **Comportamiento temporal**, que determina la existencia de un vecindario *estático* o *dinámico*, según el conjunto de vecinos definido por la función ξ cambie o no en el tiempo. En realidad, es el comportamiento de la función componente υ quien determina esta característica al decidir directamente el vecindario.
- **Topología del vecindario**: *anillo* uni/bi-dimensional, *árbol*, *estrella*, *malla*, *totalmente conectado*, ... según la disposición de las islas determinada por la definición de ξ ($\xi \neq \varnothing$).

En general, incluso una conexión aleatoria dinámica es posible [MTS93]. Sin embargo, aunque no existen demasiados estudios destinados a decidir cuál es la mejor topología para un AGP distribuido, sí que parece existir un consenso respecto a que es útil usar un anillo o un hipercubo. Estudios detallados como los de [Cant94] sobre este tema usando DGENESIS arrojan resultados similares para estas dos topologías sobre un conjunto elevado de problemas.

Esta cuestión técnica es aún objeto de investigación. Por todo ello fijaremos nuestro campo de trabajo en una topología en **anillo unidireccional estática**:

$$\upsilon(i) = (i \bmod d) + 1 \quad i \in \{1,2,...,d\} \tag{2.11}$$

Para encontrar la razón de usar un anillo (además de su simplicidad y fácil implementación MIMD) podemos empezar por caracterizar el tiempo de comunicación. Dadas las constantes C_{comm} e γ (dependientes del sistema y topología del algoritmo) podemos derivar el tiempo de comunicación como proporcional al número de islas d. Suponiendo una ley de crecimiento exponencial en relación a la topología [CG97] tenemos:

$$T_{COMM} = C_{comm} \cdot d^{\gamma} \tag{2.12}$$

La ventaja de una topología en anillo es que cada migración entre cada par de vecinos puede realizarse en paralelo y en un tiempo constante (como ocurre en una red ATM por su velocidad y características de acceso al medio). Por tanto γ=0, y T_{COMM} son constantes.

Este resultado nos permite derivar teóricamente la afirmación de que la ganancia en velocidad (*speedup* S) a medida que utilicemos más procesadores será monótonamente creciente [CG97], aunque es de esperar que vaya siendo cada vez menos acusada (asíntota). Este resultado teórico explica nuestros estudios presentados en [CAT98a]. En general, cuando el acoplamiento entre islas es muy alto y la topología muy conectada el comportamiento depende claramente del tipo de red de comunicación usada. Esto da ventajas de nuevo al anillo porque no presenta una conexión excesiva, y es así más independiente del *hardware* que otras topologías.

2.5.3. Políticas de Migración

La política de migración permite determinar el tipo de interconexión que se está produciendo entre las islas del algoritmo distribuido. Una caracterización de este concepto debería englobar la mayoría de políticas existentes y permitir proponer nuevas ideas. Con este objetivo en mente definimos la política de migración como una tupla de cuatro valores:

Definición 2.5 (Política de Migración). *Una* política de migración *en un algoritmo genético paralelo distribuido queda definida por una tupla*:

$$\boldsymbol{M} = (m, \zeta, \omega_S, \omega_R) \tag{2.13}$$

donde:

- m : *es el* número de individuos *que migran*, $m \in \{0, 1, ...\}$ (*migration rate*).
- ζ : *es la* frecuencia de migración (*expresada en número de evaluaciones*), $\zeta \in \{0, 1, ...\}$.

- ω_S : *es la política de* selección de los individuos que migran*; tradicionalmente migra una copia del individuo(s) determinado(s) por la función χ, aunque también es posible extraer de la población el individuo que va a migrar (composición de selección y reemplazo).*
- ω_R : *es la* política de reemplazo, *usada para insertar cada inmigrante que llega a una población.* ■

Con esta definición de política de migración podemos caracterizar la transferencia de estructuras entre islas. Esto es necesario porque no basta con señalar qué islas serán vecinas.

Hemos definido la frecuencia de migración en términos de cuántas evaluaciones se realizan entre cada dos migraciones consecutivas. Alternativamente se podría interpretar como el número de generaciones entre dos migraciones; sin embargo, en nuestro caso trabajamos con algoritmos cuya generación es de distinta granularidad, y por tanto caracterizar a través de generaciones puede resultar cuando menos ambiguo.

De hecho, puede resultar especialmente cómodo utilizar valores como los que presentamos en [CAT98a] en donde el número de evaluaciones entre migraciones se caracteriza como un múltiplo (quizás potencia de 2) del tamaño de la población total: $1\cdot\mu$, $2\cdot\mu$, $4\cdot\mu$, $8\cdot\mu$, ..., indicando con 0 que las islas evolucionan de forma aislada (*partitioned evolution*). Esto permite que un mismo valor de frecuencia de migración represente distintos algoritmos en distintas aplicaciones. Así, evitamos dar recetas generales para problemas arbitrarios.

De esta forma, el operador de migración ω_M se define por la acción conjunta de un modelo para compartir estructuras y una política de migración como sigue:

$$\omega_{M\,\Theta_M}(\Delta_j) = \omega_R \circ \omega_S(\Delta_i, \Delta_j) \mid \ \forall \Delta_i, \Delta_j \in \Delta_{descent} \ , \tag{2.14}$$

además, el operador de selección determina en cada conjunto de estructuras aquéllas que pertenecen al gránulo origen Δ_i y que serán insertadas en el gránulo destino Δ_j:

$$\omega_S(\Delta_i, \Delta_j) = \left\{ \vec{b} \mid \vec{b} \in \xi(\Delta_i, \Delta_j) \right\}. \tag{2.15}$$

El número de individuos migrados se define como el número de estructuras compartidas en cualquiera de los gránulos (asumiendo que todas migren el mismo número):

$$m = \left|\xi(\Delta_i, \Delta_j)\right| \ , \tag{2.16}$$

y la probabilidad de aplicación de dicho operador es

$$p_M = \frac{1}{\zeta} \ . \tag{2.17}$$

Podemos dar una definición alternativa del operador de migración en el que el criterio de migración no viene regulado por los valores de una variable aleatoria discreta. Existen trabajos [MTS93] en los que el criterio de migración es más elaborado, como por ejemplo que los valores medios o la desviación típica de la adecuación de la isla satisfaga ciertos requisitos. De hecho, la mayoría de implementaciones suponen que formalmente existe una probabilidad de migración como la descrita en la Ecuación 2.17 pero realizan la migración de forma absolutamente determinista cada ζ evaluaciones. Esto es así porque de esta forma puede conseguirse una traza fiable de los valores que se miden.

Incluso algunos trabajos como [Beld95] justifican la sincronización entre islas cada paso de evolución, a pesar de que no haya migración, con el objetivo de obtener estadísticas exactas. Sin embargo, esto es únicamente aconsejable en casos puntuales cuando el objetivo es estudiar una ejecución aislada y no el comportamiento del algoritmo, ya que éste cambiaría sustancialmente como consecuencia de la constante sincronización.

En resumen, el conjunto de parámetros de la migración queda directamente determinado por los valores de la política de migración utilizada:

$$\Theta_M=\{M\} \tag{2.18}$$

y naturalmente depende del mecanismo que define cómo compartir estructuras y la política de activación paralela. El ciclo reproductor de un AGP distribuido consiste en la composición del operador de ciclo reproductor de la isla más el de migración:

$$\omega_d = \omega_M \circ \omega_{isla} \tag{2.19}$$

Es posible definir un parámetro adicional importante en relación a cómo se implementa la migración. Nos referimos al trabajo asíncrono de las islas en lugar de una evolución síncrona como la supuesta hasta ahora. La siguiente sección profundiza en este aspecto.

2.5.4. Sincronización

La mayoría de los algoritmos evolutivos distribuidos existentes son síncronos. Esto significa que la fase de comunicación, entendida como fase de envío y recepción de estructuras, está localizada en la misma porción del algoritmo evolutivo distribuido. En este sentido, tanto la implementación como la descripción y el estudio formal se simplifican al asegurarse que todas las islas se encuentran en el mismo estado de evolución en un momento dado.

Por el contrario, el funcionamiento *asíncrono* suele resultar más eficiente en la práctica [Gorg89] [MTS93] [ZK93] [HBBK97]. Su desventaja es una implementación más complicada y también los problemas que surgen de separar la fase de envío de la de recepción de individuos. Ya que una isla puede recibir en cualquier instante de su evolución un individuo cuando trabaja en modo asíncrono, las islas emisoras y receptoras pueden encontrarse en distintas generaciones y sus individuos en diferentes estados de evolución.

Básicamente, la diferencia radica en la interpretación del operador de migración (véase la Figura 2.13). En la migración síncrona se realiza el envío de emigrantes sin espera alguna (línea discontinua) e inmediatamente después el algoritmo se bloquea (línea continua) hasta obtener los inmigrantes de las islas vecinas. Sin embargo, en el caso asíncrono se realiza el envío de emigrantes atendiendo a la misma probabilidad p_M pero no se bloquea el algoritmo en espera de estructuras de entrada. Por el contrario, cada paso de la evolución decide si existen individuos esperando y si es así los inserta de acuerdo a la política de reemplazo usada.

Esto implica que el intercambio de estructuras puede comportarse en la población destino (1) como un *superindividuo*, o (2) como un generador de descendencia *letal*. Un superindividuo es aquel cuya adecuación es considerablemente superior al de la media de la población. Asimismo definimos un individuo letal como aquel individuo de menor adecuación que sus progenitores.

Ambos tipos de individuos son, en teoría, perjudiciales porque tienden a redireccionar la búsqueda en la isla receptora (efecto de "conquista" en el caso del superindividuo) o a malgastar recursos de computación generando puntos (letales o "no-efecto") de especies intermedias lejanas a las del emisor y el receptor [LPG94] [Mare94].

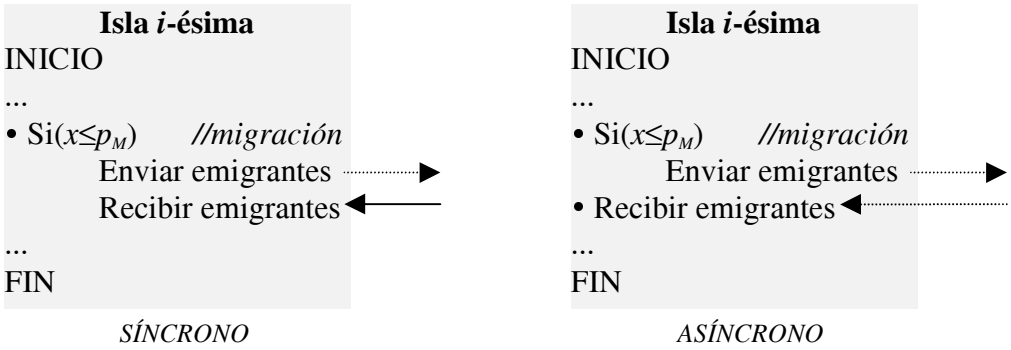


Figura 2.13. *Diferencias entre evolución síncrona y asíncrona.*

En la práctica, estos efectos son apenas acusados si la búsqueda a nivel de ejecución e implementación (véase de nuevo la Figura 1.1) es homogénea, ya que las islas progresan aplicando las mismas operaciones sobre estructuras similares al mismo tiempo (en promedio) en cada procesador. De hecho, esta solución asíncrona presenta sus ventajas respecto a ejecución más rápida sin las desventajas mencionadas en el párrafo anterior. Asimismo, permite hacer las mismas suposiciones que en el caso síncrono en término medio. En el caso heterogéneo, los resultados a priori son impredecibles, pues dependen del problema y algoritmo usados. En realidad, esta cuestión está abierta como rama de investigación.

2.5.5. Homogeneidad

La caracterización del algoritmo como homogéneo o heterogéneo a nivel de búsqueda (ejecución) viene determinada por el ciclo reproductor que se utilice en cada gránulo. De esta forma, si el ciclo reproductor es el mismo en todos los gránulos, el algoritmo distribuido se dice homogéneo a nivel de búsqueda. Esto implica que, el tipo, número y parámetros de los operadores es el mismo. Igualmente, el genotipo usado también debe ser el mismo. En otro caso estaremos tratando sobre un algoritmo distribuido heterogéneo (como ocurre en GAMAS [PGY94] por ejemplo).

Aunque ambos tipos de heterogeneidad son raros, el uso de ciclos reproductores distintos está tomando auge debido a que permite aprovechar las ventajas relativas de cada uno de los ciclos usados, virtualmente a "coste 0" (exclusivamente la complejidad de la implementación, ejecución y muchas veces su estudio formal). En este aspecto, resultados como GDGA [HL97] o CoPDEB [AP96] son esperanzadores en problemas difíciles de optimización funcional o diseño de redes de neuronas. Véase la Figura 2.14 para comprobar varias formas de introducir heterogeneidad en la búsqueda, bien disponiendo explícitamente planos de exploración y explotación como en GDGA, o bien usando distintos operadores y probabilidades como en CoPDEB.

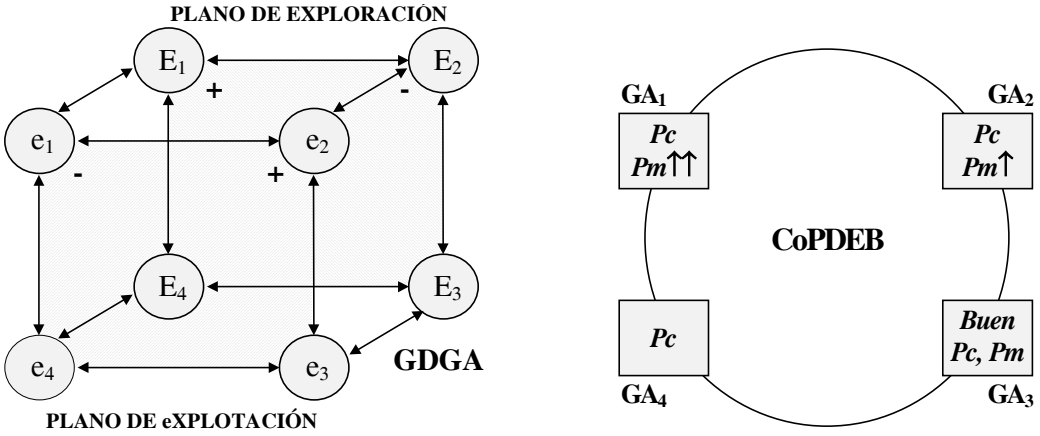


Figura 2.14. *GDGA* (izquierda) *y CoPDEB* (derecha), dos AGs distribuidos heterogéneos.

La heterogeneidad *hardware* es prácticamente nula en este campo. Aunque algunos modelos están implementados de forma genérica para ejecutarse sobre plataformas distintas tras ser recompilados (GALOPPS, GENESIS, DGENESIS, ...) no existen resultados hasta el momento en que se ejecuten *a la vez* sobre plataformas *hardware* distintas.

En el sentido anterior, quizás las implementaciones de modelos distribuidos en JAVA sean interesantes por permitir de forma intrínseca esta característica. De todas formas no está nada claro que resulte ventajoso usar distintos tipos de *hardware* para la ejecución de un AEP, más que aprovechar instalaciones existentes.

Una vez concluida la descripción unificada de algoritmos secuenciales y paralelos pasamos en la siguiente sección a presentar una aclaración del funcionamiento canónico de los modelos secuenciales, celulares y su paralelización distribuida. En este trabajo usamos algoritmos homogéneos tanto a nivel de búsqueda como a nivel de ejecución.

2.6. Discusión y Aclaraciones usando el Pseudocódigo

En esta sección presentamos un pseudocódigo de los algoritmos centralizados (generacional y de estado estacionario), así como una versión básica de un modelo celular y una versión distribuida síncrona que puede realizarse de cualquiera de ellos (véase la Figura 2.15).

Es nuestra intención proporcionar una idea intuitiva para concretar el uso de las definiciones formales anteriores e introducir los modelos básicos de los algoritmos que vamos a diseñar y analizar en detalle en los siguientes capítulos.

Con esta definición en pseudocódigo es más fácil comprender la problemática de una descripción e implementación unificadas, así como derivar su comportamiento y puntos de estudio clave. Mostramos versiones simples suponiendo que usamos selección, cruce, mutación y reemplazo. A partir de ellas es fácil obtener algoritmos más elaborados que incorporen operadores novedosos o dependientes del problema, así como variar el tipo, número y parámetros de los operadores.

```
[genGA] proc Ciclo_Reproductor(ag):
          para s←1 hasta MAX_PASOS hacer
                lista_p←Seleccionar(ag.pob);                    // Seleccionar TAM_POB cadenas
                para i←1 hasta TAM_POB/2 hacer
                      Cruzar(ag.Pc, lista_p[i],lista_p[2*i],ind_aux.crom);
                      Mutar(ag.Pm,ind_aux.crom);
                      ind_aux.adecuación←ag.Evaluar(Decodificar(ind_aux.crom));
                      Insertar_Nuevo_Ind(pob_aux,ind_aux,);
                fin_para;
                ag.pob←[elitista|no_elitista]←pob_aux;
                Recolectar_Estadísticas(ag);
          fin_para;
       fin_proc Ciclo_Reproductor;

[ssGA]  proc Ciclo_Reproductor(ag):
          para s←1 hasta MAX_PASOS hacer
                padre1←Seleccionar(ag.pob);
                padre2←Seleccionar(ag.pob);
                Cruzar(ag.Pc,padre1,padre2,ind_aux.crom);
                Mutar(ag.Pm,ind_aux.crom);
                ind_aux.adecuación←ag.Evaluar(Decodificar(ind_aux.crom));
                Insertar_Nuevo_Ind(ag,ind_aux,[si_mejor|siempre]);
                Recolectar_Estadísticas(ag);
          fin_para;
       fin_proc Ciclo_Reproductor;

[cGA]   proc Ciclo_Reproductor(ag):
          para s←1 hasta MAX_PASOS hacer
                para x←1 hasta ANCHURA hacer
                     para y←1 to ALTURA hacer
                        lista_v←Calcular_Vecinos(ag,posición(x,y));
                        padre1←Seleccionar(lista_v);
                        padre2←Seleccionar(lista_v);
                        Cruzar(ag.Pc, lista_v[padre1],lista_v[padre2],ind_aux.crom);
                        Mutar(ag.Pm,ind_aux.crom);
                        ind_aux.adecuación←ag.Evaluar(Decodificar(ind_aux.crom));
                        Insertar_Nuevo_Ind(posición(x,y),ind_aux,[si_mejor|siempre],ag,pob_aux);
                     fin_para;
                fin_para;
                ag.pop←pob_aux;
                Recolectar_Estadísticas(ag);
          fin_para;
       fin_proc Ciclo_Reproductor;

[dGA]   Arrancar_Sistema(ag);
       Generar_Subpoblaciones(ag);
       Evaluar_Subpoblaciones(ag);
       Recolectar_Estadísticas_Globales(ag);

       para s←1 hasta MAX_PASOS hacer
                para i←1 hasta NÚMERO_DE_ISLAS hacer
                      Ciclo_Reproductor(ag[i]);
                fin_para;
                si Debe_Migrar(ag,s) entonces
                      Migrar_En_Anillo(ag,[mejor|aleatorio]);          // ¿Quién emigra?
                      Recolectar_Estadísticas_Globales(ag);
                fin_si;
       fin_para;

       Solución←Mejor_Ind_Encontrado_Durante_La_Evolución;             // ¡En todos los casos!
```

Figura 2.15. *Pseudocódigo de los dos modelos de población única, en difusión y distribuido.*

El modelo generacional (genGA) secuencial genera un conjunto de estructuras para reproducción usando el operador de selección. Tras cruzarlas y mutarlas reemplaza la población antigua por la nueva. Debe notarse que ofrecemos la posibilidad de usar (o no) elitismo para que la mejor cadena sobreviva (o no) entre dos generaciones consecutivas de forma determinista. El modelo de estado estacionario (ssGA) genera en cada paso una cadena nueva aplicando selección, cruce y mutación y la inserta en la población junto con sus padres desplazando a la peor cadena existente, bien siempre, o bien sólo si es mejor que la peor cadena existente.

El modelo celular (cGA) ofrece un ciclo reproductor similar, pero la selección se realiza sobre los vecinos únicamente. Se genera una población temporal cuyas cadenas reemplazarán una a una a las de la población antigua siempre o únicamente si son mejores.

Nótese que en la versión distribuida suponemos ya que estamos usando un anillo de islas (ya justificamos por qué en la Sección 2.5.2). El modelo distribuido tiene una fase de inicio más elaborada que los modelos no distribuidos al tener que generar las islas (y sus poblaciones iniciales), controlar la migración y tomar información del sistema de forma más sofisticada. Observe que el modelo distribuido en particular deja abierta la posibilidad de que todas las islas sean o no de comportamiento secuencial y/o celular. Por tanto, no se restringe la heterogeneidad en la búsqueda (ni en la ejecución).

Pretendemos destacar en el pseudocódigo todos los componentes *reales* de estos algoritmos, incluyendo el cálculo de estadísticas, permitiendo así el seguimiento del comportamiento de cualquiera de los algoritmos.

En todos los algoritmos la solución es aquel individuo con la mayor adecuación encontrado durante todo el proceso de evolución. Aunque el criterio de terminación mostrado es completar un cierto número de pasos otro posible criterio es encontrar una solución (si se conoce de antemano). Evidentemente hemos hecho algunas elecciones concretas respecto a las definiciones formales genéricas de las secciones anteriores. En particular se muestra un cruce que devuelve un único hijo para mantener la eficiencia en dominios complejos, una rejilla en el modelo de difusión y una versión síncrona del modelo distribuido. El resto de posibilidades teóricas están aún presentes en ellos.

2.7. Fundamentos

En esta sección pretendemos recoger y estudiar algunos de los más importantes desarrollos teóricos existentes que justifican y caracterizan formalmente el comportamiento de los modelos secuenciales (Sección 2.7.1) y paralelos (Sección 2.7.2) de algoritmos genéticos. Igualmente propondremos a partir de ellos algunas suposiciones de trabajo y presentaremos nuestros resultados sobre la presión selectiva sobre la población (Sección 2.7.3).

2.7.1. Fundamentos de los Algoritmos Genéticos Secuenciales

El primer intento teórico de analizar el funcionamiento de los algoritmos genéticos lo hizo Holland y puede encontrarse en [Holl75] y [Gold89a]. Básicamente explica el trabajo de la búsqueda genética a través del estudio de esquemas presentes de manera implícita en una población de tamaño finito constante de cadenas binarias sometidas al operador de cruce de un punto y mutación (Capítulo 1). Todo ello, por supuesto, suponiendo una población única en panmixia y una selección proporcional a la adecuación con reemplazo generacional (μ,λ).

Supongamos, sin pérdida de generalidad, que construimos las cadenas sobre un alfabeto binario $\mathbf{V}=\mathbb{B}=\{0,1\}$; una población de cadenas en la generación t, $P(t)$, representa un conjunto de puntos o instancias de lo que daremos en llamar **esquemas** en un espacio l-dimensional, donde l es el tamaño de las cadenas-individuo.

Un esquema H es una cadena sobre un alfabeto con 3 elementos V^+={0,1,*}. Las posiciones que no contienen asterisco en el esquema se denominan *definidas*; un asterisco en una posición de un esquema hace de ella una posición *indefinida*. Decimos que un individuo es una instancia de un esquema si ocurre que: *(a)* donde quiera que el esquema esté definido el individuo lo está y además el valor de ambos coincide y *(b)* dada una posición con asterisco en el esquema, el valor que el individuo pueda tener en dicha posición es indiferente para decidir si el individuo es, o no, una instancia del esquema. Así, por ejemplo, la cadena S=[0111000] de longitud l=7 es una instancia del esquema H=[*11*0**] ya que sus valores coinciden allí donde H está definido.

Para una cadena de longitud l existen hasta 3^l esquemas posibles; en general, para alfabetos de v elementos, existen $(v+1)^l$ esquemas. Una cadena dada es una instancia de hasta 2^l esquemas y por tanto una población de μ individuos representa a un máximo de $\mu \cdot 2^l$ esquemas.

Un algoritmo genético trabaja sobre un elevado número de esquemas distintos, algunos más *definidos* (contienen menos *) que otros. Así, usaremos las funciones *orden de un esquema*, $o(H)$, como el número de posiciones definidas del esquema H, y *longitud de definición*, $\delta(H)$, como la diferencia entre la última y la primera posición definidas del esquema H.

- Por ejemplo, para $H1$=011*1** y $H2$=0******, tenemos que sus órdenes respectivos son $o(H1)$=4 y $o(H2)$=1, y que sus longitudes de definición son $\delta(H1)$=5-1=4 y $\delta(H2)$=1-1=0.

Los esquemas son *plantillas* (poco especificadas) y los individuos de una población son *instancias* de dichos esquemas, es decir, cadenas completamente definidas de ellos. Un AG, aunque maneja instancias, está trabajando realmente con los esquemas, combinando los mejores trozos de cadena (posiciones definidas de un esquema) en la búsqueda de mejores soluciones.

Supongamos que existen μ cadenas, S_j, con j:{0,1,...μ-1}, en una población $P(t)$ que está en su generación t. Supongamos que existen m instancias-cadena de un esquema particular H; así $m=m(H,t)$. En resumen, si tenemos en cuenta los efectos combinados de la selección, cruce, mutación y además despreciamos los términos producto resultantes que sean muy pequeños, tenemos el llamado **teorema de los esquemas** o t*eorema fundamental*:

$$m(H,t+1) \geq m(H,t) \cdot \frac{\Phi(H)}{\overline{\Phi}} \cdot \left\{ 1 - \left(p_c \cdot \frac{\delta(H)}{l-1} \right) - o(H) \cdot p_m \right\} \qquad (2.20)$$

que podemos entender como que los esquemas cortos, de bajo orden de definición y cuya adecuación está por encima de la media (llamados *Bloques de Construcción -Building Blocks* o BBs-) son los que reciben un número incrementado exponencialmente de instancias; además, el algoritmo hace este trabajo de forma indirecta y paralela para muchos esquemas.

Holland [Holl75] estimó que cuando un AG trabaja sobre μ cadenas durante una generación, realmente está trabajando con $O(\mu^3)$ esquemas. A este resultado se le conoce como *paralelismo implícito* y no tiene repercusiones en el tiempo o memoria computacional necesitados para su ejecución; es un efecto lateral del modo de funcionamiento de un AG.

Existe una interpretación geométrica (Figura 2.16) de los esquemas en los que se les hace corresponder hiperplanos en un espacio l-dimensional, donde l es la longitud de los individuos. Los individuos representan puntos de dicho espacio. Un AG cruza planos entre sí creando otros nuevos en la búsqueda del mejor punto.

De esta forma vemos cómo los fundamentos teóricos tradicionales de los algoritmos genéticos están basados en la noción de la reproducción selectiva y la recombinación de cadenas binarias. Estos van cambiando la tasa de instanciación de los hiperplanos que configuran el espacio de búsqueda para así reflejar la adecuación media de las cadenas que residen en cualquier hiperplano particular. Por tanto, durante la recombinación, las cadenas están intercambiando información de sus hiperplanos. Esta es la razón de por qué los algoritmos genéticos no necesitan buscar a lo largo de los contornos de la función que está siendo optimizada (es por tanto independiente de la función objetivo) y por qué no tienden a quedar atrapados en mínimos locales (realizan búsqueda paralela desde múltiples puntos diferentes).

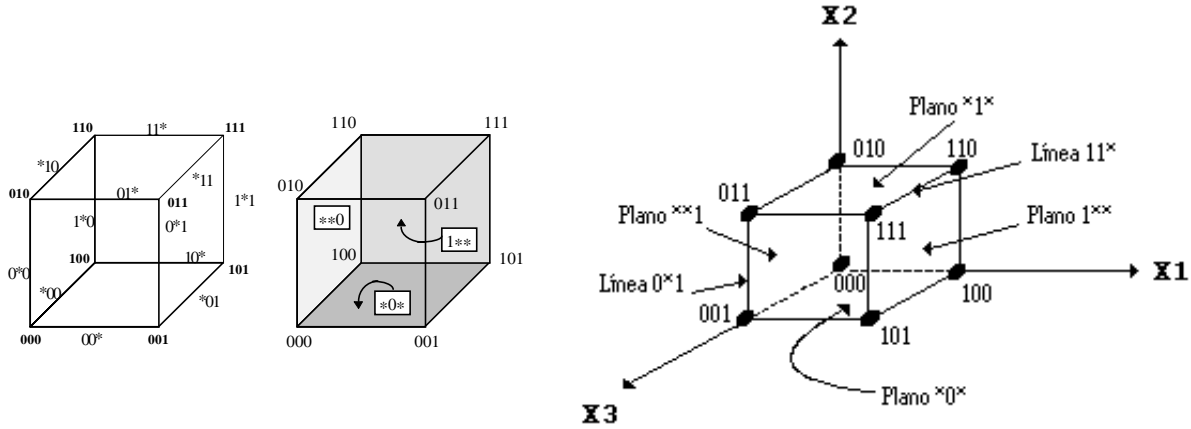


Figura 2.16. *Visualización de los esquemas como hiperplanos en un espacio tridimensional (l=*3*). Los puntos, líneas y planos son esquemas de orden* 3*,* 2 *y* 1*, respectivamente.*

2.7.1.1. Nueva Interpretación del Concepto de Esquema

En esta sección pretendemos presentar la nueva forma de entender qué es un esquema. Este nuevo concepto proporciona las bases para explicar el hecho de que múltiples aplicaciones que no trabajan sobre cadenas binarias tengan tango éxito. Una de las principales aportaciones a este terreno la hizo Antonisse [Anto89] centrando sus resultados en la interpretación respecto al símbolo comodín '*' (asterisco).

Aunque el principio del alfabeto mínimo ([Holl75], posteriormente más detallado en [Gold89a]) fue inicialmente potenciado y tomado casi como precondición para el uso de AGs, actualmente existen numerosas teorías derivadas y aplicaciones reales que demuestran la superioridad en múltiples dominios de otro tipo de alfabetos de aridad alta. Las razones para el uso de cadenas binarias han sido tradicionalmente la simplicidad de análisis de vectores de bits, la elegancia de los operadores genéticos y los requisitos de velocidad computacional. Sin embargo, la razón más importante para su uso ha sido la hipotética ventaja en relación al paralelismo implícito y a la maximización del número de esquemas manipulados. Esto parece contradecir la idea de que a mayor expresividad del alfabeto mayor es la potencia del algoritmo.

En la interpretación original que acabamos de presentar en la sección anterior, se extiende el alfabeto usado con un símbolo más llamado comodín o símbolo de "no importa". Inicialmente, este símbolo estaba destinado a expresar una cadena (esquema) sobre un alfabeto extendido que fuese la representante de un conjunto de cadenas sobre el alfabeto original.

Por ejemplo, si tenemos cadenas binarias de longitud 20 y cadenas con símbolos decimales con longitud 6, a igualdad de potencia expresiva para ambos alfabetos [binario(l=20)=$2^{20}\approx 1{,}05\cdot 10^6$ y decimal(l=6)=10^6] según la interpretación tradicional el alfabeto binario produce $(2+1)^{20}=3{,}48\cdot 10^9$ esquemas posibles de los que cada individuo representa como máximo a 2^{20}, mientras que en el caso decimal existen únicamente $(10+1)^6=1{,}77\cdot 10^6$ esquemas posibles de los que un individuo contiene sólo 2^6.

Si al trabajar sobre cada cadena lo hacemos en paralelo sobre los esquemas que indirectamente contiene, entonces parece deducirse que sería óptimo el alfabeto binario porque permite trabajar sobre el máximo número de esquemas posible.

Pero si volvemos a la definición original de esquemas que hizo Holland veremos que * se utilizó para indicar "no importa". Así, cualquier cadena sobre el alfabeto extendido representa clases de cadenas sobre el alfabeto binario. Sin embargo, el objetivo inicial de usar un símbolo comodín (*) era "cuantificar" cadenas con diferentes valores en su posición. En binario * significa "0 o 1", y 000* cuantifica a 0000 y 0001. Luego * se ha interpretado tradicionalmente como "no importa". Pero podemos ver que el caso binario es ambiguo porque es exactamente lo mismo "no importa" qué símbolo aparezca que "*cuantificar* el número de cadenas con valores distintos para una posición dada".

Aquí está el error tradicional que vetó teóricamente durante un tiempo el uso de otros alfabetos a pesar de que daban buenos resultados. En alfabetos de aridad mayor que 2 la interpretación de "*no importa*" y de "*cuantificación*" no son equivalentes.

Por ejemplo, para cadenas de longitud 4 sobre un alfabeto de 3 símbolos {0, 1, 2}, la interpretación tradicional de "no importa" especifica que 000* representa a tres cadenas {0000, 0001, 0002}. Sin embargo el "no importa" según la interpretación de "cuantificar" es realmente "0 o 1, 0 o 2, 1 o 2, o bien 0 o 1 o 2". Esto lleva a concluir que 000* representa a los conjuntos de cadenas {0000, 0001}, {0001, 0002}, {0000, 0002} y {0000, 0001, 0002}.

De esta manera, en la nueva interpretación vemos que necesitamos nombrar el símbolo comodín porque para alfabetos de aridad mayor que 2 ya no es único. Por tanto, el comodín tradicional * es realmente $*_{01}$ en la nueva interpretación para cadenas binarias. Para un alfabeto de 3 símbolos {0, 1, 2} existen realmente 4 símbolos comodín con los que debemos aumentar el alfabeto original para generar esquemas: $*_{01}$, $*_{02}$, $*_{12}$, $*_{012}$. En resumen, podemos tabular los resultados de ambas interpretaciones para hacer patente las diferencias respecto al número total de esquemas y al número de esquemas por individuo en cada caso (Tabla 2.2).

El hecho de que exista justificación teórica para el uso de alfabetos de cualquier aridad no implica que sean útiles o los mejores para cualquier problema. En particular, hay dominios en los que no todas las combinaciones de parámetros binarios están permitidas, en cuyo caso los alfabetos de alta aridad están aconsejados.

Para finalizar, es importante resaltar los recientes resultados de [FG97] que demuestran formalmente que no pueden suponerse ventajas intrínsecas a ninguna elección del alfabeto sobre el que se construyen las cadenas. Incluso tampoco puede atribuirse a priori ninguna ventaja a ningún operador de variación unario o binario sobre otro, ya que es posible construir algoritmos funcionalmente equivalentes en los casos mencionados de distinta representación u operadores.

TABLA 2.2. COMPARATIVA ENTRE LAS INTERPRETACIONES TRADICIONAL Y NUEVA DEL CONCEPTO DE ESQUEMA Y SUS IMPLICACIONES PRÁCTICAS

Interpretación	Número de Símbolos del Alfab. Ampliado	Número de Esquemas Totales	Número de Esquemas Presentes en un Individuo	EJEMPLO 1: v=2 (aridad) / l=8 v=3 (aridad) / l=5	EJEMPLO 2: v=2 (aridad) / l=20 v=10 (aridad) / l=6
Tradicional	$v+1$	$(v+1)^l$	2^l	TOTALES $(2+1)^8=6561$ >> $(3+1)^5=1024$ POR INDIVIDUO $2^8=256$ >> $2^5=32$	TOTALES $(2+1)^{20}=3.48\cdot10^9$ >> $(10+1)^6=1.77\cdot10^6$ POR INDIVIDUO $2^{20}=1.05\cdot10^6$ >> $2^6=64$
Nueva	2^v-1	$(2^v-1)^l$	$(2^{v-1})^l$	TOTALES $(2^2-1)^8=6561$ << $(2^3-1)^5=16807$ POR INDIVIDUO $(2^1)^8=256$ << $(2^2)^5=1024$	TOTALES $(2^2-1)^{20}=3.48\cdot10^9$ << $(2^{10}-1)^6=1.46\cdot10^{18}$ POR INDIVIDUO $(2^1)^{20}=2^{20}$ << $(2^9)^6=2^{54}$

2.7.1.2. Codificación Real

Una de las alternativas más importantes a la codificación binaria es la codificación real. Este tipo de codificación parece particularmente útil y natural para la búsqueda en espacios con parámetros continuos. Cada variable del problema se representa como un gen real en la cadena genética y se supone que los operadores deben preservar el rango de valores de cada variable.

Los trabajos con AGs de codificación real (FPGAs) se iniciaron a finales de los 80 con trabajos en quimiometría y sobre todo en problemas de optimización numérica sobre dominios continuos [LK89] [Davi91a] [JM91] [Eshe93]. Aunque inicialmente la teoría básica no sustentaba su aparente éxito las aplicaciones proliferaron hasta que se diseñaron herramientas teóricas como las mostradas anteriormente, que permitían corroborar el buen comportamiento de los FPGAs [Gold91] [Wrig91].

Como consecuencia del uso de genes reales han aparecido algunos conceptos nuevos sobre todo en relación con la *expresión* del genotipo para dar lugar al fenotipo. Notablemente destacan los modelos no isomórficos en los que diferentes genotipos pueden dar lugar al mismo fenotipo (no hay correspondencia uno a uno entre ambos). En [VBS93] se describe una codificación según la cual cada variable del problema tiene asociado una precisión distinta. En [Gold89a] ya se describía un concepto similar, así como la posibilidad de que cada variable pertenezca a un subrango del dominio real distinto.

La codificación real es uno de los pilares básicos de las estrategias de evolución desarrolladas en los años sesenta en Alemania [Rech73] [BHS91] [Schw95]. Posteriormente han surgido una enorme cantidad de trabajos sobre algoritmos y operadores para genotipos reales [YHK97] entre los que destacan nuevos operadores de cruce borrosos como los descritos en [HLV95].

El uso de genes reales está actualmente tomando auge sobre todo en el terreno de la optimización funcional, aunque arrancó también con fuerza con los primeros estudios sobre diseño de redes de neuronas usando algoritmos genéticos. Tradicionalmente se les ha prestado poca atención a los genotipos reales en algoritmos genéticos, muy al contrario de lo que ha ocurrido con las estrategias evolutivas desde su nacimiento, en las que se potencia su uso.

Cuando se desea optimizar una función de p parámetros continuos podemos utilizar una codificación binaria pura de q bits por parámetro, una codificación Gray de q bits (con la ventaja de poder alcanzar cualquier punto del espacio con cambios de un sólo bit cada vez sin los acantilados típicos de la codificación binaria pura) o bien una codificación de p parámetros en coma flotante, aprovechando las facilidades de los lenguajes de programación actuales.

En [Salo96] podemos encontrar la demostración de los límites superiores a la complejidad computacional de cada tipo de genotipo (binario, Gray y flotante) suponiendo alta probabilidad de cruce y baja de mutación (p_m=1/l para genes binarios o Gray y p_m=1/p para genes reales).

Podemos resumir estos resultados como sigue. Cuando la optimización de cada uno de los p parámetros es independiente la complejidad algorítmica es $O(l \cdot \ln l)$ o bien O $(p \cdot \ln p)$ que realmente son la misma porque sólo se diferencian en un valor constante ya que $l=q \cdot p$. En caso de que la complejidad de optimización de un parámetro x_i dependa de p entonces la complejidad resultante es el producto de $O(p \cdot \ln p)$ por la complejidad de optimizar dicho parámetro. Por ejemplo si la complejidad de optimizar x_i fuese $O(p^2)$ entonces la complejidad del algoritmo para optimizar p parámetros sería $O(p^3 \cdot \ln p)$.

La mutación en genotipos reales consiste en sumar pequeños números y normalizar al rango de cada variable (suponemos que se aplica con probabilidad p_m=1/p). La ventaja de este tipo de genotipo y mutación es que la complejidad de optimizar un parámetro individual de la cadena está acotada siempre por una constante.

De esta forma, la complejidad en caso de epistasis [NDV97] entre dos parámetros de la cadena es $O(p^2 \cdot \ln p)$ y además el operador de mutación tiene mayor probabilidad de causar cambios en paralelo que en el caso binario (y existe mayor libertad de elección del operador).

Entre las ventajas de usar codificación real podemos citar la posibilidad de manejar búsquedas en grandes dominios (incluso desconocidos) para las variables problema, alta precisión, "gradualidad", mayor posibilidad de ajuste local, facilidad de expresión del genotipo en fenotipo, velocidad de ejecución, etc... En numerosos trabajos se aboga por usar un alfabeto lo más expresivo posible [Anto89] e incluso en definir tantos operadores como sea posible en el dominio del fenotipo [Radc92].

2.7.1.3. Otros Resultados Interesantes Relacionados

Inicialmente los resultados sobre el teorema de los esquemas conducían siempre a desarrollos basados en la destrucción más que en la construcción de soluciones que realizan los operadores genéticos. Esta tendencia cambió con trabajos como [BG87] [Whit93a] en que se presenta el cálculo de una expresión exacta de la proporción esperada de una cadena particular en la próxima generación basándose en las ganancias y pérdidas de los esquemas. Estos y otros trabajos se están desarrollando para conseguir una versión ejecutable que permita simular el comportamiento del algoritmo mejorando los valores predichos.

El teorema de los esquemas, a pesar de las mejoras que se están estudiando no caracteriza totalmente el funcionamiento del algoritmo porque sólo sirve para predecir qué pasa en la siguiente generación. Además, el trabajo con esquemas es relativamente independiente de la función que se optimiza. Sin embargo, parece claro que incorporar en los estudios información sobre las características de dicha función podría resultar ventajoso [Radc92].

De acuerdo con esta idea existen estudios muy interesantes basados en el uso de probabilidades y análisis de Fourier, procesos de Markov y velocidad de convergencia de los que el lector puede obtener un resumen en detalle especialmente brillante en [BGKK97].

A pesar de los esfuerzos recientes en este campo, estos conceptos más detallados que el teorema de los esquemas consiguen a duras penas ser útiles en la práctica, ya que pueden estudiarse en funciones tan simples como "contar unos", en las que se puede derivar una función cerrada a la *respuesta a la selección* [MS93] o modelos similares después de un importante desarrollo matemático. Existen numerosos estudios sobre conceptos concretos como el tamaño óptimo de la población [Gold89b] [Reev93b], probabilidades de los operadores tradicionales [Bäck96], tiempo de convergencia [LR93], etc.

2.7.2. Extensión de los Fundamentos para los Modelos Paralelos

Los resultados que el teorema de los esquemas y el resto de modelos teóricos representan se refieren a algoritmos genéticos o evolutivos secuenciales. Los modelos celular y distribuido han recibido poca atención teórica. En particular, los resultados de [PL89] permiten extender las características de crecimiento exponencial y eficacia de un modelo secuencial a un modelo distribuido en islas débilmente acopladas como los que nos interesan.

Para ello se considera de hecho las probabilidades de selección, mutación y cruce y se deriva el *teorema de los esquemas modificado* introduciendo la probabilidad de selección de emigrantes, frecuencia de envío y probabilidad de aceptación en la población receptora, atendiendo a criterios realmente genéricos.

Debemos mencionar que para llegar a esta extensión la única suposición es que *las islas mantienen* en el momento de la migración *valores medios de adecuación similares* y que la migración es esporádica. En un modelo distribuido síncrono homogéneo en que las poblaciones se generen siguiendo una distribución de probabilidad uniforme es normal que esta suposición se cumpla. De hecho, se ha comprobado en la práctica para el grupo de funciones de DeJong. Igualmente, es de suponer su validez para un modelo distribuido asíncrono homogéneo si la ejecución es sobre estaciones del mismo tipo, ya que no existe cuello de botella en las comunicaciones y además las islas progresan prácticamente a la par.

Sin embargo, en general, si la búsqueda o el sistema de ejecución son heterogéneos, los supuestos teóricos no se cumplen a priori. Esta es una línea de investigación interesante tanto teórica como experimentalmente, ya que existen pocos algoritmos ejecutables de este tipo. Analizaremos el caso homogéneo, en donde los resultados teóricos se mantienen para el modelo distribuido. Adicionalmente, puede consultarse en [AT99a] un resumen sobre cómo es posible asegurar formalmente una elevada eficiencia con índices de paralelismo razonables como los que pueden encontrarse en una red de estaciones de trabajo y equipos de fácil disponibilidad.

Por otro lado, casi todos los resultados existentes son para algoritmos genéticos paralelos distribuidos. Los modelos celulares han recibido mucho interés experimental en los últimos años pero pocos resultados teóricos. A pesar de existir muchas comparativas importantes [Sten93] [GW93], quizás los logros teóricos de DeJong y Sarma [DS95] [SD96] [SD97] sean los más serios y los que abordan el problema fundamental de la selección local, ya que éste es el elemento que distingue a estos modelos respecto al resto. Nuestra caracterización de la rejilla de un modelo celular es una extensión de estos trabajos.

2.7.3. Modelos Teóricos de Presión Selectiva

La caracterización de los modelos de presión selectiva es muy importante para entender cuantitativamente el funcionamiento del operador de selección, tan definitivo en cualquier algoritmo evolutivo descentralizado o no. Un primer trabajo en este sentido [Sysw91] permitió demostrar que un modelo estacionario con reemplazo aleatorio es equivalente a un modelo generacional, así como sugerir las ventajas de otros tipos de reemplazo en el modelo estacionario. En este sentido, pretendemos aquí caracterizar ambos suponiendo un reemplazo del peor existente. Este funcionamiento introduce en el modelo estacionario un elitismo implícito a coste cero y es razonable esperar mejores resultados.

Además, vamos a extender este estudio comparativo para incluir los modelos celular y distribuido. Atendiendo a una selección proporcional, presentamos en la Tabla 2.3 la presión selectiva de cada modelo. Mostramos el número esperado de instancias $n_{t+1}(f)$ que una cadena cualquiera de adecuación f tendrá en el próximo paso del algoritmo. A continuación pasamos a explicar su derivación.

En un modelo generacional, cada cadena recibe tantas instancias como indica la relación de su adecuación respecto de la adecuación media de la población. Respecto al modelo estacionario, el reemplazo aleatorio muestra una presión equivalente al generacional después de haber generado de forma estacionaria $n=\mu$ individuos.

En ssGA, si reemplazamos la peor cadena en cada paso entonces el número de instancias sólo disminuye para dicha cadena, mientras que aumenta proporcionalmente para el resto de cadenas.

Para el algoritmo celular, reproducimos los resultados de [SD96]; $n^*(f)$ es la proporción esperada en la población final (convergencia) de la cadena cuya adecuación es f. El parámetro a depende de la técnica de selección y de la relación entre los radios de la malla y del vecindario. Podemos deducir la expresión genérica para calcular el porcentaje de la *mejor clase* como:

$$P_{b,t} = \frac{1}{1 + \left(\frac{1}{P_{b,0}} - 1 \right) \cdot e^{-at}} \tag{2.21}$$

Finalmente, el modelo distribuido genera instancias de individuos atendiendo a la suma de instancias de las islas componentes. La migración es el único operador que perturba esta operación incrementando de manera uniforme el número de instancias de cada cadena debido a que presentamos el modelo suponiendo que se elige un único emigrante de forma aleatoria. La única reducción ocurre sobre la peor cadena y únicamente cuando se recibe la cadena emigrante de la isla vecina.

TABLA 2.3. MODELOS DE PRESIÓN SELECTIVA PARA CADA ALGORITMO

- **GENERACIONAL:** $$n_{t+1}(f) = n_t(f) \cdot \frac{f}{\bar{f}}$$
- **ESTADO ESTACIONARIO(aleatorio):** $$n_{t+1}(f) = n_t(f) + n_t(f) \cdot \left(\frac{f}{\sum_{i=1}^{n} f_i} - \frac{1}{n} \right)$$
- **ESTADO ESTACIONARIO(peor):** $$n_{t+1}(f) = n_t(f) + n_t(f) \cdot \left(\frac{f}{\sum_{i=1}^{n} f_i} \right) - \left(1 - \left\lceil \frac{f - f_{min}}{f} \right\rceil \right)$$
- **CELULAR:** $$n_{t+1}(f) = \frac{n^*(f)}{1 + \left(\frac{n^*(f)}{n_0(f)} - 1 \right) \cdot e^{-at}}$$
- **DISTRIBUIDO:** $$n_{t+1}(f) = \sum_{i=1}^{d} n_{t+1}^i(f) \quad | \quad n_{t+1}^i(f) = n_t^i(f) + n_t^i(f) \cdot \left(\frac{1}{n} \right) - \left(1 - \left\lceil \frac{f - f_{min}}{f} \right\rceil \right)$$

Para aclarar y profundizar en estos modelos hemos desarrollado simulaciones por computador de ellos y ofrecemos la salida en los diagramas de la Tabla 2.4. Usamos una población de 100 individuos dividida en 10 grupos de 10 valores cada grupo presentando adecuaciones de 1, 2, ... 10 (a=0.225 para el caso celular). Todos ellos utilizan selección proporcional a la adecuación.

Confirmamos así la similitud entre la presión del modelo generacional y del estacionario con reemplazo aleatorio (primera y segunda filas de la Tabla 2.4) para igual número de evaluaciones (llamémosle PS1). El modelo estacionario con reemplazo siempre de la peor cadena presenta una convergencia más rápida (PS2) en teoría, que además confirmaremos empíricamente en el Capítulo 4. El modelo celular presenta una presión ajustable (PS3) dependiente del valor de a.

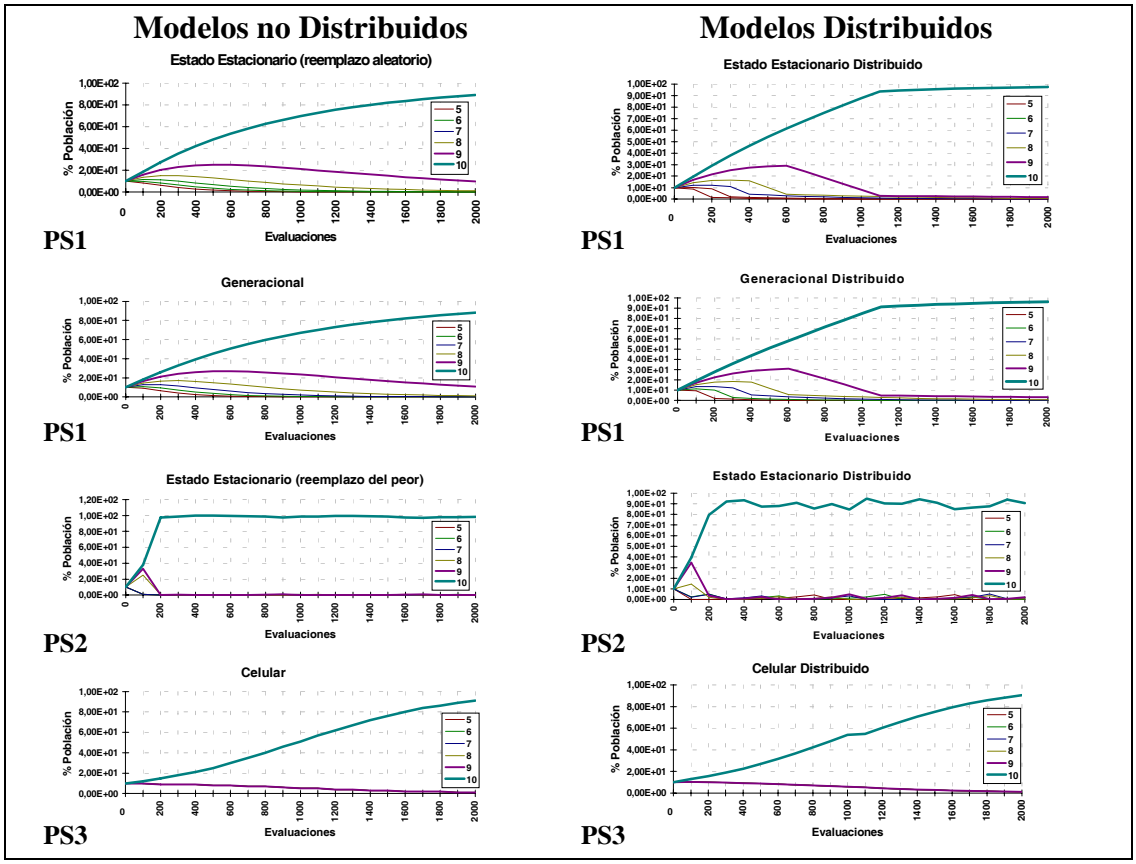
En resumen, podemos distinguir tres tipos de presión selectiva:

- **PS1** ⇨ Típica de una solución de compromiso entre exploración y explotación.
- **PS2** ⇨ Típica de una rápida explotación de las soluciones de la población.
- **PS3** ⇨ Presión selectiva ajustable.

La evolución del modelo paralelo afecta a PS1 acelerando su convergencia, mejora la diversidad de PS2, y acelera ligeramente el tipo de selección PS3. Hay que tener en cuenta que estos resultados no consideran las operaciones de cruce ni mutación, quienes pueden modificar cualquiera de los modelos presentados, aunque las curvas básicas se mantienen en la práctica (Capítulo 4).

La presión selectiva puede variarse atendiendo al operador de selección. En general, es un hecho demostrado que la presión en un modelo secuencial es menor si se usa selección proporcional y mayor para *ranking* o torneo binario (equivalentes en este aspecto). Estos resultados se aplican también al modelo celular, si bien cuantitativamente el comportamiento es distinto respecto a las versiones secuenciales en panmixia [DS95].

TABLA 2.4. SIMULACIÓN POR COMPUTADOR DE CADA MODELO DE PRESIÓN SELECTIVA



Una de nuestras hipótesis de trabajo es que, puesto que el modelo celular cambia su presión selectiva atendiendo a la relación entre los radios de vecindario y topología, es más simple variar la forma de la malla que la definición del vecindario cuando el objetivo es cambiar la presión.

2.8. Implicaciones de los Resultados Presentados

Las principales implicaciones de los resultados mostrados en este capítulo, enunciadas en el orden que se han presentado, son las siguientes:

- Existe una base algorítmica común en los modelos centralizados y descentralizados que hemos desarrollado en teoría y hemos usado para describir modelos existentes y proponer un nuevo modelo de evolución de islas secuenciales o celulares. Esto supone la creación de un nexo de unión entre dichos algoritmos para poder trasvasar desarrollos de uno a otro.
- Los algoritmos genéticos minimizan el número de puntos visitados más rápido que otras técnicas como el enfriamiento simulado aunque más lentamente que algoritmos a la medida sobre los problemas estudiados. Esto sugiere que son ventajosos en múltiples tipos de problemas de interés actual.

- Los dominios de aplicación reales requieren de estructuras complejas. Los algoritmos genéticos pueden resolver problemas en estos campos de forma satisfactoria. Como consecuencia de los resultados analizados y de la literatura existente concluimos que la eficiencia es un punto crucial que se debe mejorar. En esta línea hemos ofrecido resultados preliminares sobre las ventajas de usar un algoritmo paralelo con islas de estado estacionario.
- El teorema de los esquemas y las correcciones para convertirlo en una igualdad son muy interesantes para comprobar si un modelo nuevo se adecua o no a las bases teóricas de trabajo. Sin embargo, no tiene en cuenta la función objetivo, y en ese aspecto es criticable y mejorable a través de otros tipos de análisis que permitan obtener soluciones cerradas, o al menos que permitan simular el comportamiento de un algoritmo genético.
- Las desventajas históricas predichas por el teorema de los esquemas para usar alfabetos de aridad mayor que 2 son erróneas. La nueva interpretación permite fundamentar el éxito de codificaciones reales o enteras en muchas aplicaciones. Asimismo, se presenta en algunos dominios como más interesante la codificación Gray que la estándar pura, con la explicación de que la primera permite evitar las largas distancias de Hamming entre algunas codificaciones que representan valores numéricos adyacentes ("acantilados").
- Los modelos distribuidos síncronos presentan las características teóricas de los modelos secuenciales con la ventaja añadida de disminuir potencialmente el tiempo real de ejecución y muchas veces el esfuerzo de evaluación. Igual ocurre con los modelos asíncronos sobre sistemas homogéneos, permitiendo además acelerar la comunicación entre islas (hipótesis que debemos constrastar). La topología en anillo es buena por provocar un tiempo de comunicación constante y permitir así mejorar la ganancia en velocidad al aumentar el número de islas (ligeramente acopladas).
- La relación entre el radio de la rejilla y del vecindario en un algoritmo genético celular permite introducir en dicho modelo cualquier presión selectiva que se desee. Es más simple modificar la forma de la malla que el vecindario. Esto, junto con los resultados existentes sobre su eficiencia, hacen pensar que distribuir algoritmos celulares es muy ventajoso (hipótesis de trabajo).
- La presión selectiva es mayor en una evolución por estado estacionario con reemplazo de la peor solución que en el caso generacional. La evolución por difusión presenta en teoría una presión intermedia que puede llevarse a uno u otro extremo variando la forma de la rejilla (hipótesis en nuestro acercamiento) o bien el tipo de vecindario (típico en otros trabajos).

3
El Modelo Propuesto

En este capítulo formalizaremos la propuesta de un modelo paralelo distribuido de algoritmo genético cuyos fundamentos, necesidad, características globales y pseudocódigo básico han sido ya desarrollados en los dos capítulos anteriores.

Se espera conseguir con este modelo un sistema de búsqueda flexible y eficiente que sea, además, lo suficientemente genérico como para poder derivar de él distintas familias de algoritmos paralelos.

En la Sección 3.1 realizamos un conjunto de consideraciones iniciales que nos permitan comprender el alcance de la propuesta y exponer los beneficios y problemas que cabe esperar. Se propone un vasto abanico de familias de algoritmos cuyas diferencias se basan en los aspectos paralelos alternativos más importantes que se pueden obtener de su formalización. La Sección 3.2 concreta las características del modelo propuesto, mientras que la Sección 3.3 detalla el modelo en sí y la relación con los conceptos formales y computacionales de los dos capítulos precedentes.

La Sección 3.4 contiene un estudio de la presión selectiva de los diferentes modelos derivables del modelo propuesto. El objetivo es comprobar el comportamiento teórico descrito en la Sección 2.7.3 sobre instancias des/centralizadas de algoritmos derivables del modelo de sistema adaptativo granulado. En la Sección 3.5 discutimos las ventajas e inconvenientes de utilizar un algoritmo genético paralelo distribuido en el que cada isla evoluciona atendiendo a un algoritmo genético celular. La razón de este análisis es profundizar y conocerlo mejor como método de búsqueda diferenciado del resto de algoritmos de su clase.

La Sección 3.6 presenta las principales decisiones y técnicas de diseño e implementación de un sistema paralelo que deba sustentar los variados requisitos de funcionamiento para una familia tan diversa de algoritmos. En particular, se discute el diseño en forma de clases de objetos, el sistema distribuido de comunicación y algunas consideraciones importantes sobre las decisiones de implementación.

Posteriormente, la Sección 3.7 describe la relación existente entre nuestro modelo genérico y los marcos de programación. El diseño de un sistema orientado a objetos para estos algoritmos posee ciertas características deseables en sí mismas que además pueden extenderse al mencionado paradigma de diseño de *software* paralelo. Los marcos de programación permiten hacer llegar sistemas como el nuestro a una audiencia no especializada en paralelismo, aumentando así su interés y su uso. Tanto el diseño orientado a objetos como nuestra propuesta de extensión a un marco de programación son dos aspectos del diseño que afectan a la implementación, requiriendo que el *software* evolutivo usado sea especialmente flexible y modular.

Para terminar este capítulo, la Sección 3.8 resume el contenido de este capítulo.

3.1. Crítica a los Modelos Paralelos Tradicionales

Parece claro en este punto del desarrollo que, atendiendo a las aplicaciones y desarrollos teóricos existentes, el trabajo sobre sistemas evolutivos paralelos es de innegable interés y proporciona múltiples ventajas. Sin embargo, no podemos extraer conclusiones claras sobre algunos aspectos importantes basándonos en trabajos existentes. Esto se debe a que dichos trabajos estudian siempre muy parcialmente modelos paralelos absolutamente propios y nada estándares, y además lo hacen sobre problemas de laboratorio o incluso sobre un único problema.

Una de las razones que explica la situación actual ha sido la falta de unificación en el estudio, lo que pretendemos subsanar en este trabajo.

Existen algunos modelos paralelos que guardan cierta similitud con el que proponemos. Estos modelos se han propuesto para estudiar aspectos concretos como los siguientes:

- Migración y búsqueda local; el modelo se organiza como un algoritmo genético distribuido jerárquico llamado HSDGA [VSB92], usado para optimización funcional.
- Aprovechamiento de 4 máquinas conectadas entre sí, cada una con varios procesadores unidos por una malla física [Grua94]; usado para el diseño de redes de neuronas.
- Simulación de sistemas naturales [Pedr96].
- Lenguaje de descripción de algoritmos evolutivos RPL2 [RS94].

A pesar de surgir de forma no estructurada ni estándar, estos modelos han proporcionado, cada uno en su campo de aplicación, resultados bastante interesantes. Existen otros modelos donde las ideas expuestas han llevado a generalizaciones o mezclas en cuanto a paralelismo. Sin embargo, ninguno de los mencionados hasta el momento representa un *modelo* en sí ni ha sido estudiado como tal. En general, se trata de *implementaciones* que ningún otro autor puede analizar de forma justa ni replicar (por el tipo de máquinas u operaciones que utiliza).

En nuestro caso pretendemos caracterizar una plantilla de búsqueda paralela a la que llamaremos *xx*GA, indicando con "*xx*" que podemos generar a partir de la visión unificada tanto modelos de población única como ssGA o genGA ("*xx*"="ss", "*xx*"="gen"), como celulares ("*xx*"="c"), distribuidos celulares ("*xx*"="dc") o muchas otras familias de algoritmos atendiendo a cómo definamos los parámetros que determinan la evolución básica, la política de migración y las técnicas de búsqueda presentes. La Figura 3.1 describe gráficamente el origen de las ideas subyacentes para la elección de esta nomenclatura.

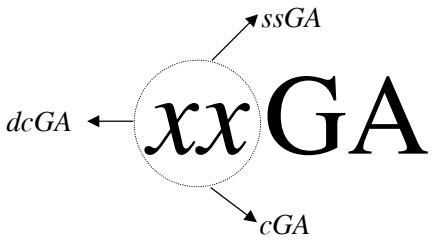


Figura 3.1. *xxGA es un modelo genérico del que pueden extraerse varias familias concretas.*

El modelo propuesto es distribuido a alto nivel. Su implementación en computadores MIMD es directa, pudiéndose situar una o más islas por procesador. Cumplimos así con el objetivo de hacerlos útiles en la práctica gracias a la amplia disponibilidad de este tipo de máquinas en cualquier organización.

3.2. El Modelo *xx*GA

Nuestra visión unificada de los modelos secuenciales y paralelos nos permite pensar en cualquier modelo de gránulos conectados (Capítulo 2), donde cada gránulo es un algoritmo de complejidad arbitraria.

Esta concepción permite de forma natural definir *xx*GA como numerosas familias de algoritmos, en nuestro estudio algoritmos genéticos, cada uno con sus respectivas propiedades y especialmente indicados para problemas con distintos requisitos.

Las extensiones con hibridación [Cott98] son especialmente claras con nuestra formalización en términos de **cooperación/competición** o de **modelos coercitivos** donde un algoritmo decide cuándo y cómo usar a otro.

La Figura 3.2 muestra gráficamente cómo pensar en una familia concreta de los algoritmos descritos por *xx*GA atendiendo a los procesos de evolución de islas-gránulo con migración. Esta visión recoge las ideas modernas sobre el diseño de algoritmos evolutivos y las tendencias sobre análisis y uso de familias parametrizables que puedan adaptarse fácilmente para su uso en distintos problemas [BHS97].

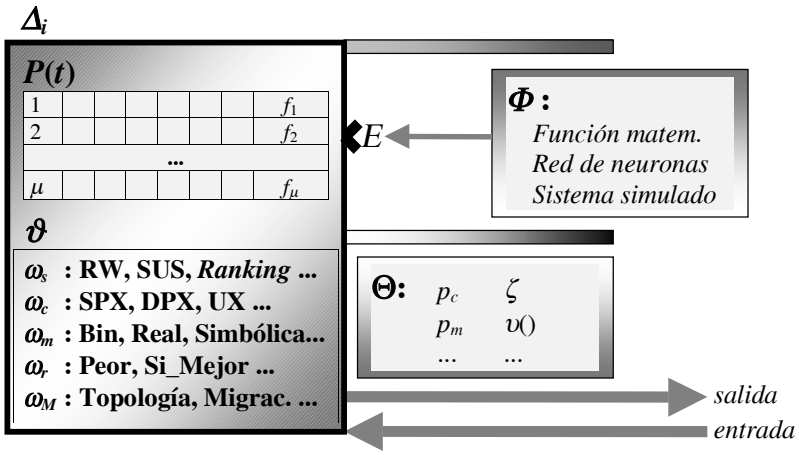


Figura 3.2. *xxGA puede verse como un modelo de población única, celular, o de otro tipo, listo para trabajar de forma distribuida usando migración.*

Debemos tener en cuenta que el modelo *xx*GA es intrínsecamente heterogéneo a nivel de búsqueda, ya que nada fuerza a usar gránulos iguales. Incluso los gránulos podrían ejecutar distintos algoritmos, quizás no evolutivos.

La siguiente tabla resume la forma en que podemos referirnos a las distintas familias de algoritmos homogéneos que pueden derivarse de *xx*GA.

TABLA 3.1. SIMBOLOGÍA USADA PARA CONCRETAR UNA FAMILIA DE ALGORITMOS

NOMENCLATURA	SIGNIFICADO	
*xx*GA(*d*,*s*,*m*,*g*,*e*,*re*)	$d \in \{1,...,\infty\}$	número de subpoblaciones ejecutándose en paralelo
	$s \in \{1,...,\infty\}$	tamaño de la población en cada isla
	$m \in$ {r,best}	emigra un único individuo: uno aleatorio o el mejor
	$g \in \{1,...,\infty\}$	frecuencia de migración (#evaluaciones de evolución aislada)
	$e \in$ {ss,rw,sus,cel}	modo de evolución: estacionario, generacional (RW/SUS) o celular
	$re \in$ {alw,ifb,e,ne}	política de reemplazo: siempre, si mejor, elitista o no elitista, respect.

EJEMPLOS:

xxGA(_,100,_,_,_,ss,ifb) ⇨ Un AG con 100 individuos con islas estacionarias y reemplazo sólo si mejor
xxGA(4,2x32,r,64,cel,ifb) ⇨ Cuatro cGAs cada uno de malla 2x32 individuos emigrando una cadena aleatoria cada 64 evaluaciones reemplazando cada punto sólo si el hijo generado es mejor que él

ACRÓNIMOS:

ssi ≡ *estado estacionario con reemplazo "si mejor"* — **ssa** ≡ *estado estacionario, reemplazo "siempre"*
rwe ≡ *ruleta con elitismo* — **rwn** ≡ *ruleta sin elitismo*
suse ≡ "*muestreo estocástico" universal elitista* — **susn** ≡ "*muestreo estocástico" universal sin elitismo*
ci ≡ *cGA con reemplazo "si mejor"* — **ca** ≡ *cGA con reemplazo "siempre"*
dssi, **dssa**, **drwe**, **drwn**, **dsuse**, **dsusn**, **dci**, **dca** ≡ *versiones distribuidas de los anteriores algoritmos*

La extensión directa a modelos asíncronos la denotaremos usando corchetes. Por ejemplo *xx*GA[4,2x32,r,64,cel,ifb] es la versión asíncrona del modelo síncrono distribuido definido por *xx*GA(4,2x32,r,64,cel,ifb). La notación para modelos heterogéneos no se incluye, ya que no se presentan resultados directos con ellos, si bien el diseño y las implementaciones realizadas permiten usarlos y proponemos esta línea como trabajo futuro.

Puede observarse que en la notación no se especifican operadores de variación. Esto se debe a que los modelos estudiados (comparados) en cada apartado usan los mismos. De otra forma no podría realizarse un estudio justo entre modelos paralelos.

3.3. Caracterización

A continuación presentamos la descripción de un modelo distribuido de islas de población única o celulares. En general, este modelo puede funcionar como un algoritmo no distribuido tradicional (una única isla sin comunicación) o como un algoritmo (paralelo) distribuido.

El algoritmo está distribuido y por tanto consta de d gránulos comunicantes en una topología de anillo unidireccional. Todos los gránulos paralelos distribuidos se conducen internamente de acuerdo a un plan reproductor propio. En todo caso cada isla comienza generando y evaluando una población inicial de estructuras.

El algoritmo básico en cada isla itera hasta cumplir el criterio de terminación distribuido. En la mayoría de los casos este criterio consiste en alcanzar un número predeterminado de evaluaciones totales (suma de las realizadas en todas las islas) o bien en alcanzar una estructura solución. Esta última condición de parada sólo es posible si se conoce de antemano cuál es la solución al problema estudiado.

En el algoritmo notamos con μ el tamaño de cada subpoblación y con n el tamaño de la lista de individuos de donde seleccionar una pareja. Los parámetros propios de cada subalgoritmo se denotan con un superíndice (^{i}) .

Definimos en el algoritmo k vecindarios en cada isla $NP_k^i(t^i)$. En el caso celular se trata de $k=\mu^i$ vecindarios de 5 cadenas (n=4+1) cada uno, mientras que en el caso generacional, por comodidad en la notación, usamos también $k=\mu^i$ vecindarios todos ellos iguales entre sí e iguales a la población de la isla $P^i(t^i)$. En el caso de un algoritmo genético de estado estacionario existe un único vecindario que trabaja sobre toda la población de la isla, k=1. Podríamos hacer funcionalmente equivalentes a los tres algoritmos si en el caso estacionario damos μ^i pasos consecutivos sin atender la comunicación ni otro tipo de operaciones, pero así perderíamos el grano básico de esta variante (μ^i+1) que lo hace tan interesante.

La operación de selección elige a dos padres distintos entre sí para limitar en lo posible las redundancias con un coste casi nulo (esto no implica que los hijos generados sean absolutamente nuevos, pero mejora la diversidad). Tras cruzar, mutar y evaluar el hijo generado (podría usarse una *ventana* ω para ello) se lo introduce por reemplazo, bien en la población actual (estado estacionario) o bien en una población temporal hasta haber completado el trabajo sobre todos los vecindarios. $P^{i'}(t^i)$ es quien contiene la nueva población en ambos casos.

El reemplazo elimina la cadena considerada siempre, a menos que el parámetro *ifb* (*if_better*) esté activo, indicando que debe reemplazarse sólo si la nueva es mejor que la existente. Si no es así, se mantiene la anterior. Para terminar, la fase de comunicación envía/recibe un único individuo (m=1) y reemplaza en cada subpoblación su peor cadena con la cadena que llega desde su vecino en el anillo (determinado por la función υ). Este proceso se realiza de forma determinista cada ζ pasos o con probabilidad $p_M=1/\zeta$ en cada subalgoritmo.

A continuación pasamos a detallar el significado de los nuevos símbolos usados.

❖ $best \in \{0,1\}$ $n = 4$ $ifb \in \{\texttt{false},\texttt{true}\}$ (*estrategia de reemplazo*)

El símbolo *best* indica si se está migrando la mejor solución de la isla (1) o bien una estructura aleatoria (0). El tamaño del vecindario es constante y de valor n=4. Si *ifb*=`true` entonces reemplazamos únicamente si la nueva cadena es mejor que la elegida para reemplazar, en otro caso (`false`) siempre debe aplicarse el reemplazo.

❖ $cel, gen \in \{\texttt{false},\texttt{true}\}$ *gen*: *¿algoritmo generacional?* *cel*: *¿algoritmo celular?*

El símbolo *gen* indica si se está ejecutando un AG generacional (`true`) o no (`false`). El valor *cel* indica si se trata de un AG celular (`true`) o no (`false`).

❖ $\pi : \{\texttt{false},\texttt{true}\} \rightarrow Z \quad \pi(x) := \begin{cases} 1 & x = \texttt{false} \\ \mu^i & x = \texttt{true} \end{cases}$

Define el número de gránulos vecinos. Si tratamos con un modelo secuencial entonces existe un único vecindario (panmixia). En otro caso (celular) existen μ^i gránulos.

ALGORITMO *xx*GA$_i$, $\forall i \in \{0, ..., d\text{-}1\}$

$t^i := 0;$

inicializar : $P^i(0) := \left\{\vec{a}_1^{\,i}(0),...,\vec{a}_{\mu^i}^{\,i}(0)\right\} \in \mathcal{G}^{\mu^i}$;

evaluar : $P^i(0) : \left\{\Phi\left(\vec{a}_1^{\,i}(0)\right),...,\Phi\left(\vec{a}_{\mu^i}^{\,i}(0)\right)\right\}$ *donde* $\Phi\left(\vec{a}_k^{\,i}(0)\right) := \varphi\left(f\left(\Gamma^{-1}\left(\vec{a}_k^{\,i}(0)\right)\right), P^i(0)\right)$;

mientras no $\iota_i\left(P^0\left(t^i\right),...,P^{d-1}\left(t^i\right)\right)$ **hacer:**

$\forall NP_k^i\left(t^i\right), k := \{1,...,\pi(cel \vee gen)\}$**:** // *inicialmente* $NP_k^{i\,\prime}\left(t^i\right) := NP_k^i\left(t^i\right)$, $NP_k^{i\,\prime\prime}\left(t^i\right) := \varnothing$

// *y* $NP_k^i(t^i) := \neg cel \wedge \left\{P^i(t^i)\right\} \vee cel \wedge news(k)$;

seleccionar : $\left[\vec{a}_a^{\,i}(t^i) := s_{\{p_s\}}\left(NP_k^i(t^i)\right)\right] \wedge \left[\vec{a}_b^{\,i}(t^i) := s_{\{p_s\}}\left(NP_k^i(t^i)\right)\right]$ *siendo* $\vec{a}_a^{\,i}(t^i) \neq \vec{a}_b^{\,i}(t^i)$,

donde $p_s\left(\vec{a}_k^{\,i}(t^i)\right) := \Phi\left(\vec{a}_k^{\,i}(t^i)\right) \Big/ \sum_{j=1}^{n} \Phi\left(\vec{a}_j^{\,i}(t^i)\right)$;

recombinar : $\vec{a}_k^{\,i\,\prime}(t^i) := \otimes_{\{p_c\}}\left(\vec{a}_a^{\,i}(t^i), \vec{a}_b^{\,i}(t^i)\right)$;

mutar : $\vec{a}_k^{\,i\,\prime\prime}(t^i) := m_{\{p_m\}}\left(\vec{a}_k^{\,i\,\prime}(t^i)\right)$;

evaluar : $\Phi\left(\vec{a}_k^{\,i\,\prime\prime}(t^i)\right) = \varphi\left(f\left(\Gamma^{-1}\left(\vec{a}_k^{\,i\,\prime\prime}(t^i)\right)\right), P^i(t^i - \omega)\right)$;

pobl. auxiliar : $NP_k^{i\,\prime\prime}(t^i) := NP_k^{i\,\prime\prime}(t^i) \cup \left\{\vec{a}_k^{\,i\,\prime\prime}(t^i)\right\}$;

reemplazar : $\forall k := \{1,...,\pi(cel \vee gen)\}$: $NP_k^{i\,\prime}(t^i) := r_{\{p_r\}}\left(NP_k^{i\,\prime}(t^i), \vec{a}_k^{\,i\,\prime\prime}(t^i)\right)$,

donde $p_r\left(\vec{a}_k^{\,i}(t^i)\right) := \begin{cases} 1 \;\; \textbf{si} \;\; gen \\ \textbf{si no,} \;\; 1 \;\; \textbf{si} \;\; \left(\neg ifb \wedge cel\right) \vee \neg cel \wedge \left[\left(\neg ifb \wedge k = worst\right) \vee \left(ifb \wedge \left(\Phi\left(\vec{a}_k^{\,i}(t^i)\right) \leq \Phi\left(\vec{a}_k^{\,i\,\prime\prime}(t^i)\right)\right)\right)\right] \\ \qquad\qquad 0 \;\; \textbf{si} \;\; \left[ifb \wedge \left(\Phi\left(\vec{a}_k^{\,i}(t^i)\right) > \Phi\left(\vec{a}_k^{\,i\,\prime\prime}(t^i)\right)\right)\right] \wedge cel \;\; \vee \;\; \neg cel \wedge k \neq worst \end{cases}$

comunicación$(aplicar\,\omega_{M\{\mu\}},\ \left(sínc \wedge \left(t^i \bmod \zeta = 0\right) \wedge t^i \neq 0\right) \vee \left(asínc \wedge \left(\sigma \leq p_M = 1/\zeta, \sigma \in [0..1]\right)\right),\ m = 1)$:

seleccionar emigrante : $\vec{a}_m^{\,i}(t^i) := \omega_{S\{p_S\}}\left(P^{i\,\prime}(t^i)\right)$, *donde* $P^{i\,\prime}(t^i) = \bigcup_{k=1}^{\pi(cel \vee gen)} NP_k^{i\,\prime}(t^i)$

y $p_S\left(\hat{\vec{a}}_k^{\,i}(t^i)\right) = (1 - best) \cdot \frac{1}{\mu^i} + best \cdot \beta\left(\Phi\left(\vec{a}_1^{\,i\,\prime\prime}(t^i)\right),...,\Phi\left(\vec{a}_{\mu^i}^{\,i\,\prime\prime}(t^i)\right), k\right)$, $\hat{\vec{a}}_k^{\,i}(t^i) \in P^{i\,\prime}(t^i)$ **;**

migrar : $P^i(t^i + 1) := \omega_{R\{p_R\}}\left(P^{i\,\prime}(t^i), \vec{a}_m^{\,z}(t^i)\right)$,

donde $z := \upsilon(i) = (i+1) \bmod d$ *y* $p_R\left(\hat{\vec{a}}_k^{\,i}(t^i)\right) = \begin{cases} 1 & \textbf{si} \quad k = worst' \\ 0 & \textbf{en otro caso} \end{cases}$ **;**

$t^i := t^i + 1;$

fin mientras.

❖ $P^i(t^i) := \neg cel \wedge P^i(t^i) \;\cup\; cel \wedge \left\{\xi(\Delta_a, \Delta_b) \neq \varnothing \quad \forall a,b \in \left\{1,...,\mu^i\right\}\right\}$

P^i define el conjunto de individuos sobre el que se trabaja, dependiendo del tipo de evolución básica (celular o no).

❖ $\forall P^i(t^i),\;\; \exists worst \in \left\{1,...,\mu^i\right\} \mid \forall k \in \left\{1,...,\mu^i\right\},\;\; \Phi\left(\vec{a}^i_k(t^i)\right) \geq \Phi\left(\vec{a}^i_{worst}(t^i)\right)$

El valor *worst* distingue la posición en la población actual donde reside la estructura de peor adecuación. Idem para *worst'* y $P^{i'}(t^{i'})$.

❖ $\beta : IR^{\mu^i} \times IN \rightarrow \{0,1\}$

$$\beta\left(\Phi\left(\vec{a}^i_1(t^i)\right),...,\Phi\left(\vec{a}^i_\mu(t^i)\right),k\right) := \left\{\begin{array}{ll} 1 & \textbf{si} \quad \forall j \in \left\{1,...,\mu^i\right\} \mid \Phi\left(\vec{a}^i_k(t^i)\right) \geq \Phi\left(\vec{a}^i_j(t^i)\right) \\ 0 & \textbf{en otro caso} \end{array}\right\}$$

La función β traslada un conjunto de valores de adecuación y una posición al valor 1 si dicha posición contiene la estructura con la mejor adecuación.

Respecto al algoritmo presentado usaremos, además de la versión síncrona, versiones asíncronas de todas las familias paralelas distribuidas que extraigamos. Además, en el caso generacional usaremos 1-elitismo (también algoritmos sin elitismo) para hacer sobrevivir de forma determinista a la mejor cadena en la siguiente población (de forma que el reemplazo actúe sólo sobre μ^i–1 peores cadenas).

3.4. La Familia dcGA

Por tratarse de un modelo evolutivo novedoso, dedicamos en particular esta sección a hacer algunas consideraciones sobre la distribución de islas celulares. Nuestra visión de un modelo celular no está ligada a su ejecución en una máquina SIMD que mantenga un individuo por elemento de proceso. Este tipo de máquinas es bastante infrecuente en la mayoría de organizaciones. De hecho, su baja disponibilidad ha sido una de las razones para que el trabajo con algoritmos celulares haya estado siempre en minoría a pesar de sus ventajas.

Aunque hasta ahora la alternativa ha sido apenas usar cGAs, esta situación parece un desperdicio de posibilidades debido a que estos modelos poseen características interesantes que no dependen de su ejecución en un sistema paralelo físico. Trabajos como [GW93] y otros entienden que un cGA es un algoritmo de optimización más, con la ventaja añadida de disponer de una población estructurada espacialmente.

En cuanto al mecanismo de selección del modelo celular, algunos trabajos claramente definen ventajas al usar presiones selectivas elevadas en el vecindario conseguidas usando torneo estocástico, *ranking* o incluso ruleta con reemplazo elitista [DS95] [Mühl91]. Sin embargo, cuando un modelo está distribuido, las islas suelen usar rejillas de tamaño moderado y una excesiva presión de la selección acabaría rápidamente con la diversidad, teniendo entonces que recurrir a utilizar mayor mutación o migración (o poblaciones más grandes).

El modelo dcGA permite afinar los parámetros como migración y presión en cada isla de forma que podamos aprovechar las ventajas de los modelos distribuidos y celulares. Es un hecho que ambos tienen sus puntos a favor y en contra [Sten93] y que también ambos presentan ventajas sobre los modelos tradicionales.

Los requisitos de ejecución paralela física únicamente permiten conseguir mayor velocidad de ejecución sin alterar el algoritmo básico. Esto implica que cualquier aplicación puede usar un modelo distribuido y/o celular sin poseer una red de comunicaciones o un sistema multiprocesador. Aunque, por otro lado, en el caso de disponer de un sistema paralelo físico, las ventajas se acrecientan notablemente.

Por último, debemos mencionar algunos modelos, también híbridos en otros sentidos, que proporcionan igualmente ventajas del mismo tipo que dcGA usando otros mecanismos. En particular, son muy interesantes las propuestas de Cantú y Goldberg [CG97] de un modelo distribuido con nodos que realizan internamente una paralelización global para mejorar la eficiencia. Igualmente interesante resulta la nueva versión de ASPARAGOS conocida como ASPARAGOS96 [Gorg97] que ha abandonado su topología en escalera para usar un anillo de subpoblaciones cuyos individuos están dispuestos en un vecindario en forma también de anillo (dcGA se diferencia en usar una malla interna) debido a que el anillo permite un diámetro mayor y favorece una mejor diferenciación de los individuos. De esta manera las migraciones se producen a dos niveles.

3.5. Implementación

Una vez formalizado *xx*GA como familia de algoritmos evolutivos, en esta sección abordamos las principales decisiones de implementación que más pueden llegar a influir y caracterizar un modelo, paralelo distribuido o no, de algoritmo evolutivo y en particular genético. Respecto a la implementación, en general el diseño de las clases cuando se utiliza programación orientada a objetos es un aspecto fundamental, pues define la generalidad, flexibilidad y utilidad final del sistema de búsqueda. Por ello, dedicamos la Sección 3.5.1 a discutir distintas soluciones alternativas para el sistema de clases.

En cuanto a la implementación de modelos paralelos, parece evidente que el sistema de comunicación es de vital importancia. Su eficiencia y flexibilidad deben ser muy elevadas para poder ser usado con garantías de que no restringe por él mismo la aplicación del algoritmo resultante. Igualmente importantes son tanto las estructuras de datos intercambiadas (para configuración o durante la ejecución) como el conjunto de reglas que regulan dicha comunicación (protocolo). Por esto dedicamos la Sección 3.5.2 a exponer nuestras decisiones de implementación respecto a dicho protocolo. Debe notarse que presentamos resultados de múltiples implementaciones y por tanto siempre planteamos alternativas según el uso del algoritmo y no soluciones definitivas ni recetas universales.

Por último, en la Sección 3.5.3 describimos brevemente algunos resultados validados experimentalmente que atraen nuestra atención sobre la importancia de algunos puntos de la implementación de *software* evolutivo. En particular, discutimos la estructura que debería tener la población, algunos efectos negativos derivados de un uso puramente teórico del paradigma de la orientación a objetos, y una propuesta flexible para implementar el ciclo reproductor.

3.5.1. Diseño de Clases

Hasta ahora se ha prestado poca atención a la calidad del *software* usado en el campo de la computación evolutiva. Existen numerosas implementaciones nada estructuradas, muy poco flexibles, y en general únicamente útiles para sus creadores. Muchos autores presentan resultados sin sugerir siquiera el tipo de implementación usada. En nuestro caso, y aunque no se trate de un objetivo primario, nos interesa discutir las ventajas innegables de usar clases de objetos. Esto nos lleva a la primera pregunta evidente: ¿qué clases deben existir y cómo podemos relacionarlas?

Para empezar, es importante arrancar con una fase de diseño. Aunque esta fase no sea exhaustiva, comparar y realizar distintos diseños nos enseñará cómo generar *software* de calidad, algo que consideramos imprescindible discutir en cualquier trabajo de este tipo en Informática, aunque nuestro objetivo no sea la implementación. Prácticamente el único modelo evolutivo diseñado con estos objetivos en mente es GAME. Presentamos el modelo de clases expresado en OMT en la Figura 3.3.

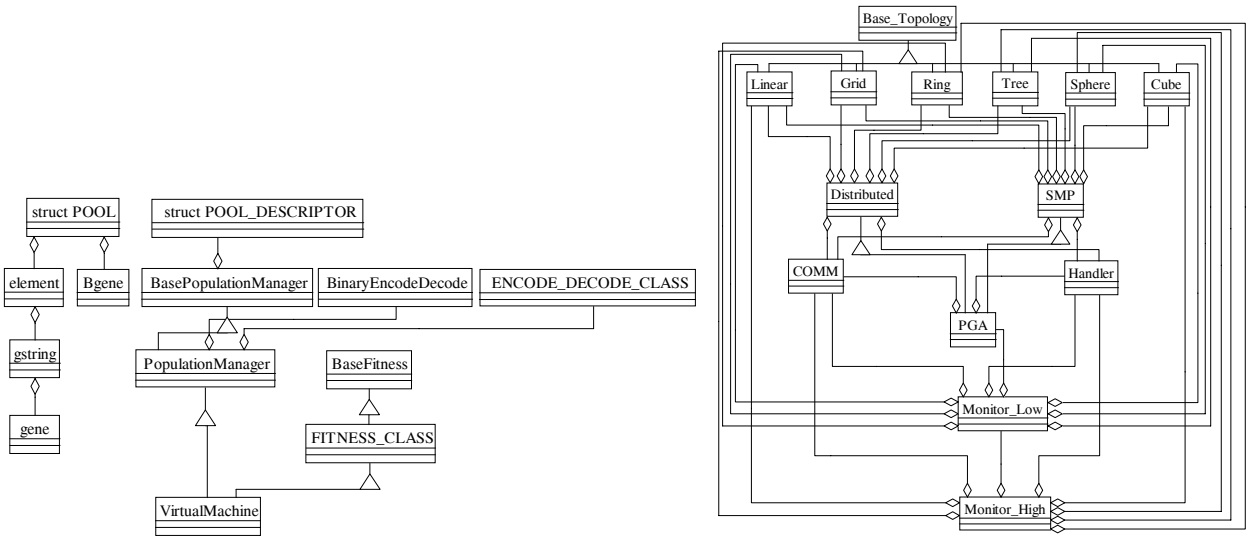


Figura 3.3. *Diseño de clases en GAME* (a) *y una extensión paralela básica -dGAME-* (b).

El diseño de clases de GAME es el esperado en un algoritmo evolutivo, e incorpora clases para los elementos básicos como genes (`gene`, `Bgene`) y cadenas de símbolos en general (`element`, `gstring`, `POOL`). La traslación entre genotipo binario y fenotipo es directa (`BinaryEncodeDecode`) a partir de estas clases, y existen mecanismos genéricos para traslaciones más sofisticadas (`ENCODE_DECODE_CLASS`). La población se maneja como un conjunto de cadenas a través de un descriptor de población (`POOL_DESCRIPTOR`). Las operaciones sobre uno o más individuos pueden realizarse utilizando un descriptor de este tipo.

El manejo básico de la población se realiza en las clases `PopulationManager` y `BasePopulationManager`. Se distinguen igualmente clases para la adecuación (`FITNESS_CLASS`, `BaseFitness`) para permitir que los usuarios realicen escalado, *sharing* u otro tipo de manipulación. La máquina virtual (`VirtualMachine`) permite definir operaciones de creación de la población y aplicación de operadores, típicas en cualquier AE.

Este esquema de clases es en teoría completo y genérico, pero en la práctica es tedioso de utilizar y en general lento. De hecho, los módulos de monitorización y paralelización que se anunciaban desde la aparición de GAME (alrededor de 1993) apenas han sido usados.

Para conocer y extender esta implementación hemos realizado una paralelización como se observa en la Figura 3.3b [Manj96] diseñando dos niveles monitorizados de ejecución (`Monitor_Low` y `Monitor_High`) que permiten combinar de cualquier forma un algoritmo evolutivo paralelo distribuido (`Distributed`) y celular (`SMP`) con gránulos de cualquier otro tipo. Las relaciones de agregación múltiples pueden mejorarse usando relaciones de herencia de la clase base `Base_Topology`, como ocurre en los modelos que actualmente estamos desarrollando. A pesar de las ventajas del uso de una comunicación eficiente (`COMM`), variedad de topologías posibles (`Grid`, `Ring`, `Tree`, ...) y otras características, la constante aparición de errores en la implementación de GAME y el volumen de los programas resultantes han hecho que, por el momento, se abandone esta línea de trabajo a pesar de la experiencia ganada.

En la Figura 3.4 mostramos el diseño de clases de dos implementaciones evaluadas en otros trabajos GPTEXT [Rubi96] e incGA [Domí97], donde podemos observar que existe un núcleo común. Es útil definir clases para los genes, tanto binarios (`Gray_Bin_Indiv`) como reales (`Double_Indiv`). Directamente podemos crear una clase individuo con su adecuación (Figura 3.4b) o bien crear una clase cromosoma para poder después generalizar y crear un individuo *n*-ploide (3.4a en la que `Individual=Chromosome+Fitness`) en vez de haploide [Gold89a].

Los operadores pueden existir como clases derivadas de una clase genérica de operador (como en la Figura 3.4a ocurre con `SUS`, `SPX`, ... y la clase `Operator`) o incluidas en el propio algoritmo (como ocurre en 3.4b), aunque esto resulte menos adecuado desde el punto de vista de la extensibilidad. De la misma forma, es más flexible que el problema también exista como clase abstracta que permita derivar problemas concretos especificando el mecanismo de evaluación deseado (Figura 3.4a).

En general, es también conveniente que las estadísticas existan como clase aparte (o estructura) para facilitar la monitorización (`PopStats` o `Statistics` en las Figuras 3.4a-b, respectivamente). El seguimiento de la ejecución es importante y delicado en un sistema paralelo evolutivo. Igualmente necesario es utilizar una clase aparte para realizar la comunicación entre gránulos (`Communications`). Por ejemplo, la Figura 3.4b presenta el diseño para un tipo de AGPs que denominamos *incrementales* (incGA) en sus cuatro versiones posibles [AT95]. En ellos, la aparición y desaparición de islas evolutivas es dinámica atendiendo a los porcentajes de mejora de la adecuación media que presentan durante la búsqueda. Esto implica que el diseño en clases debe reflejar el modelo de forma además flexible, extensible, y sobre todo eficiente.

En todos los diseños presentados es especialmente simple incluir tanto un nuevo operador como un nuevo problema. Igualmente, es posible modificar en tiempo de ejecución el conjunto de operadores y sus parámetros de trabajo gracias al concepto que hemos desarrollado de `Operator_Pool` derivado de la definición formal de ciclo reproductor.

Además, es importante señalar que la eficiencia en el uso de memoria lleva a cuidar muy especialmente las clases básicas del genotipo. La comunicación por la red de datos en el caso de los modelos distribuidos (dGAME e incGA) se hace en los casos presentados de forma empaquetada para minimizar el tiempo de tránsito entre gránulos.

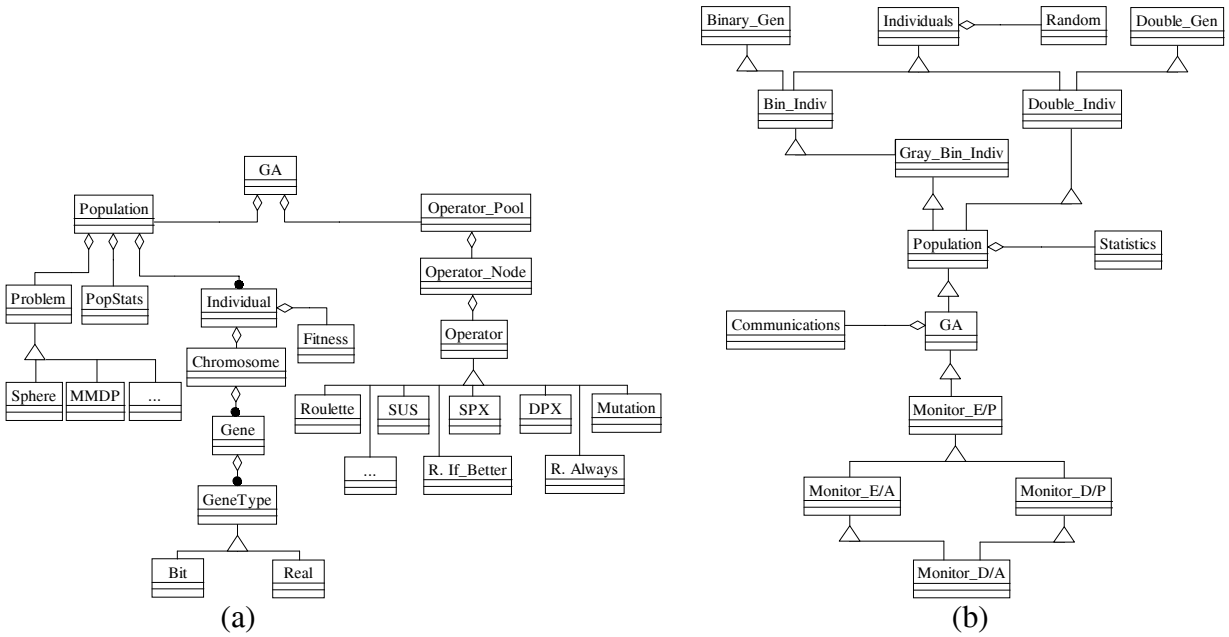


Figura 3.4. *Diseño de clases en GP-TEXT* (a) *e incGA* (b).

Todos estos conceptos, junto con la generalización de algunos de ellos han dado lugar a una de nuestras implementaciones más completas de *xx*GA en C++ (véase la Figura 3.5). Aunque existen numerosas implementaciones que aportan resultados a esta tesis escritas en C [AC97], Modula-2 (como tipos abstractos de datos) y JAVA [Muño98], sólo se han usado parcialmente debido, respectivamente, a su poca modularidad, limitación del paralelismo, o lentitud.

Puede observarse en la Figura 3.5 que hemos incluido clases para genes binarios y reales, individuos compuestos por un vector de genes y una adecuación (`Individual`) y una población de individuos (`Population`). Sobre la población es posible estudiar numerosos parámetros y definir muy distintos tipos de vecindario bi/tri-dimensionales (clase `Neighborhood`). Se incluye también una clase abstracta `Problem` para derivar problemas concretos como funciones numéricas (`F1`, `F2`, ...) o incluso redes de neuronas (`FFN`, `Pattern` y `BPN`) pensadas para trabajar con perceptrones multicapa.

Los operadores derivados y potencialmente derivables de este diseño pueden ser o no (híbridos) evolutivos (`Hybrid_Operator` y `Operator_Pool`). El algoritmo genético (clase `GA`) usa la población y el grupo de operadores para realizar el ciclo reproductor y resolver un problema concreto. En general, muchas clases necesitan de forma auxiliar un generador de números aleatorios, por lo que se proporciona como clase aparte `Random` con algunos métodos adicionales para la toma de decisiones básicas no deterministas en el código.

El sistema de monitorización en línea requiere de toma de tiempos y consumo de recursos como memoria o CPU, localizado todo ello en la clase `Proc_Stats`. La clase de comunicación `Comm` permite llevar a cabo el protocolo de comunicación entre el monitor del sistema y las islas de evolución arbitraria (actualmente estacionaria, generacional y celular). Asimismo, permite intercambiar de forma fiable y rápida tanto individuos como estadísticas entre los distintos procesos paralelos del sistema.

Es en esta clase donde se encierran las características paralelas relativas a la sincronización, apertura de puntos de acceso al servicio de comunicación (PAS), tabla de procesos del sistema y otros detalles que permiten un sistema flexible y rápido (usamos la interfaz *socket* [Come91]).

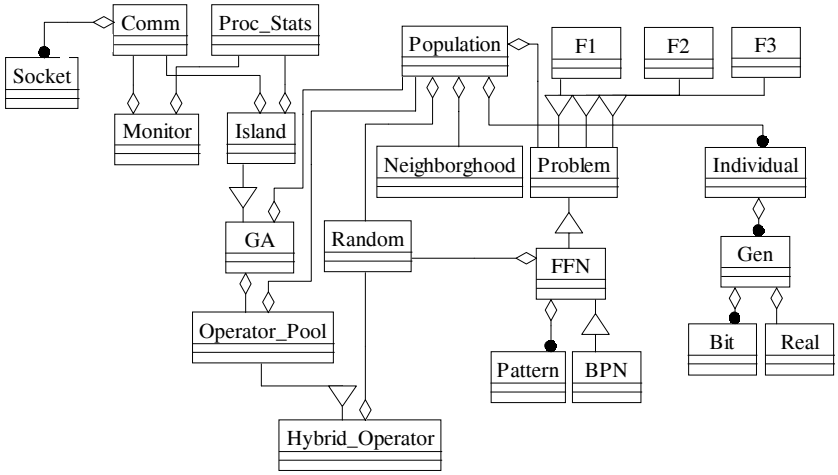


Figura 3.5. *Diseño de clases para xxGA.*

Por último, las clases del monitor (`Monitor`) e isla (`Island`) permiten derivar instancias que dan lugar a programas ejecutables para resolver un problema concreto. Ambas utilizan profusamente un sistema de configuración flexible basado en ficheros de texto con secciones para facilitar la ejecución y toma de resultados de pruebas independientes. La clase para el monitor no participa en la búsqueda de la solución ni provoca ningún tipo de retención en ella. Es la encargada de generar las islas, seguir paso a paso la búsqueda, e informar al usuario durante, y al final de la ejecución.

En la siguiente sección se presentan las características más importantes del sistema de comunicación que utilizan tanto el monitor como las islas para implementar *xx*GA.

3.5.2. El Sistema de Comunicación

En esta sección describimos el sistema de comunicación utilizado para implementar el sistema paralelo. Esta descripción se basa en la explicación del número y tipo de las *unidades de datos del protocolo* (UDPs, PDUs en inglés) o mensajes intercambiados por las dos entidades comunicantes. Son estas UDPs quienes permiten llevar a cabo la interconexión correcta entre las islas y entre islas y monitor (véase la Figura 3.6). Cualquier unidad de datos posee una cabecera identificando el *tipo* de mensaje, un campo de *longitud* del cuerpo y el *cuerpo* en sí mismo (vacío o) conteniendo los datos, parámetros o valores de control necesarios. Consúltese el Apéndice B para mayor detalle sobre cada tipo de UDP.

Orden	Longitud Mensaje	CUERPO DEL MENSAJE
4 [char]	*1 [unsigned long]*	$0 \leq$ *longitud* $\leq$ *MAX_MESSAGE_LENGTH*

Figura 3.6. *Formato General de una UDP.*

Para la ejecución paralela distribuida hemos definido la clase de comunicación (`Comm`) de forma que sus instancias permitan mantener el estado del sistema (*stateful client/server model*). La Tabla 3.2 describe los tipos de UDPs intercambiadas (en orden alfabético) por los extremos comunicantes, así como su significado. Explicamos en la propia tabla su funcionamiento, así como su clasificación dentro de uno de los cuatro grupos (según su uso): **a**: arranque, **t**: terminación, **m**: monitorización y **f**: funcionamiento del algoritmo.

TABLA 3.2. RESUMEN DE UDPS USADAS POR EL PROTOCOLO (I=ISLA, M=MONITOR)

PDU	De...	A...	Grupo	Explicación
CONF	M	I	a	*Contiene datos de configuración para dirigir la búsqueda*
ENDP	M	I	t	*El monitor pide a la isla iniciar el protocolo de desconexión*
ENDP	I	M	t	*La isla informa de su terminación indicando la razón e inicia descon.*
GAST	M	I	m	*El monitor solicita ser informado de la situación de la búsqueda*
GAST	I	M	m	*La isla actualiza sus estadísticas y las envía dentro de este mensaje*
GETI	M	I	m	*El monitor pide a una isla que le envíe una copia del individuo i-ésimo*
GEVA	M	I	m	*El monitor solicita el número de generaciones y evaluaciones hechas*
GEVA	I	M	m	*La isla informa al monitor del número de generaciones y evaluaciones*
INDI	I	M	f	*La isla envía en esta UDP una copia de uno de sus individuos*
INDI	I	I	f	*Una isla envía una copia de un individuo a su vecina en el anillo*
INIG	I	M	a	*La isla informa al monitor de sus datos para la comunicación*
KILL	M	I	t	*El monitor quiere que la isla termine definitivamente su ejecución*
LOAD	M	I	m	*El monitor solicita a la isla que lea su población desde un fichero*
PSTA	M	I	m	*El monitor solicita ser informado del uso de recursos (tiempos, ...)*
PSTA	I	M	m	*La isla informa al monitor del uso de recursos*
SAVE	M	I	m	*El monitor solicita a la isla que guarde su población en un fichero*
SCHS	M	I	m	*El monitor solicita información sobre el esquema bajo estudio*
SCHS	I	M	m	*La isla informa al monitor de cómo van las estadísticas sobre esquema*
SOLU	M	I	m	*El monitor solicita el mejor individuo encontrado hasta el momento*
STRT	M	I	a	*El monitor solicita que la isla comience la búsqueda evolutiva*

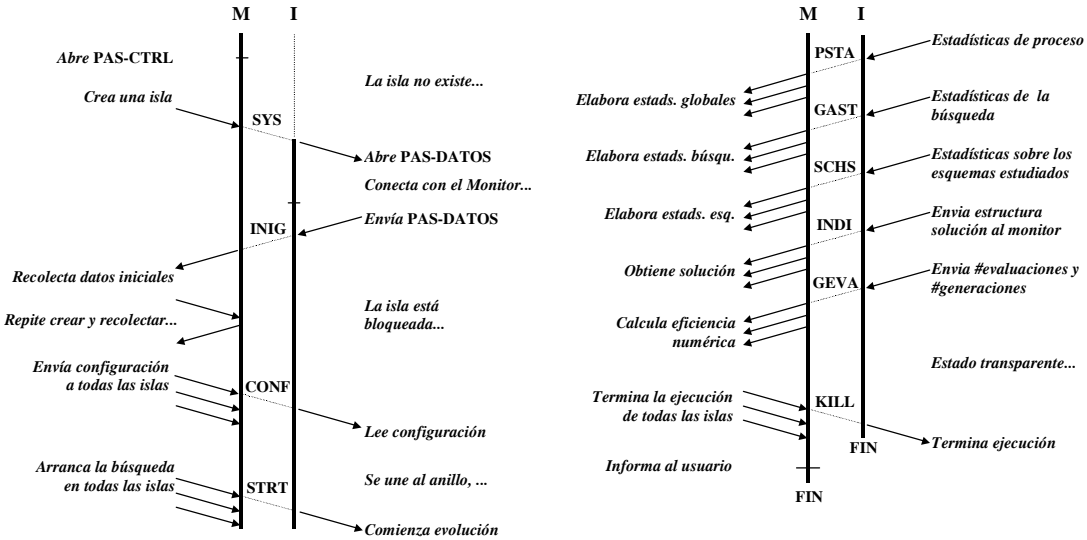
Igualmente importante para la definición del sistema de comunicación es la especificación de las fases de apertura de conexión, uso de dicha conexión y cierre de la misma. La Figura 3.7 describe el protocolo de conexión y desconexión. Como puede observarse, durante la conexión el monitor arranca los procesos isla y estos envían sus datos de configuración iniciales relativos a los puntos de conexión en los que intercambiar información con el monitor y sus vecinos. El monitor lee un fichero de configuración del sistema global (más detalles al final de esta sección) y después envía a las islas la configuración necesaria para realizar la búsqueda distribuida.

Cada isla puede utilizar su propio fichero de configuración local distinto del resto (heterogeneidad en la búsqueda). En él se describen los operadores y parámetros (por ejemplo probabilidades o forma de la rejilla en un cGA) que se usarán durante la búsqueda.

Cada isla, tras ser creada en el sistema (SYS), abre un punto de acceso a la red para conexión con su isla vecina (PAS-DATOS) e informa de ello al monitor tras conectar con él. El monitor recolecta los puntos de conexión para datos de cada isla y rellena su tabla de procesos con estos y otros datos referentes a la conexión y configuración de la búsqueda. El monitor informa seguidamente a cada isla de sus datos de trabajo y configuración de la isla vecina (CONF).

Las islas esperan entonces en una barrera de sincronización hasta que el monitor les indica (STRT) que comiencen a ejecutar cada una su modelo de evolución propio. Durante la búsqueda, el monitor queda en espera de mensajes de seguimiento o mensajes de terminación de las islas.

Si alguna de las islas termina por encontrar una solución antes del número máximo fijado de evaluaciones (ENDP), el monitor detiene a las restantes iniciando el protocolo de desconexión. Si ninguna isla encuentra una solución, o bien si la adecuación de la solución es desconocida, entonces ejecutarán completamente el número de ciclos especificados e irán terminando.



Protocolo de Conexión **Protocolo de Desconexión**

Figura 3.7. *Protocolos de conexión y desconexión del sistema implementado.*

El protocolo de desconexión supone que el monitor recogerá las estadísticas finales respecto a tiempo de ejecución, tiempo de búsqueda, consumo de memoria y otros datos de uso de recursos del sistema distribuido (PSTA). Posteriormente, el monitor recogerá de cada isla las estadísticas relativas a la búsqueda (adecuaciones finales, medias, desviaciones, ... todo ello incluido en GAST), las estadísticas sobre el estudio de esquemas (si procede, usando SCHS), el individuo solución (INDI) y el número de evaluaciones y generaciones (GEVA).

El monitor promedia y analiza todos los valores y, finalmente, indica a las islas que deben terminar su ejecución (KILL). El control de errores de comunicación es bastante estricto, de forma que si la ejecución termina correctamente es muy improbable que se haya producido un error no detectado (los mensajes inesperados, incorrectos, operaciones aritméticas incorrectas, petición de memoria, etc. están controlados en el código).

Durante su ejecución, las islas realizan puntos de chequeo (*checkpoints*) para decidir si enviar datos al monitor, volcar en fichero, etc... según el tipo de ejecución. Debe notarse que en general cualquier envío de datos es asíncrono y por tanto no retiene ni provoca cuellos de botella en la ejecución. Si una isla termina antes que el resto, entonces deja de trabajar y únicamente pasa cualquier mensaje a la isla vecina sin interpretarlo (en espera de un mensaje `KILL`). De otro modo podrían aparecer interbloqueos, sobre todo en la recepción de emigrantes en su isla vecina.

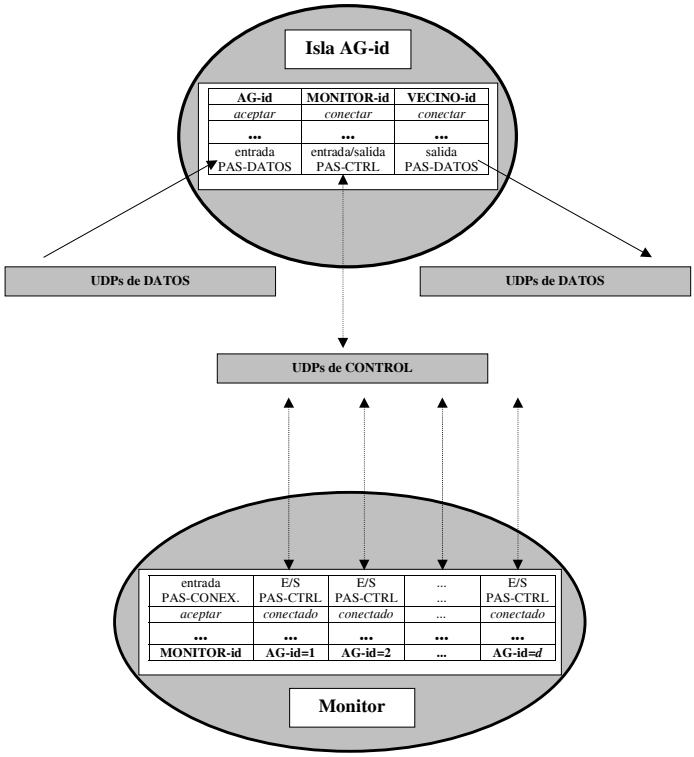


Figura 3.8. *Escenario después de la conexión.*

Puede observarse en la figura de la izquierda que tanto el monitor como las islas disponen de una tabla de procesos para gestionar las comunicaciones (clase `Comm`).

Sólo la tabla del monitor está completa. Cada isla únicamente utiliza las entradas relativas a sí misma (para esperar emigrantes), la de su vecina (para enviar emigrantes) y la del monitor (para mensajes de control).

El concepto de *punto de acceso al servicio* (PAS) típico de redes de comunicación se utiliza aquí para definir un punto a través del cual enviar/recibir datos por la red.

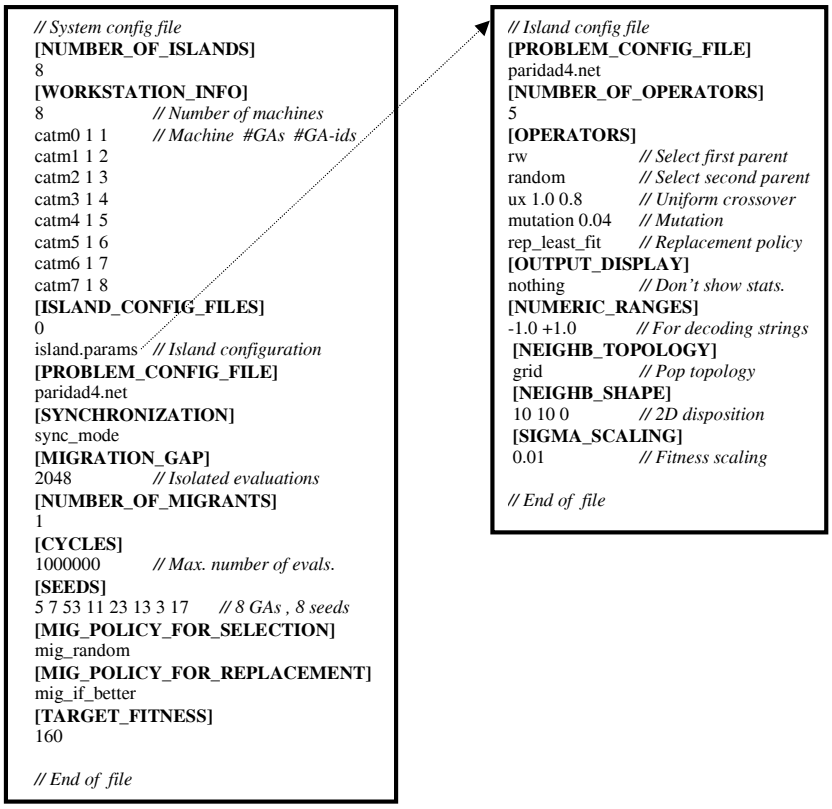
Nuestra aplicación utiliza la interfaz de programación *BSD socket*. Usamos el protocolo TCP (nivel de transporte -4- del modelo OSI) para una transmisión segura.

El uso de la interfaz *socket* se debe a nuestro interés en mantener tan rápida como podamos la implementación. Otras alternativas como PVM [GBDJM94], MPI [DEC94], Java-RMI [OH97], etc.. son también posibles, pero en general están basadas sobre conceptos similares al de *socket* y cargan al sistema con servidores propios, realizan operaciones de empaquetado, etiquetado, notificaciones internas, ... que preferimos evitar en aras de una mayor eficiencia final.

Evidentemente, el uso directo de *sockets* requiere un mayor esfuerzo de programación, pero las clases `Socket` y `Comm` encapsulan todas las operaciones de apertura y cierre de PASs, así como el envío y recepción de cualquier tipo de mensajes, y todo ello de forma síncrona o asíncrona. El resto de clases únicamente utilizan los servicios de éstas sin importar cómo están programados. Adicionalmente, es importante señalar que, debido a la definición original de TCP, son necesarios algunos cambios en su funcionamiento básico para que no se haga esperar innecesariamente a los mensajes.

Igualmente, merecen mención los problemas potenciales derivados de la aparición de efectos laterales insoslayables debidos a que tanto el generador de números aleatorios de C++ como la tabla de descriptores de *sockets* son comunes a una aplicación, independientemente del diseño en clases realizado. Esto impide muchas veces que los constructores y destructores puedan seguir un comportamiento ortodoxo. De nuevo, se trata de un compromiso entre ejecución eficiente y correcta frente a uso ortodoxo de la programación orientada a objetos.

Para terminar esta sección, simplemente presentaremos un ejemplo del fichero de configuración del sistema y otro de la configuración de una isla (Figura 3.9). Para más información sobre el formato y posibilidades de configuración consulte el Apéndice B.



```
// System config file
[NUMBER_OF_ISLANDS]
8
[WORKSTATION_INFO]
8                   // Number of machines
catm0 1 1           // Machine  #GAs  #GA-ids
catm1 1 2
catm2 1 3
catm3 1 4
catm4 1 5
catm5 1 6
catm6 1 7
catm7 1 8
[ISLAND_CONFIG_FILES]
0
island.params       // Island configuration
[PROBLEM_CONFIG_FILE]
paridad4.net
[SYNCHRONIZATION]
sync_mode
[MIGRATION_GAP]
2048                // Isolated evaluations
[NUMBER_OF_MIGRANTS]
1
[CYCLES]
1000000             // Max. number of evals.
[SEEDS]
5 7 53 11 23 13 3 17      // 8 GAs , 8 seeds
[MIG_POLICY_FOR_SELECTION]
mig_random
[MIG_POLICY_FOR_REPLACEMENT]
mig_if_better
[TARGET_FITNESS]
160

// End of file
```

```
// Island config file
[PROBLEM_CONFIG_FILE]
paridad4.net
[NUMBER_OF_OPERATORS]
5
[OPERATORS]
rw                  // Select first parent
random              // Select second parent
ux 1.0 0.8          // Uniform crossover
mutation 0.04       // Mutation
rep_least_fit       // Replacement policy
[OUTPUT_DISPLAY]
nothing             // Don't show stats.
[NUMERIC_RANGES]
-1.0 +1.0           // For decoding strings
[NEIGHB_TOPOLOGY]
grid                // Pop topology
[NEIGHB_SHAPE]
10 10 0             // 2D disposition
[SIGMA_SCALING]
0.01                // Fitness scaling

// End of file
```

Figura 3.9. *Ejemplo del fichero de configuración del sistema* (izqda.) *y de una isla* (dcha.).

Puede observarse que ambos siguen el mismo esquema de trabajo. Cada fichero muestra secciones encabezadas por una etiqueta identificativa del valor que les sigue. Los ficheros contienen texto ASCII y pueden crearse o modificarse con cualquier editor de texto normal. El sistema de lectura controla que los valores leídos tengan sentido por sí mismos (probabilidades no negativas y no mayores de 1.0, por ejemplo) y su relación con el resto de parámetros (mismo número de semillas aleatorias que de islas, por ejemplo). Podemos configurar de forma flexible el número de máquinas, el esquema de migración, los operadores, ...

Es bastante fácil y directo con este sistema de ficheros de configuración cambiar el algoritmo usado, sus parámetros, el número de máquinas, el problema bajo estudio, y muchos otros aspectos que determinan la calidad del sistema *software* y la extracción de conclusiones. La adición de una interfaz gráfica es directa, limitándose a generar como salida este tipo de ficheros.

3.5.3. Consideraciones Globales sobre la Implementación

En esta sección pretendemos demostrar que el diseño e implementación de los modelos secuenciales y también paralelos sobre un ordenador tienen cierto impacto en la calidad de la búsqueda. Básicamente nos centraremos en tres aspectos fundamentales:

- Diseño y relación entre las clases de objetos.
- Implementación de la población.
- Almacén de operadores.

Respecto al diseño de clases, está fuera de duda la utilidad y flexibilidad de usar este tipo de paradigma para obtener un algoritmo seguro, reutilizable y del que se pueda extraer fácilmente información sobre la búsqueda. El problema que queremos señalar es el compromiso entre todas estas ventajas y la eficiencia. La Figura 3.10 muestra los resultados (20 ejecuciones sobre un procesador UltraSparc 1) de usar una implementación orientada a objetos en JAVA de *xx*GA (parámetros en Tabla 3.3).

TABLA 3.3. PARÁMETROS MMDP8 Y SPH3 PARA MEDIR LA EFICIENCIA USANDO JAVA

Problema MMDP8		**Problema SPH3**	
Parámetros	**Modelos**	**Parámetros**	**Modelos**
l 8x6=48 bits	*xx*GA(1,128,_,_,ss,ifb)	*l* 3x32=96 bits	*xx*GA(1,128,_,_,ss,ifb)
μ 128	*xx*GA(1,128,_,_,sus,ifb)	μ 128	*xx*GA(1,128,_,_,sus,ifb)
Pc 1	*xx*GA(1,2x64,_,_,cel,ifb)	*Pc* 1	*xx*GA(1,2x64,_,_,cel,ifb)
Pm 0.05		*Pm* 0.01	
max. evals. 100000		*max. evals.* 100000	

Los tres algoritmos, como puede observarse muestran un tiempo innecesariamente alto e indeseable en todos los modelos para ambos problemas. Esto se debe a que la herencia, usada de forma libre, produce constantes llamadas a métodos internos de generación (*clonación*) y destrucción de instancias locales en cadena.

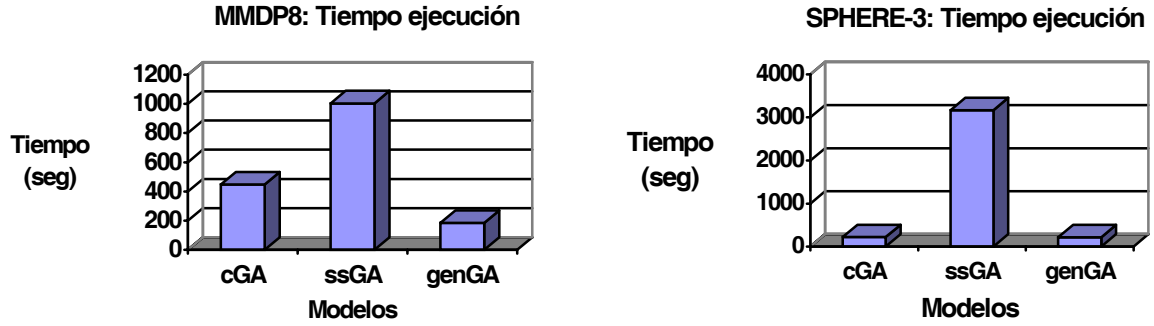


Figura 3.10. *Tiempo en solucionar el problema MMDP y la esfera (*8 *y* 3 *genes, respect.).*

En la figura anterior aparece otro problema potencial, esta vez respecto a la forma de implementar la población. Obsérvese cómo el modelo estacionario (ssGA) requiere unos tiempos muy por encima del modelo celular o generacional. Esto no es nada usual, y se debe al trabajo con una estructura estática y al requisito de mantener la población ordenada en cada paso para que la selección, estadísticas, etc. funcionen correctamente (por ejemplo para usar *ranking*).

Esto nos da pie a discutir el segundo punto mencionado en la presentación de esta sección. Para diseñar una población eficiente deberíamos primero definir las operaciones más frecuentemente realizadas. Parece claro que en general las operaciones sobre la población son el acceso directo y/o secuencial y también las adiciones y borrados continuos. Las alternativas de implementación son utilizar un vector estático de individuos o una lista dinámica. Una tercera opción es el uso de cursores, pero el resultado es equivalente a usar la lista y es el usuario el que debe manejar a mano las referencias.

La ventaja a priori de un vector estático es el acceso directo a sus elementos. El acceso secuencial es igual de eficiente en ambos casos. Si la población debe estar ordenada, un vector presenta ventajas al realizarse, por ejemplo, una ordenación binaria más rápidamente que sobre una lista (por permitir accesos directos). El punto claro donde reside la diferencia es en la inserción y el borrado de individuos. En el caso de usar un vector estático los constantes desplazamientos a que se somete a la población hacen intuir que en este aspecto la lista dinámica será más eficiente.

Presentados los puntos fuertes y débiles de cada uno, queda preguntarse cuáles son las operaciones más frecuentes. La respuesta es la inserción y el borrado. La Figura 3.11 demuestra que usar una lista de individuos es la decisión más eficiente para un algoritmo genético. Incluso si la población debe estar ordenada podemos insertar al principio de forma ordenada por adecuación y realizar también las posteriores inserciones ordenadamente, lo que evita la sobrecarga de utilizar constantemente algún tipo de algoritmo de ordenación.

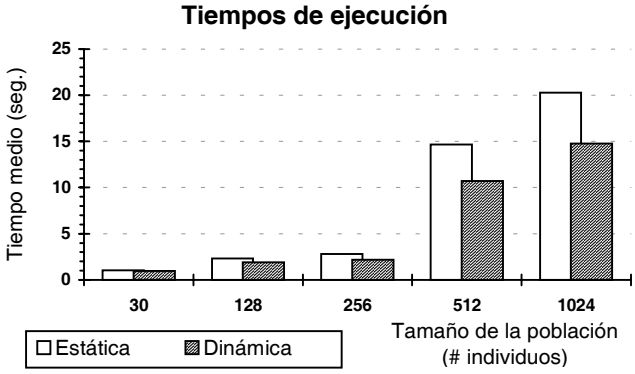


Figura 3.11. *Relación entre trabajar con una población estática o dinámica en C para el problema de la mochila (apéndice A). Presentamos la media de* 10 *ejecuciones independientes.*

Por tanto, debemos estar prevenidos frente a implementaciones estándares de algoritmos genéticos en las que se utilicen estructuras de datos estáticas para la población, ya que los resultados aparecerán tras un tiempo de ejecución innecesariamente alto.

Un tema relacionado con la orientación a objetos y con la población es la implementación de la memoria privada necesaria para las clases del genotipo. En general, la clase base de los genes debe contener directamente el valor del gen, ya que si contiene un apuntador (como es típico en muchas implementaciones de clases genéricas en programación orientada a objetos), éste puede ocupar más memoria que el propio valor apuntado.

Puede comprobarse gráficamente este efecto en la Figura 3.12. Si se crea una clase base de genes binarios usando para almacenar cada alelo un carácter (`char`) ahorraremos mucho más espacio que si se crea la clase base de los alelos binarios como un apuntador a un carácter (propio de una implementación ortodoxa en C++ que crea y destruye instancias, por ejemplo).

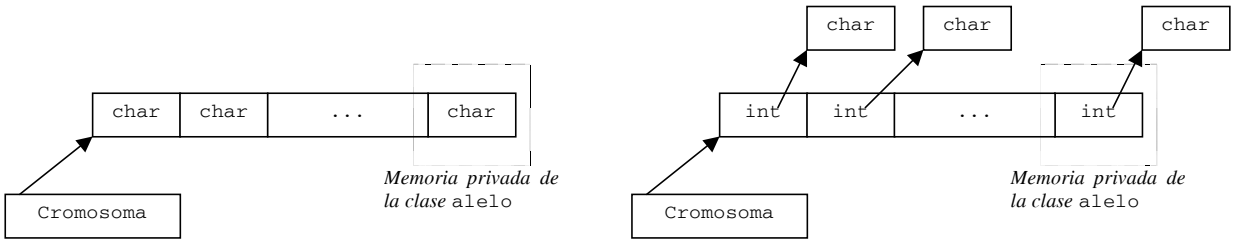


Figura 3.12. *Clase para una cadena en la que la clase componente para los alelos utiliza como memoria privada un carácter* (izqda.) *o bien un apuntador a carácter* (dcha.).

Por último, nos queda discutir la implementación del conjunto de operadores. Parece evidente que su implementación como funciones aisladas sin relación es poco estructurada. Con ello se tiende a generar errores y a tomar decisiones locales en cada operador que no conducen a su estandarización; además, la implementación no se asemejaría a cómo aparecen los operadores en el algoritmo evolutivo. Es por esto que resulta más conveniente implementar el grupo de operadores bajo un diseño en clases de objetos común, de forma que el concepto formal de ciclo reproductor teórico exista en la práctica (Figura 3.13).

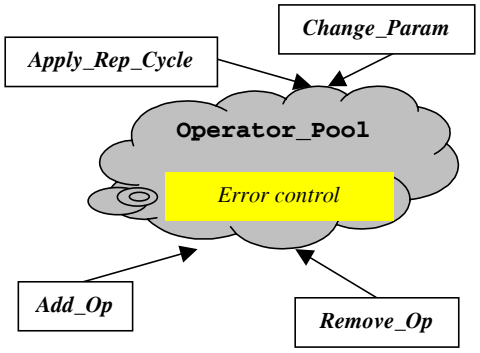


Figura 3.13. *Noción de ciclo reproductor como clase de objetos* (`Operator_Pool`).

De esta forma podremos tener instancias de ciclos reproductores a las que es posible añadir, modificar o extraer operadores en tiempo de ejecución, dando lugar a una implementación realmente eficiente, flexible, compacta y segura (control de errores). Añadiendo una clase para operadores híbridos como hacemos en el sistema de la Figura 3.5 (`Hybrid_Operator`) permitimos la posterior inclusión en el algoritmo de técnicas de búsqueda no genética que permitan mejorar el funcionamiento del sistema. De esta forma podemos incorporar operadores como el enfriamiento simulado, búsqueda tabú, o cualquier tipo de búsqueda local en el sistema sin alterar su estructura.

3.6. Relación de *xx*GA con los Marcos de Programación

En este trabajo se ha realizado un esfuerzo de generalización y unificación de modelos AGP tanto desde el punto de vista algorítmico como desde el punto de vista de la ingeniería del *software*. En este segundo aspecto las posibilidades de extensión y abstracción adicionales, entre otras, son aún mayores de las presentadas hasta el momento. Para demostrarlo pondremos en contacto las técnicas e implementaciones realizadas con un concepto de muy alto nivel orientado a promover el uso de sistemas paralelos entre todo tipo de usuarios, expertos o no.

Nos estamos refiriendo a los *marcos de programación* (*programming frames*). En todas las disciplinas científicas es importante el acercamiento de las tecnologías desarrolladas tanto a usuarios finales como a otros investigadores que no conocen (o no quieren conocer) los detalles propios de las aportaciones realizadas, y que sin embargo quieren usarlas. Los marcos de programación [RP97] son esqueletos de trabajo, en nuestro caso para utilizar algoritmos sobre máquinas de memoria distribuida, a los que el usuario debe aportar únicamente información sobre el problema que desea resolver. Un **marco** contiene en general tres tipos de información: *algoritmos básicos*, *estructuras de datos* y los *marcos de programación* en sí mismos.

Los marcos poseen muchas características en común con la programación orientada a objetos (POO) que hemos utilizado para diseñar nuestras implementaciones. La característica más sobresaliente que extiende las ventajas de la POO es la *automatización* del proceso de generación de nuevos algoritmos ya dispuestos para ejecutar y analizar en una máquina de memoria distribuida, y el uso de una interfaz gráfica para el usuario. Para aclarar más el funcionamiento de los marcos pasamos a discutir sus principales características:

Reusabilidad: Posibilidad de reutilizar la definición del problema (que el usuario aportará) en otras plataformas de ejecución distintas.

Portabilidad: Posibilidad de ejecutar con un esfuerzo mínimo de adaptación todo el marco en nuevos sistemas con mecanismos de comunicación distintos (MPI, PVM, *sockets*, ...).

Técnicas de última generación: Posibilidad de mantener actualizado el conjunto de técnicas usadas para las estructuras de datos y para los algoritmos.

Eficiencia: Mantenimiento de un alto nivel de eficiencia en la ejecución del resultado final (algoritmo) debido al refinamiento de sus componentes y al uso de mecanismos de comunicación propios de la máquina que lo ejecuta.

Fácil aprendizaje: Facilidad en el uso del marco resultante para que el usuario pueda disponer tanto de un algoritmo útil como de una implementación eficiente de dicho algoritmo.

Interfaz gráfica: Generación automatizada de una interfaz adecuada para manipular los parámetros del problema y visualizar los resultados durante y después de la ejecución.

El usuario únicamente rellena los huecos dejados por un experto en el dominio del algoritmo (en nuestro caso algoritmos genéticos paralelos distribuidos) y el marco genera un programa ejecutable que atiende a dichos parámetros y permite su estudio y seguimiento. Un segundo experto es necesario para especificar internamente y de forma oculta al usuario el mejor mecanismo de paralelización del algoritmo resultante.

Podría considerarse que el experto en el algoritmo diseña la interfaz con los elementos definibles por el usuario, mientras que el experto en paralelización define los aspectos de eficiencia según el sistema paralelo considerado. Ambos expertos deben acordar el mecanismo total de fusión de sus respectivos dominios en el marco final. El usuario recibe un sistema *software* con tres características muy deseables: fácil de usar, genérico y eficiente.

Tanto el sistema en clases del modelo *xx*GA como su filosofía de generalidad encajan en la idea de los marcos. Sin embargo, el camino que se debe recorrer en un marco hasta obtener un algoritmo ejecutable es más largo que el uso de un conjunto de clases de objetos como los de la Sección 3.5.1. Para generar un algoritmo ejecutable se necesitan tres niveles de especificación:

- Nivel **abstracto**.
- Nivel de **instancia**.
- Nivel de **implementación**.

Cada nivel se compone de un conjunto de ficheros propios que son compilados para dar lugar a ficheros útiles para el nivel inferior. La especificación del nivel abstracto lo proporciona un experto en el algoritmo. Para ello determina y documenta (incluso en formato HTML) los parámetros y técnicas relevantes para el problema. En el nivel de instancia se ligan los valores concretos seleccionados por el usuario a los parámetros del marco, es decir, se genera una instancia concreta del marco para las necesidades del usuario. Finalmente, los conocimientos de los expertos en el problema/algoritmo y en la máquina/comunicación destino se concretan en un conjunto de ficheros conteniendo reglas y facilidades para enlazar todos los ficheros necesarios. Es en este nivel donde se utiliza una implementación concreta de algoritmo como la nuestra.

Siguiendo este esquema genérico se han construido múltiples marcos, por ejemplo, un marco dedicado a métodos de elementos finitos [RP97] o para algoritmos genéticos paralelos [Peti97]. Nuestro interés se centra en este último caso. El marco mencionado utiliza la biblioteca PGAPack [Levi96] para el nivel de implementación. Esta biblioteca tiene múltiples facilidades para ejecución paralela usando MPI, codificaciones, operadores genéticos, incorporación de heurísticos e interfaz de uso. Además de la especificación abstracta se puede dar una instanciación, tanto genérica usando **FIT** (*Frame Instantiator Tool*), como propia, usando una GUI específica para PGAPack llamada **xpga**.

PGAPack permite un paralelismo global en forma maestro/esclavo en el que la evaluación de la función de adecuación se realiza en paralelo en los procesadores disponibles. Nuestro modelo distribuido es algo más genérico en términos de los operadores, ya que PGAPack considera cada operador como una entidad separada. Además, el esquema de paralelización de nuestro modelo es considerablemente más rico y complejo (poblaciones paralelas estructuradas o en panmixia, política de migración, etc.).

Para demostrar la capacidad de extensión de nuestro modelo y su relación con los marcos proponemos en las siguientes secciones una descripción textual de los niveles abstracto y de instancia. Para el nivel de implementación podría usarse la implementación existente del modelo distribuido usando *sockets*. La extensión a otros mecanismos de comunicación se reduce a modificaciones (no menores pero sí simples y localizadas) en las clases `Comm` y `Socket`. Las extensiones algorítmicas están claramente guiadas por el diseño existente.

3.6.1. Nivel Abstracto

Para realizar un nivel abstracto para un marco con *xx*GA debemos definir las operaciones y propiedades de alto nivel del algoritmo. La notación que se suele usar es similar a la de C y es muy portable porque no contiene elementos dependientes del sistema. La documentación HTML no se incluye en la tesis debido a que estamos realizando una propuesta y no un marco real. A pesar del elevado número de parámetros libres podemos hacer una estructuración en familias:

Problema:

Contienen información sobre la codificación usada y el mecanismo de evaluación utilizado. Consideramos siempre un problema de maximización.

Inicialización:

Mecanismo de generación de la población inicial. Por defecto la creación es aleatoria, pero el usuario podría utilizar un mecanismo propio.

Operadores:

Ya que hemos introducido en *xx*GA el concepto de *operator pool* (conjunto de operadores) el nivel abstracto podría permitir determinar el número de operadores y el tipo. En general esto supone en un algoritmo genético al menos identificar la selección, cruce y mutación elegidos. Existen varias técnicas predefinidas para cualquiera de estos operadores, pero además el usuario puede proporcionar técnicas nuevas. Sería posible además especificar el orden de ejecución de estos operadores, e incluso podría extenderse a un orden de ejecución y parámetros de ejecución dinámicos (proporcionando un criterio de modificación apropiado).

Evolución de cada isla:

En este grupo englobamos las técnicas básicas para la evolución de cada isla. En principio el marco que estamos describiendo se ofrece para islas idénticas, aunque es perfectamente posible usar parámetros distintos en cada una de ellas con nuestra implementación. Entre ellos destaca el esquema de evolución estacionario/generacional/celular y parámetros como el tipo de vecindario y/o topología del modelo celular.

Política de migración:

En este conjunto de parámetros encontramos algunos como la frecuencia de migración, el número de individuos que migran, el tipo de selección del emigrante, el tipo de reemplazo en la población receptora y el tipo de sincronización entre islas. Debemos aclarar que esta última es una propiedad del modelo aunque internamente exista un concepto equivalente que dependa del tipo de implementación. De hecho, incluso podemos simular en monoprocesador un algoritmo síncrono/asíncrono. Hacemos esta aclaración para dejar patente que esto no entra en conflicto con el objetivo de independencia de la máquina del nivel abstracto en el que nos movemos.

Adecuación:

Este grupo incluye operaciones sobre el valor de adecuación de los individuos tales como el escalado lineal, sigma o de otro tipo, encaminadas a evitar la convergencia prematura.

Reinicio:

El sistema puede cargar con cualquier frecuencia deseable una nueva población desde fichero generada atendiendo a cualquier criterio que se considere oportuno.

Criterio de terminación:

Básicamente se puede elegir entre ejecutar por un número predeterminado de evaluaciones el algoritmo paralelo o bien hasta encontrar una solución. Otros criterios son posibles, pero aún no está contemplada una modificación simple en la implementación existente.

Opciones de informe:

Múltiples tipos de estadísticas son posibles en este grupo: *on-line*, *off-line*, estudios sobre esquemas, tiempos de ejecución, diferencia entre medias sucesivas, entropía de la población, distancia de Hamming de los individuos, etc.

Misceláneas:

Activación del control de errores en tiempo de ejecución, visualización de cadenas dirigida por el usuario, versión de la biblioteca, ...

Una posible especificación del nivel abstracto del marco puede ser la siguiente:

```
/*
       * Abstract frame xxGA (reduced version)
       *
       * Enrique Alba, 1998
       * Departamento de Lenguajes y Ciencias de la Computación
       * Universidad de Málaga
*/
     ABSTRACT FRAME xxGA;

     DECLARATION

         /* Enumeration types */
         TYPE Boolean                = enum {False, True};
         TYPE DataTypeEnum           = enum {Binary, Gray, Real};
         TYPE EvolTypeEnum           = enum {steady_state, generational};
         TYPE SyncTypeEnum           = enum {sync, async};
         TYPE TopologyTypeEnum       = enum {grid, uniDring, biDring, tree, all};
         TYPE NeighborhoodTypeEnum   = enum {NEWS, C9, C25};
         TYPE ReplacementTypeEnum    = enum {ReplAlways, ReplIfBetter, ReplRandom, ReplWorstOfN};
         TYPE CrossoverTypeEnum      = enum {SPX, SPX1, DPX, DPX1, UX, AX, CrossoverUser};
         TYPE MutationTypeEnum       = enum {MutationConstant, MutationRange, MutationUser};
         TYPE FitnessMappingTypeEnum = enum {FitnessRaw, FitnessLinearSc, FitnessSigmaSc};
         TYPE SelectionTypeEnum      = enum {SelectionProportional, SelectionSUS,
                                             SelectionRanking,      SelectionTournament,
                                             SelectionRandom,       SelectionBest};
         TYPE InitializationTypeEnum = enum {InitBinRandom, InitRealRandom, InitFromFile};

         /* Out types (private) */
         OUT TYPE xxGAContext;
         OUT TYPE FILE;

         /* Pointer equivalent types */
         TYPE xxGAContextPtr = xxGAContext*;
         TYPE FILEPtr        = FILE*;
         TYPE charPtr        = char*;

         /* Problem parameters */
         CONSTANT DataTypeEnum DataType;
         CONSTANT int GeneNumber;
         CONSTANT int GeneLength;
         FUNCTION double Evaluate (xxGAContextPtr ctx, int p, int pop);
         OPTIONAL STATEMENT UserCode ();

         /* Population and replacement */
         OPTIONAL CONSTANT int IslandPopSize                        = "100";
         OPTIONAL CONSTANT int NumberOfIslands                      = "8";
         OPTIONAL CONSTANT ReplacementTypeEnum ReplacementType      = "ReplIfBetter";
```

```
        /* Initialization */
        OPTIONAL CONSTANT  InitializationTypeEnum  InitializationType;
        OPTIONAL CONSTANT  double BinInitProbBit1                    = "0.5";
        OPTIONAL CONSTANT  double MinInitRangeLow                    = "0.0";
        OPTIONAL CONSTANT  double MaxInitRangeHigh                   = "1.0";
        OPTIONAL PROCEDURE Initialize (xxGAContextPtr ctx, int p, int pop);
        /* Selection */
        OPTIONAL CONSTANT  SelectionTypeEnum SelFirstParent          = "SelectionTournament";
        OPTIONAL CONSTANT  SelectionTypeEnum SelSecondParent         = "SelectionTournament";
        OPTIONAL CONSTANT  int TournamentSize                        = "2";
        OPTIONAL CONSTANT  RankingBias                               = "1.5";
        /* Crossover */
        OPTIONAL CONSTANT  CrossoverTypeEnum CrossoverType           = "DPX1";
        OPTIONAL CONSTANT  double CrossoverProb                      = "0.85";
        OPTIONAL CONSTANT  double UniformCrossoverProb               = "0.8";
        OPTIONAL PROCEDURE Crossover (xxGAContextPtr ctx, int c1, int c2, int c_pop,
                                                          int p1, int p2, int p_pop);
        /* Mutation */
        OPTIONAL CONSTANT  double MutationProb                       = "0.01";
        OPTIONAL CONSTANT  MutationTypeEnum MutationType             = "MutationConstant";
        OPTIONAL CONSTANT  Boolean BoundMutation                     = "True";
        OPTIONAL FUNCTION  int Mutation (xGAContextPtr ctx, int p, int pop, double mr);
        /* User operator */
        OPTIONAL FUNCTION  int Operator (xxGAContextPtr ctx, int c1, int c2, int c_pop,
                                                           int p1, int p2, int p_pop);
        /* Island parameters */
        OPTIONAL CONSTANT  EvolTypeEnum BasicIslandEvolution         = "steady_state";
        OPTIONAL CONSTANT  TopologyTypeEnum Topology                 = "grid";
        OPTIONAL CONSTANT  int GridHeight                            = "10";
        OPTIONAL CONSTANT  int GridWidth                             = "10";
        OPTIONAL CONSTANT  NeighborhoodTypeEnum Neighborhood         = "NEWS";
        /* Migration policy */
        OPTIONAL CONSTANT  int MigrationFrequency                    = "32";
        OPTIONAL CONSTANT  int MigrationRate                         = "1";
        OPTIONAL CONSTANT  SelectionTypeEnum   MigrantSelection      = "SelectionRandom";
        OPTIONAL CONSTANT  ReplacementTypeEnum MigrantReplacement    = "ReplIfBetter";
        OPTIONAL CONSTANT  SyncTypeEnum        SyncMode              = "async";
        /* Fitness */
        OPTIONAL CONSTANT  FitnessMappingTypeEnum FitnessMappingType = "FitnessRaw";
        OPTIONAL CONSTANT  double DeltaFactor                        = "0.0001";
        OPTIONAL CONSTANT  double SigmaFactor                        = "2.0";
        /* Restart */
        OPTIONAL CONSTANT  Boolean ApplyRestartOperator              = "False";
        OPTIONAL CONSTANT  int RestartFrequency                      = "9999999";
        /* Stopping criteria */
        OPTIONAL CONSTANT  Boolean UserDefinedStopRule               = "False";
        OPTIONAL CONSTANT  Boolean StopMaxEvals                      = "True";
        OPTIONAL CONSTANT  int MaxEvalsValue                         = "1000";
        OPTIONAL FUNCTION  int StopCondition (xxGAContextPtr ctx);
        /* Report options */
        OPTIONAL CONSTANT  Boolean OnlineAnalysis                    = "False";
        OPTIONAL CONSTANT  Boolean OfflineAnalysis                   = "False";
        OPTIONAL CONSTANT  Boolean WorstEvaluation                   = "False";
        OPTIONAL CONSTANT  Boolean AverageEvaluation                 = "False";
        OPTIONAL CONSTANT  Boolean SigmaAnalysis                     = "False";
        OPTIONAL CONSTANT  Boolean HammingDistance                   = "False";
        OPTIONAL CONSTANT  Boolean MeanEntropy                       = "False";
        OPTIONAL CONSTANT  Boolean SchemaActualInst                  = "False";
        OPTIONAL CONSTANT  Boolean SchemaPredictedInst               = "False";
        OPTIONAL CONSTANT  Boolean SchemaAvgFitness                  = "False";
        OPTIONAL CONSTANT  Boolean StringItself                      = "False";
        OPTIONAL CONSTANT  Boolean SavePopToFile                     = "False";
        OPTIONAL CONSTANT  Boolean LoadPopFromFile                   = "False";
        OPTIONAL CONSTANT  int     FrequencyValue                    = "5120";
        OPTIONAL PROCEDURE SetSchemaToStudy(xxGAContextPtr ctx, CharPtr schema);
        OPTIONAL PROCEDURE EndOfGeneration (xxGAContextPtr ctx);
        /* Miscellaneous */
        OPTIONAL CONSTANT  int     RandomSeed;
        OPTIONAL CONSTANT  Boolean NumberOfEvaluations               = "True";
        OPTIONAL CONSTANT  Boolean ElapsedRunTime                    = "True";
        OPTIONAL CONSTANT  Boolean PrintxxGAVersionNumber            = "False";
        OPTIONAL CONSTANT  Boolean PrintxxGAFrameVersionNumber       = "False";
        OPTIONAL PROCEDURE PrintString (xxGAContextPtr ctx, FILEPtr f, int p, int pop);

    DOCUMENTATION
        DOC(xxGA) = URL "http://???//xxGA.html";

    END xxGA.
```

Puede observarse en el nivel asbtracto anterior que la mayoría de las características mencionadas para *xx*GA están presentes listas para que el usuario las instancie en el marco. En primer lugar, el usuario debe proporcionar la función de evaluación `Evaluate` así como el tamaño del problema: número de genes (`GeneNumber`) y longitud de cada gen (`GeneLength`). También debe especificar el tipo base de las estructuras (`DataType`).

Por defecto la población se genera aleatoriamente (usando un rango de valores para las variables codificadas) y se aplica un bucle reproductor típico consistente en selección, cruce, mutación y reemplazo. Puede observarse que distinguimos entre la selección de cada uno de los dos padres para generar una nueva estructura: `SelFirstParent` y `SelSecondParent`. Varios de los tipos de selección más populares están disponibles: proporcional a la adecuación, torneo, seleccionar al mejor individuo, uno aleatorio, etc. (`SelectionTypeEnum`). Es interesante poder decidir por separado el mecanismo de selección de cada padre porque puede ser útil utilizar dos mecanismos distintos (véase el Capítulo 4).

En el caso de un modelo estacionario o generacional tradicional la selección actuará sobre toda la población de la isla, mientras que en el caso celular (reconocible porque se especifica topología -`TopologyTypeEnum`- y vecindario -`NeighborhoodTypeEnum`-) la selección sólo actúa sobre el vecindario de cada cadena.

Los operadores de cruce y mutación son los descritos en la tesis: cruce de uno o dos puntos, uniforme o aritmético (`CrossoverTypeEnum`), así como mutación binaria o real en cierto rango (`MutationTypeEnum`). El usuario puede además especificar métodos específicos para el problema para ambos operadores (procedimiento `Crossover` y función `Mutation`). Además, se puede especificar un nuevo operador definiendo la función `Operator`.

El mecanismo de reemplazo considerado es uno de entre los disponibles en `ReplacementTypeEnum`: reemplazo siempre del peor (o del nodo considerado si es un modelo celular), reemplazo sólo si es mejor que el designado para ser reemplazado, etc.

Además de poder especificar el tipo de evolución de cada isla podemos proporcionar una política de migración para cada problema: frecuencia de migración (`MigrationFrequency`), número de individuos que migran (`MigrationRate`), qué individuo será considerado como emigrante (`MigrantSelection`), cuál será reemplazado con un emigrante entrante (`MigrantReplacement`) o si deseamos un modelo síncrono o asíncrono (`SyncTypeEnum`).

Existe un conjunto de operaciones interesantes que podemos realizar destinadas a mejorar la búsqueda y el análisis de los resultados. Podemos manipular la adecuación haciendo escalado para separar entre sí los valores y facilitar la selección (`FitnessMappingTypeEnum`). También podemos cargar o guardar poblaciones en disco en cualquier momento de la ejecución. El criterio de finalización puede modificarse, y es posible también obtener trazas de la adecuación media, mejor, entropía media, distancia de Hamming, etc. indicando además la frecuencia con que esto se desea (`FrequencyValue`). Por otro lado, es también posible estudiar la adecuación media de un esquema en particular (`SetSchemaToStudy`), así como el número de instancias reales y predichas por el teorema de los esquemas (véanse "informes").

Es posible también ser informado del tiempo de ejecución (`ElapsedRunTime`) y del número de evaluaciones realizadas para alcanzar la solución (`NumberOfEvaluations`), ambos importantes en un algoritmo genético paralelo.

3.6.2. Nivel de Instancia

En el nivel de instancia de un marco se pide al usuario que proporcione valores para los parámetros abstractos, convirtiendo así al AGP genérico en uno específico para el problema. Los valores asignados se proporcionan en una sintaxis similar a la de ANSI C sin construcciones paralelas. Esta es la razón de que el usuario no necesite conocer detalles de programación paralela.

Existen tres formas de generar una instancia:

- Usando un *lenguaje textual* en el que (básicamente) se asignan valores a parámetros.
- Usando una *interfaz gráfica propia* del marco, diseñada *adhoc* (como **xpga** para PGAPack).
- Usando **FIT** para generar gráficamente la instancia a partir de la especificación abstracta. El sistema es más genérico y menos sofisticado, pero igualmente útil.

A continuación ofrecemos una instancia básica para resolver el problema de la esfera de n=3 variables usando un dcGA de 8 islas. En ella podemos observar que definimos valores incluso para algunas operaciones que disponen ya de valores por defecto. En el apartado de código de usuario se proporcionan declaraciones globales de variables y funciones auxiliares.

```
FRAME SPHERE IS xxGA (SOCKET)

              CONSTANT DataType             = Binary;
              CONSTANT GeneNumber           = 3;
              CONSTANT GeneLength           = 32;

              CONSTANT InitializationType   = InitBinRandom;
              CONSTANT IslandPopSize        = 64;
              CONSTANT BasicIslandEvolution = generational;

              CONSTANT Topology             = grid;
              CONSTANT Neighborhood         = NEWS;
              CONSTANT GridHeight           = 4;
              CONSTANT GridWidth            = 16;

              CONSTANT SelFirstParent       = SelectionTournament;
              CONSTANT SelSecondParent      = SelectionRandom;
              CONSTANT TournamentSize       = 2;

              CONSTANT CrossoverType        = DPX1;
              CONSTANT CrossoverProb        = 1.0;

              CONSTANT MutationType         = MutationConstant;
              CONSTANT MutationProb         = 0.010417;

              CONSTANT ReplacementType      = ReplIfBetter;

              CONSTANT NumberOfIslands      = 8;
              CONSTANT MigrationFrequency   = 1;
              CONSTANT SyncMode             = async;

              FUNCTION Evaluate = double Evaluate (xxGAContextPtr ctx, int p, int pop)
              {         int i;
                        double fitness=0.0, x;

                        for ( i=0; i<n; i++ )
                        {       x = xxGADecodeGene(ctx,p,pop,i);

                                fitness += x*x;
                        }
                        return fitness;
              };

              STATEMENT UserCode =
              {
                         int n = 3;                  /* Size of the problem */
              };

END SPHERE.
```

3.6.3. Nivel de Implementación

El nivel de implementación contendría los ficheros fuente de la implementación de *xx*GA y las reglas sobre cómo modificarlos para generar un fichero ejecutable que contemple los parámetros elegidos por el usuario usando la interfaz gráfica. Un proceso de pre-compilación daría lugar según el sistema de [RP97] a un directorio aparte con ficheros fuente y un fichero `makefile` listos para ser compilados en la máquina que se desee como destino (suponiendo en este caso que es posible usar la interfaz *socket* sobre ella). De hecho, dada las capacidades de configuración usando ficheros de texto de *xx*GA la recompilación real sería poco frecuente.

El proceso global consiste en generar los tres niveles para diseñar un marco, hacerlo disponible en un repositorio de Internet con su documentación asociada y dejarlo listo para usar. El usuario después podría generar los ficheros fuente para la ejecución a través de una interfaz gráfica amigable sin necesidad de conocimientos detallados sobre el tema ni los procesos internos de pre-compilación a los que se somete cada nivel del marco.

3.7. Resumen de la Propuesta

En resumen, las principales consideraciones hechas en este capítulo son las siguientes:

- Es necesario un enfoque de diseño y de estudio unificado en el campo de los algoritmos evolutivos paralelos. Los beneficios de este trabajo permitirán elegir el modelo más conveniente para un problema, así como trasvasar conocimientos entre modelos paralelos.
- Los algoritmos evolutivos paralelos distribuidos no son una alternativa a los celulares. Es posible distribuir algoritmos con población estructurada o única. Se espera potenciar las ventajas teóricas en ambos casos, así como un menor tiempo de ejecución debido al uso de un sistema *hardware* distribuido.
- La propuesta de un modelo distribuido de islas de población única y/o celulares añade a las ventajas de menor tiempo de ejecución la flexibilidad de elegir la presión selectiva deseada, además de las posibilidades de ejecución heterogénea y asíncrona.
- Un diseño en forma de clases de objetos es una manera muy natural y práctica de implementar *software* evolutivo paralelo. El paradigma de la programación orientada a objetos es beneficioso en sí para el sistema de manipulación genética, así como para la ejecución paralela y el seguimiento de la ejecución. Sin embargo, deben cuidarse los detalles referentes a la herencia, sobre todo en las estructuras de datos básicas. Asimismo, las decisiones sobre la forma de implementar la población de estructuras y el almacén de operadores influyen en la flexibilidad y velocidad de cómputo resultante.
- El diseño de un sistema de comunicación eficiente y de ficheros de configuración son puntos de especial interés en un sistema paralelo de esta envergadura en aras de una comunicación segura y con vistas a facilitar la extracción y uso de una familia concreta de algoritmos. La gran generalidad y versatilidad del sistema puede extenderse a un sistema de marcos de programación, permitiendo así un acercamiento de los algoritmos genéticos paralelos distribuidos a usuarios finales o investigadores de otras áreas.

4
Evaluación de los Resultados

Atendiendo a los planteamientos genéricos del Capítulo 1, debido a las necesidades y consideraciones del Capítulo 2, y utilizando los modelos algorítmicos concretos propuestos en el Capítulo 3, realizamos en este capítulo un estudio de dichos algoritmos y analizamos los resultados obtenidos. De esta forma, haremos hincapié en una comparativa que permita poner de manifiesto las principales ventajas e inconvenientes de cada técnica y modelo. En concreto, este capítulo se encuentra dividido en 5 secciones.

La primera sección (4.1) contiene un estudio del comportamiento canónico teórico de los numerosos modelos y variantes desde tres puntos de vista: (a) el mantenimiento de la diversidad, (b) la presión selectiva y (c) el tratamiento de esquemas. Todos estos puntos de vista aclaran de forma comparativa el tipo de búsqueda que realiza cada modelo, permitiendo decidir en el futuro cuál es más apropiado según el tipo de problemática al que se enfrente.

La Sección 4.2 discute por separado la importancia práctica de algunos de los parámetros más influyentes en los modelos de población descentralizada. Nuestro objetivo es profundizar en su conocimiento tanto como aportar datos que puedan ayudar a futuros estudios. En concreto, la Sección 4.2.1. presenta la influencia real de la política de migración en un modelo distribuido. La Sección 4.2.2 estudia la considerable importancia del tipo de reemplazo en un algoritmo celular. Finalmente, la Sección 4.2.3 discute en detalle la influencia del tipo de malla usada en un algoritmo genético celular en términos de su eficiencia y porcentaje de éxito.

Una vez realizado el estudio teórico sobre comportamientos básicos, y establecida la importancia de parámetros propios del uso de poblaciones estructuradas, discutimos en la Sección 4.3 el esfuerzo de evaluación de los modelos *xx*GA sobre instancias de problemas conocidos, con la intención de tomar un punto de vista práctico discutiendo algunos estudios preliminares y comparando con otros algoritmos. El número de evaluaciones (puntos visitados) es una interesante métrica para discernir el tipo de algoritmos con el que nos enfrentamos y resaltar sus características más sobresalientes con vistas a su uso o extensiones futuras.

La Sección 4.4 aborda en detalle la eficiencia de los algoritmos paralelos distribuidos en términos del esfuerzo de evaluación, ganancias de velocidad e influencia del sincronismo. En la Subsección 4.4.1 se estudian exhaustivamente dichas características sobre el problema SPH16-32, mientras que en la Subsección 4.4.2 se confirman y afinan algunas conclusiones sobre los modelos algorítmicos analizados. Para este fin se utilizan problemas considerablemente más difíciles como el entrenamiento de RNs y el problema de la suma del subconjunto.

En último lugar, realizamos un breve estudio sobre la forma en que los múltiples modelos resisten el crecimiento en la dificultad del problema que resuelven usando recursos computacionales constantes. La Sección 4.5 muestra un aspecto en la evaluación de algoritmos que puede orientar sobre la potencia de búsqueda de las distintas variantes de estos heurísticos.

4.1. Comportamiento Teórico Básico

En esta sección pretendemos comprobar el comportamiento teórico de los modelos más interesantes de variantes derivadas de *xx*GA. Para ello comenzaremos con un análisis comparativo del mantenimiento de la diversidad, tanto en modelos de población única como de población estructurada (y tanto secuenciales como paralelos). En la segunda subsección discutiremos sobre dos problemas la existencia empírica de los distintos tipos de presión selectiva derivados teóricamente en el Capítulo 2. Finalmente, la tercera subsección revisa el tratamiento de esquemas en las variantes algorítmicas mencionadas para comprobar de qué forma se ajusta cada algoritmo al comportamiento predicho por el teorema de los esquemas.

4.1.1. Mantenimiento de la Diversidad

A continuación iniciamos el estudio comparativo analizando uno de los aspectos más importantes de estos algoritmos: la forma en que cada uno mantiene la diversidad de su genotipo a lo largo de la evolución. Esta característica es importante debido a que está relacionada con el funcionamiento de los operadores, la amplitud de la búsqueda, la aparición de mínimos locales e incluso con la capacidad de solucionar problemas cuya función objetivo cambia dinámicamente -capacidad de adaptación- a lo largo de la evolución (problemas no-estacionarios) [KH97].

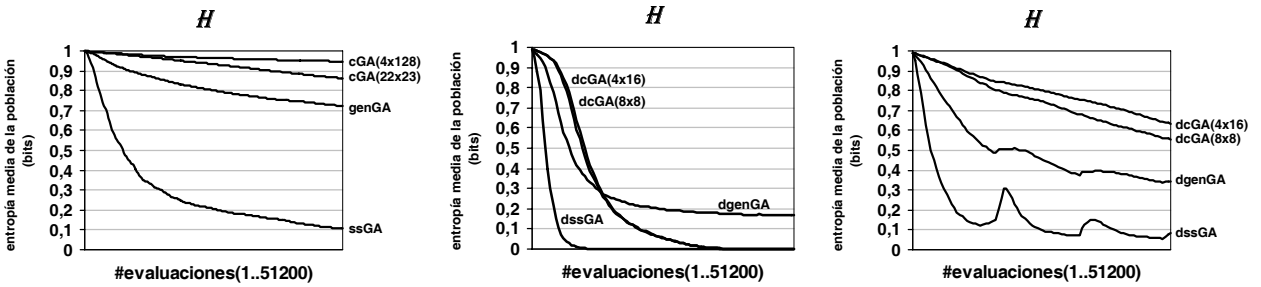
Existe un amplio abanico de posibilidades para medir la diversidad de una población y cómo ésta varía a lo largo de una ejecución. Una solución es medir la distancia de Hamming media de los individuos o incluso usando fórmulas a la medida [ER96]. Sin embargo, en este trabajo hemos optado por un concepto tradicional de teoría de la información como es la *entropía* (Ecuación 4.1). De esta forma, atendiendo a la formulación general de cantidad de información, medimos la entropía media (en bits) de la población $P(t)$ en la generación t como:

$$H\left[P(t)\right] = -\frac{1}{l} \cdot \sum_{i=1}^{l} \left(P_0^i \cdot \log_2 P_0^i + P_1^i \cdot \log_2 P_1^i \right) \ , \tag{4.1}$$

donde P_0^i es la proporción de ceros en la posición i-ésima y P_1^i la proporción de unos.

Es de esperar que la población inicial de cualquier modelo presente altos valores de entropía debido a que su composición sigue una distribución aleatoria uniforme. A lo largo de la evolución la selección tiende a disminuir la entropía mientras que el cruce normalmente no afecta a este valor. La mutación sí que tiene un efecto apreciable al tender a introducir mayor entropía de la existente para mantener una búsqueda útil.

Puede observarse en la Figura 4.1a (media de 20 ejecuciones) que los modelos generacionales y celulares mantienen la diversidad de forma mucho más acusada que el modelo estacionario (512 individuos). Este resultado es de esperar debido a la componente de explotación de este modelo. En el caso distribuido (8 islas de 64 individuos) las curvas cambian sustancialmente al disponer de islas de poblaciones más pequeñas donde se pierde diversidad a mayor velocidad. Básicamente, los tres modelos aceleran su pérdida de diversidad. En caso de una migración cada generación (Figura 4.1b) la pérdida es más rápida que en el caso de una migración infrecuente (Figura 4.1c) donde se generan distintas especies de solución que promocionan la diversidad cuando migran a otra población (montañas en los modelos estacionario y generacional).



(a) (b) (c)

Figura 4.1. *Entropía media de la población (en bits) en la resolución del problema SPH16-32 para los modelos no distribuidos* (a) *y distribuidos: migración cada* 64 *evaluaciones* (b) *y cada* 2048 *evaluaciones* (c). *Usamos* 512 *individuos, en el caso distribuido* 8 *islas con* 64 *individuos cada una, cruce de dos puntos con* p_c=1.0 *y mutación binaria con* p_m=1/*l*.

Nótese cómo dcGA se ve menos afectado debido a sus elevados índices de diversidad mantenida gracias al aislamiento por la distancia. De hecho, son las migraciones las que impiden en la Figura 4.1c un descenso tan acusado como el de la Figura 4.1b.

Sin embargo, la diversidad no es por sí sola un indicio de la calidad de la búsqueda, ya que lo ideal es perder diversidad muy rápido siempre que sea en beneficio de una solución al problema. La excepción a esta regla la encontramos cuando se desean encontrar múltiples soluciones al mismo tiempo en la población final, o bien cuando se busca la adaptación a entornos cambiantes. En la Figura 4.1 la convergencia a una solución ocurre para los modelos estacionario y celular, y en menor medida para el generacional, que no llega a afinar totalmente una solución a este problema en el rango de evaluaciones mostrado. De hecho, el modelo estacionario encuentra la solución muy rápido debido a la sencillez relativa del problema (SPH16-32).

En la Figura 4.2 podemos observar cómo esta situación cambia ante un problema difícil como el de la suma del subconjunto, para modelos no distribuidos (Figura 4.2 izqda.) y distribuidos (Figura 4.2 dcha.). Nótese en el gráfico de la izquierda cómo el rango del eje Y es [0,9..1], debido a que la entropía media es muy alta para todos los algoritmos, ya que el espacio de búsqueda es considerablemente extenso (y también la varianza de la adecuación).

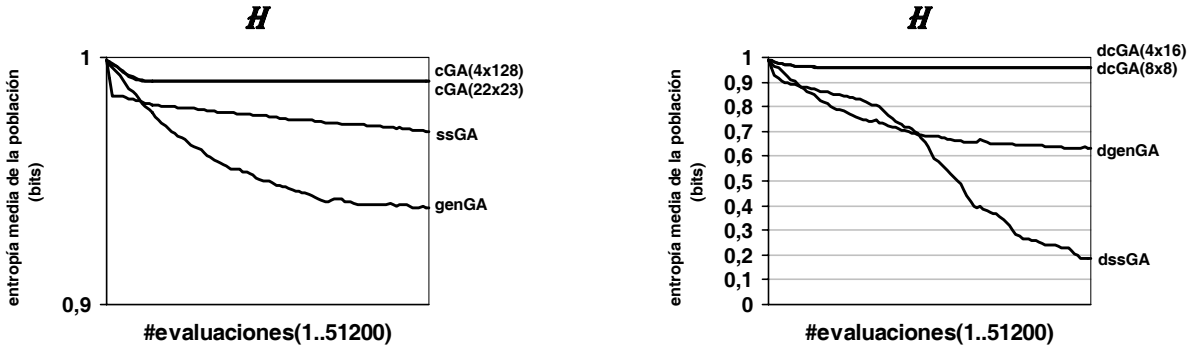


Figura 4.2. *Entropía media de la población (en bits) en la resolución del problema SSS128 para los modelos no distribuidos* (izqda.) *y distribuidos* (dcha.). *Usamos* 512 *individuos, en el caso distribuido* 8 *islas con* 64 *individuos cada una, cruce de dos puntos con* p_c=1.0 *y mutación binaria con* p_m=1/*l -migración de una cadena aleatoria cada* 2048 *evaluaciones-*.

Este escenario explica por qué los algoritmos estacionarios, incluso cuando están distribuidos, con frecuencia no presentan de forma acusada los problemas que teóricamente pueden derivarse sobre pérdida de diversidad, proporcionando además resultados competitivos [AAT93b] [Whit89] [ACT99]. En este problema, las versiones distribuidas de los modelos celulares (Figura 4.2 dcha.) continúan manteniendo una diversidad muy elevada (>0,9) durante toda la búsqueda, haciendo patente su componente de exploración. Ni siquiera las poblaciones de 64 individuos en el caso estacionario muestran tendencias preocupantes de pérdida de diversidad. Esta situación es ideal para estos modelos que de por sí agudizan la explotación del material de la población.

La diversidad de la población es de esperar que dependa de su tamaño. Para estudiar esta característica presentamos en Figura 4.3 el conjunto de resultados para resolver SSS128 como en el caso anterior pero usando una población global de $256/n_{proc}$ en vez de $512/ n_{proc}$.

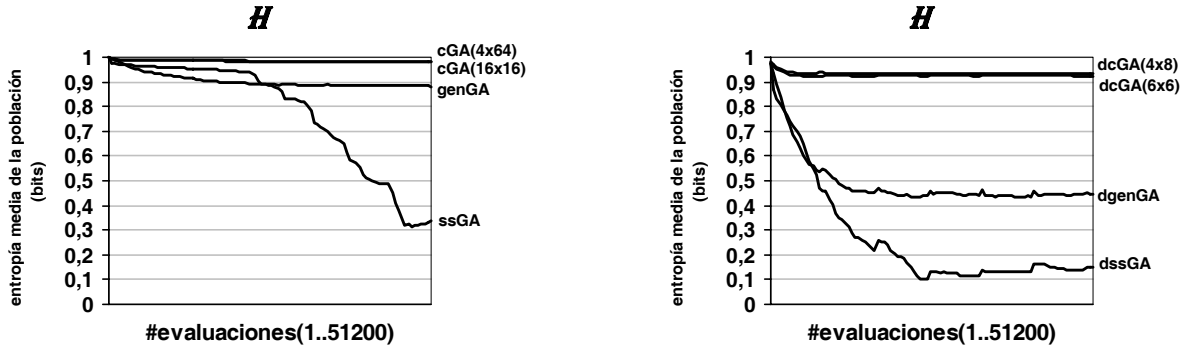


Figura 4.3. *Entropía media de la población (en bits) en la resolución del problema SSS128 para los modelos no distribuidos* (izquierda) *y distribuidos* (derecha)*. Usamos* 256 *individuos, en el caso distribuido* 8 *islas con* 32 *individuos cada una (migración cada* 2048 *evaluaciones).*

Puede observarse cómo el modelo generacional y celular no distribuidos (Figura 4.3 izqda.) mantienen una diversidad también superior al modelo estacionario. Éste, sin embargo, no presenta la típica curva de decrecimiento exponencial de la diversidad, sino que demuestra un descenso paulatino bastante deseable. El mismo efecto se mantiene en el caso distribuido con 8 procesadores (Figura 4.3 dcha.). Obsérvese de nuevo en ambos casos la elevada diversidad de los modelos celulares. En este y en el resto de casos anteriores el tipo de rejilla parece influenciar mínimamente la diversidad media de la población, tanto si están distribuidos como si no lo están. En el caso de SPH16-32 se debe a la alta velocidad con que se encuentra y propaga la solución, y en el caso de SSS128 se debe al elevado número de posibles valores de adecuación que fácilmente aparecen.

Otro parámetro que afecta al mantenimiento de la diversidad es el intervalo de migración. Ya comprobamos cómo un aislamiento pronunciado permite mantener gran cantidad de información en el algoritmo (Figura 4.1c). En el problema SS128 el efecto es similar, aunque no tan acusado, debido al elevado número de valores de adecuación distintos posibles. Compruebe cómo al reducir de 2048 (Figura 4.4a) a 64 (Figura 4.4b) el número de evaluaciones entre migraciones la diversidad tiende a disminuir en el modelo celular debido al mayor acoplamiento. El intervalo de migración parece influir más en el comportamiento celular distribuido que la reducción del tamaño de la población, lo cual nos recuerda la importancia de este parámetro de la migración. Es por esto que estudiaremos aparte en la Sección 4.2 dicha influencia.

Los modelos dgenGA y dssGA tienden a subir ligeramente respecto al uso de un intervalo de migración mayor debido a que el algoritmo no puede encontrar en la mayoría de los casos una solución con un intervalo tan pequeño (generando continuamente buenos y distintos individuos que mantienen la diversidad pero que no son solución).

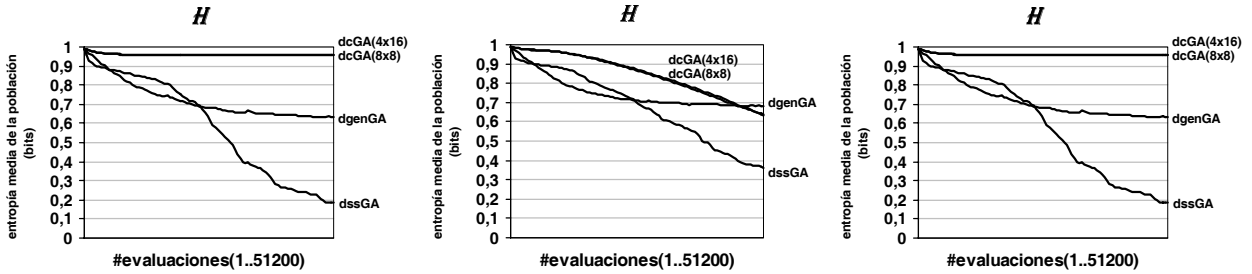


Figura 4.4. *Entropía media de la población (en bits) en la resolución del problema SSS128 para los modelos distribuidos con* (a) *intervalo de migración de* 2048 *evaluaciones* (b) *de 64 evaluaciones y* (c) *distribuidos asíncronos con migración cada* 2048 *evaluaciones. Usamos siempre* 512 *individuos en total, 8 islas, e iguales probabilidades que para la* Figura 4.2*.*

Ya que hemos comprobado la influencia del tamaño de la población y del intervalo de migración nos resta confirmar la hipótesis de que ejecutar el modelo de forma asíncrona no altera significativamente el espectro de entropía. Efectivamente, la Figura 4.4c confirma esta hipótesis. Podemos comprobar su similitud casi total con el modelo síncrono equivalente de la Figura 4.4a.

En resumen, hemos podido observar que, en la práctica, los modelos estudiados mantienen una diversidad excelente en el caso generacional y celular y sorprendentemente buena en el caso estacionario si tenemos en cuenta que, en teoría, debería converger mucho más rápido. Estos resultados permiten comprender mejor el comportamiento en la solución de los problemas abordados y hacen de los modelos presentados algoritmos ideales para combinarlos con operadores de búsqueda local, aumentando así las expectativas de éxito [Cott98].

El intervalo de migración usado y el tamaño de la población afectan a la diversidad sensiblemente, y el valor más adecuado para cada uno de ellos depende del problema. En general, los algoritmos distribuidos usan poblaciones globales mayores que los secuenciales y también intervalos de migración altos, lo que favorece la diversidad mucho, preparando al algoritmo para trabajar con problemas complejos. Hemos comprobado asimismo cómo el tipo de sincronización no afecta sensiblemente al nivel de entropía, aunque es de esperar que sí lo haga en cuanto a tiempo de ejecución (Sección 4.4).

Para terminar este estudio sobre la diversidad queremos presentar algunos detalles del funcionamiento del modelo celular en relación a este aspecto. En las gráficas anteriores hemos presentado la entropía para una malla cuadrada y para otra delgada por separado porque existen evidencias anteriores a nuestro trabajo de que dicho parámetro afecta al tipo de búsqueda del algoritmo. En particular, la Sección 4.2.3 profundizará en este tema. Por el momento, queremos comprobar cómo se difunden las estructuras en una población estructurada en malla. Primero discutiremos algunos detalles generales y después demostraremos empíricamente la forma en que dicha mayor diversidad está presente en las mallas delgadas.

El tiempo necesario para que una solución se extienda por toda la malla de μ individuos es $O(\sqrt{\mu})$ para una malla cuadrada y $O(\mu)$ para una malla rectangular de altura 4, 3, 2, 1 y valores altos de μ. En la práctica aparecen diferencias notables debido a que en su viaje por la malla una cadena debe competir con nuevas cadenas cada vez mejores, produciéndose una difusión más suave en el caso de una malla delgada. Esta es la razón de su lenta convergencia y mayor exploración. De hecho, sería necesario demostrar que mantener la diversidad implica mejorar la probabilidad de encontrar una solución, aunque, en general, esto se asume en la práctica.

En la Figura 4.5 mostramos la distribución de la adecuación a través de varias generaciones para distintos tipos de mallas que resuelven MMDP15 (selección proporcional, DPX1 con probabilidad 1.0, mutación con probabilidad 0.1 y reemplazo por torneo binario). Un color más oscuro significa un valor de adecuación mayor. Puede observarse cómo aparecen regiones (*nichos*) del mismo color. Es en los límites entre estas regiones donde se produce la exploración [SM91]. Existe una relación entre las regiones en la malla y las regiones del espacio de búsqueda explorado. Por tanto, a mayor número de regiones mejor exploración.

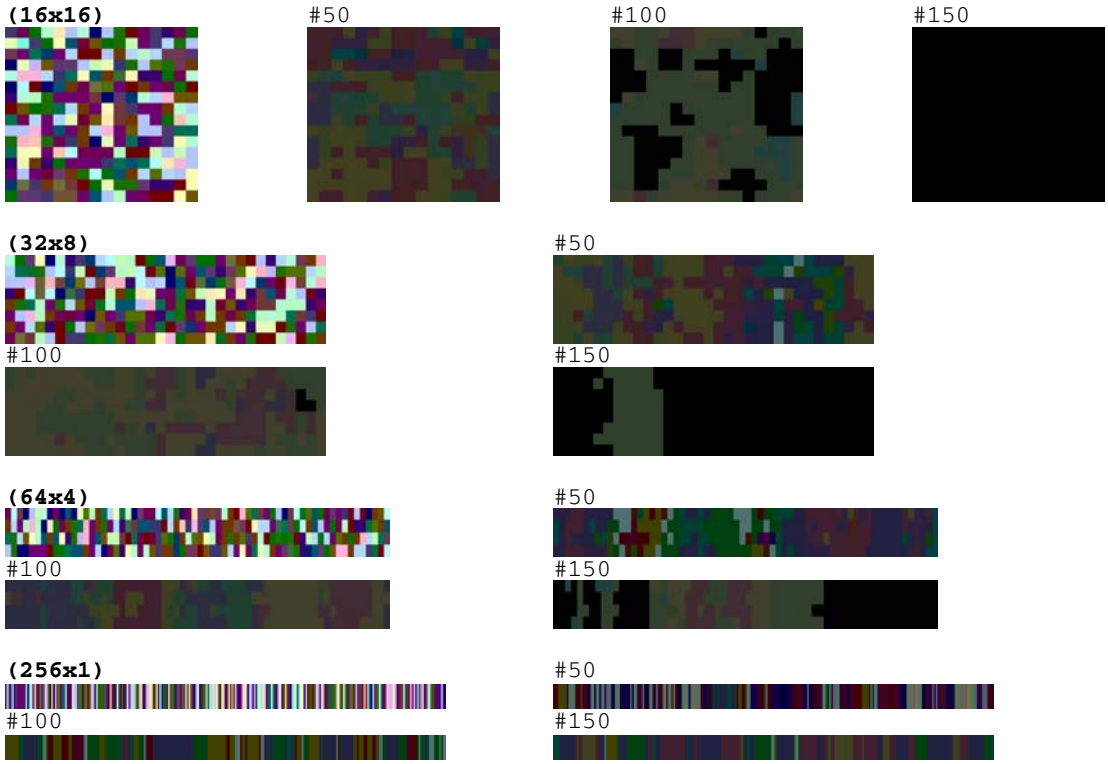


Figura 4.5. *Evolución (#paso) de la adecuación en un algoritmo genético celular de malla progresivamente más delgada, sobre el problema MMDP15.*

Una vez estudiado el mantenimiento de la diversidad en los modelos que nos interesan pasamos en la siguiente sección a comprobar empíricamente que la presión selectiva es distinta en cada uno de ellos, y que a pesar de las modificaciones de los operadores de variación, esta presión teórica puede comprobarse en problemas reales.

4.1.2. Presión Selectiva. Teoría y Práctica

En esta sección presentamos la presión selectiva en términos de la proporción de la mejor clase de estructuras presente en la población a lo largo de múltiples generaciones hasta obtener convergencia. Para ello, simulamos una población infinita usando 1024 individuos con valores de adecuación distribuidos uniformemente en el rango [0..255] y estudiamos el crecimiento de la mejor clase con selección proporcional a la adecuación (media de 50 ejecuciones usando RW).

La Figura 4.6 demuestra que, efectivamente, la intensidad de la selección es cualitativamente la esperada de la formulación matemática (Capítulo 2). En particular, la relación entre los modelos básicos de evolución resulta en una rápida convergencia del modelo estacionario, una lenta evolución del modelo generacional 1-elitista y una presión ajustable del modelo celular. Este último modelo es especialmente interesante por la facilidad con que podemos variar la presión para llevarla desde el extremo estacionario (explotación dominante) al generacional (exploración dominante) únicamente cambiando la forma de la malla (de 32x32 a 4x256).

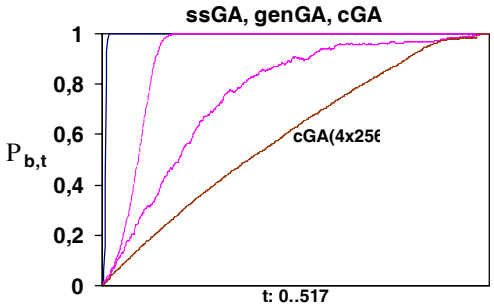


Figura 4.6. *Proporción de la mejor clase en el modelo secuencial estacionario y generacional, así como dos casos extremos de modelo celular: rejilla cuadrada* 32x32 *y rectangular* 4x256.

Efectivamente, la rejilla cuadrada proporciona un buen compromiso entre exploración/explotación, mientras que una rejilla muy estrecha (como 4x256) puede dar lugar incluso a presiones menores que en el caso generacional. La importante diferencia respecto a él es la disposición bidimensional del cGA que permite mayor diversidad, y el reemplazo elitista que evita degradaciones de la búsqueda por provocar un crecimiento monótono de los valores de adecuación. Todos ellos son aspectos positivos del algoritmo celular, con la ventaja añadida del fácil control de su presión selectiva. En todo caso, en lo que a presión selectiva se refiere, puede conseguirse un algoritmo celular de igual presión que otro secuencial cualquiera [SD96].

Por otro lado, el modelo genérico *xx*GA permite también especificar modelos distribuidos. Proporcionamos en la Figura 4.7 la presión selectiva de las versiones distribuidas síncronas de los modelos discutidos arriba usando 8 islas de 128 individuos cada una. En estos resultados distinguimos en primer lugar cómo cualquiera de las versiones distribuidas tiene mayor presión que su respectivo no distribuida.

Cuando las subpoblaciones están completamente aisladas (frecuencia de migración nula) la convergencia es más rápida que si existiese una única población (panmixia). En el caso del modelo estacionario apenas si existen diferencias en relación a las distintas frecuencias de migración. Mostramos los resultados cuando la migración ocurre tras un número de evaluaciones igual a 0, 1 o 32 veces el tamaño total de la población.

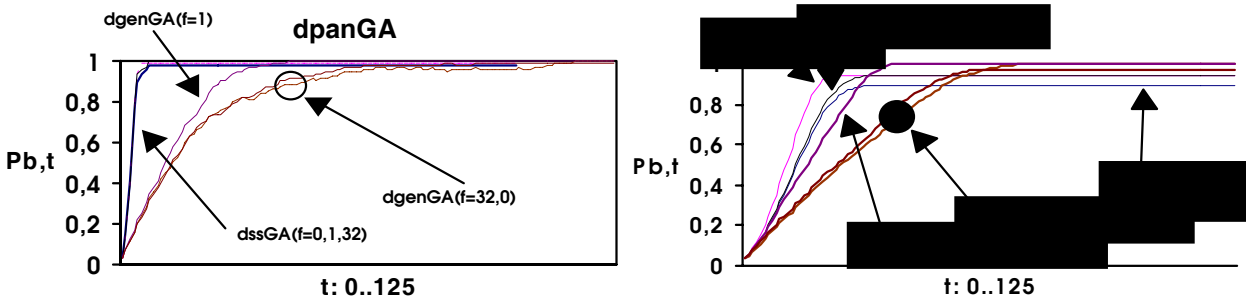


Figura 4.7. *Presión de la selección en los modelos distribuidos con evolución básica presentada en la* Figura 4.6. *Usamos* 8 *islas de* 128 *individuos. En el caso celular, los* 128 *individuos se disponen bien en rejilla cuadrada* (11x11) *o rectangular* (4x32).

En el caso generacional, la presión de su versión distribuida explica los buenos resultados de la mayoría de aplicaciones que comparan modelos generacionales con sus contrapartidas distribuidas. Puede observarse cómo la conexión en cada generación produce una mayor presión, mientras que el modelo con migración cada 32 generaciones locales se comporta de forma muy parecida al modelo aislado.

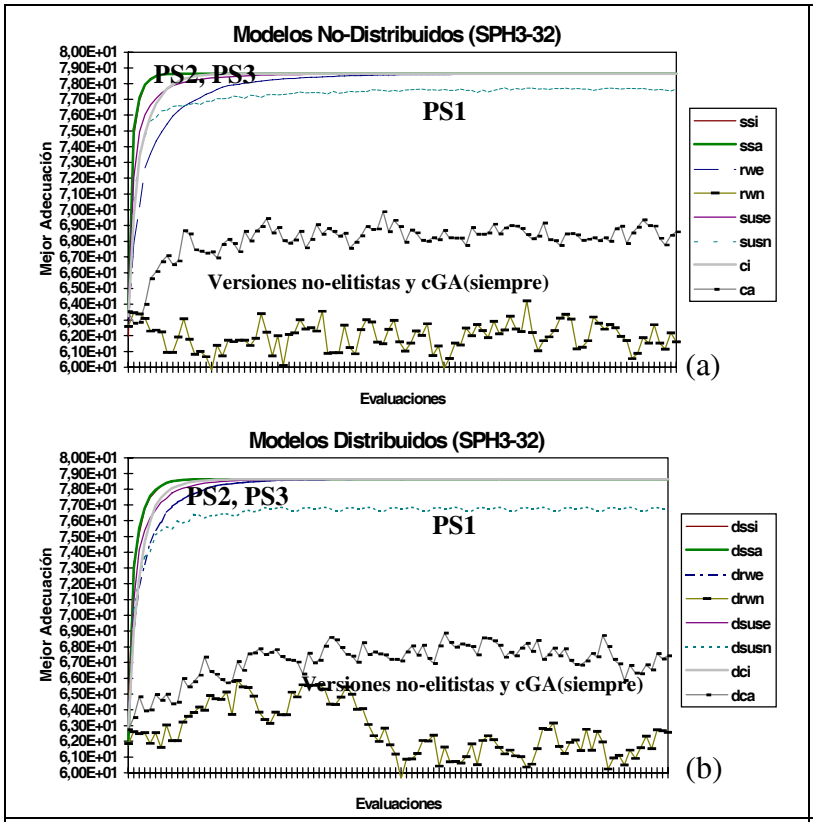
Finalmente, el modelo distribuido con islas celulares proporciona varias conclusiones interesantes. En primer lugar, concluimos el hecho de que usar islas cuadradas en dcGA puede provocar incluso convergencia prematura en algunos casos, sobre todo cuando la conexión es mínima (32) o inexistente (0). Este efecto es menor si la rejilla es rectangular. Una ventaja adicional de usar una rejilla rectangular es que es posible aumentar la presión a pesar de usar poblaciones relativamente pequeñas, ya que la diversidad se agota más lentamente.

Las curvas presentadas diferencian formas de comportamiento en la práctica para los distintos algoritmos. En el caso de una población no estructurada podemos variar la presión selectiva usando una versión distribuida con mayor/menor frecuencia de migración. En el caso celular tenemos una fuente *adicional* para modificar esta presión: la forma de la malla.

Para concretar aún más y entender cómo evolucionan los modelos estudiados es necesario seguir su evolución. Además, de esta forma tendremos la oportunidad de comprobar si en aplicaciones reales, bajo la acción combinada de todos los operadores, es posible distinguir los distintos tipos de presión selectiva. Ofrecemos en la Figura 4.8 los resultados de la evolución del mejor individuo a lo largo de los diferentes pasos de los modelos (en términos de cuántas evaluaciones se han realizado) para los problemas SPH3-8 y RAS20-8.

Podemos observar en dicha figura cómo los resultados sobre ambas funciones muestran las tendencias ya encontradas en los resultados numéricos preliminares. Intuitivamente es fácil descubrir las ventajas de los modelos con presión selectiva PS2 y PS3 y confirmar las hipótesis sobre la lentitud de los modelos generacionales, en igualdad de condiciones, respecto a los estacionarios y celulares (mayor aclaración sobre los símbolos en el Apéndice C y Tabla 3.1).

Ya que RAS20-8 es considerablemente más difícil para los modelos estudiados que SPH3-8 podemos apreciar cómo el primero distingue mejor que el segundo las clases relativas a las presiones selectivas mencionadas. Adicionalmente, el efecto de la distribución (8 islas) es siempre beneficioso y además permite acelerar la curva de crecimiento del mejor individuo.



ALGORITMOS COMPARADOS

No Distribuidos:

*xx*GA(1,128,_,_,ss,alw)
*xx*GA(1,128,_,_,ss,ifb)
*xx*GA(1,128,_,_,rw,e)
*xx*GA(1,128,_,_,rw,ne)
*xx*GA(1,128,_,_,sus,e)
*xx*GA(1,128,_,_,sus,ne)
*xx*GA(1,2x64,_,_,cel,alw)
*xx*GA(1,2x64,_,_,cel,ifb)

Paralelos Distribuidos:

*xx*GA(8,16,r,4800,ss,alw)
*xx*GA(8,16,r,4800,ss,ifb)
*xx*GA(8,16,r,32,rw,e)
*xx*GA(8,16,r,32,rw,ne)
*xx*GA(8,16,r,32,sus,e)
*xx*GA(8,16,r,32,sus,ne)
*xx*GA(8,2x8,r,32,cel,alw)
*xx*GA(8,2x8,r,32,cel,ifb)

ALGORITMOS COMPARADOS

No Distribuidos:

*xx*GA(1,180,_,_,ss,alw)
*xx*GA(1,180,_,_,ss,ifb)
*xx*GA(1,180,_,_,rw,e)
*xx*GA(1,180,_,_,rw,ne)
*xx*GA(1,180,_,_,sus,e)
*xx*GA(1,180,_,_,sus,ne)
*xx*GA(1,2x90,_,_,cel,alw)
*xx*GA(1,2x90,_,_,cel,ifb)

Paralelos Distribuidos:

*xx*GA(8,22,r,720,ss,alw)
*xx*GA(8,22,r,720,ss,ifb)
*xx*GA(8,22,r,32,rw,e)
*xx*GA(8,22,r,32,rw,ne)
*xx*GA(8,22,r,32,sus,e)
*xx*GA(8,22,r,32,sus,ne)
*xx*GA(8,2x11,r,32,cel,alw)
*xx*GA(8,2x11,r,32,cel,ifb)

Figura 4.8. *Resultados sobre SPH3-32/RAS20-8* (96/160 *bits,* 128/180 *individuos,* p_c=1.0/1.0, p_m=0.01/0.01) *para las versiones no distribuidas* (a)/(c) *y distribuidas* (b)/(d).

Para terminar, hemos comprobado la importancia de usar un reemplazo elitista: las cuatro gráficas anteriores sitúan mal las versiones no elitistas y a cGA con reemplazo siempre. También hemos confirmado las diferencias de asignación de instancias existentes en teoría para las técnicas de selección RW y SUS, esta última mejor en casi todos los aspectos [Bake87].

4.1.3. Tratamiento de Esquemas

En esta sección estudiamos el tipo de procesamiento de esquemas que cada modelo presenta para un problema. En primer lugar estamos interesados en comprobar si las curvas de presión selectiva se mantienen cuando se estudian esquemas. En segundo lugar pretendemos investigar la tasa de asignación de instancias (exponencial o no) de cada algoritmo.

El problema que abordamos consiste en encontrar una de entre dos cadenas solución de 36 bits. La evaluación de una cadena (buscamos un máximo) resta de 72 la suma de distancias de Hamming de la cadena evaluada a ambas cadenas solución (resta 0 si se trata de una de las dos soluciones). El óptimo es por tanto 72 y el problema es bi-modal. En la Tabla 4.1 proporcionamos los detalles sobre las cadenas objetivo y el esquema que estudiamos. Ambas cadenas pertenecen al esquema estudiado, y existen otras 14 cadenas que pertenecen al esquema aunque no son soluciones al problema.

TABLA 4.1. PARÁMETROS, ALGORITMOS Y EL ESQUEMA ESTUDIADO

PARÁMETROS	VALOR	ALGORITMOS	CADENAS Y ESQUEMA SOLUCIÓN
Número de vbles.	3	*xx*GA(1,80,_,_,ss,ifb)	SOLUCIÓN1*:*
Long. de la cadena	36	*xx*GA(1,80,_,_,sus,e)	110000000000 110000000000 000000000011
Tamaño de la pobl.	80	*xx*GA(1,2x40,_,_,cel,ifb)	SOLUCIÓN2*:*
p_c	1.0	*xx*GA(4,20,r,640,ss,ifb)	110000000000 000000000011 000000000011
p_m	0.01	*xx*GA(4,20,r,32sus,e)	ESQUEMA: [*orden*=68*, long. de defin.*=71]
Número máx. evals.	10400	*xx*GA(4,2x10,r,32,cel,ifb)	110000000000 **00000000** 000000000011

En la Figura 4.9 izqda. podemos observar las diferencias entre genGA/dgenGA y el resto de los algoritmos. Volvemos a constatar que ambas versiones son muy lentas en comparación y no asignan instancias de forma exponencial en este problema. El modelo ssGA es el más rápido en explotar la población. Su versión distribuida en cuatro islas (dssGA) suaviza la curva y mejora la exploración, sin llegar a los extremos de las versiones generacionales.

Podemos confirmar también que las curvas de presión selectiva se mantienen en presencia de los operadores de cruce y mutación. Es posible distinguir los modos de presión selectiva (PS1, PS2 y PS3) descritos en teoría en la Sección 2.7.3.

Confirmamos también que en este estudio de esquemas la presión ajustable de cGA está presente, aunque debido a que la población es pequeña no se aprecian notables diferencias respecto a la forma de la malla. Esta es también la razón de la similitud entre cGA y dcGA. Se necesitan poblaciones mayores para corroborar las ventajas del modelo distribuido.

El crecimiento del número de instancias del esquema solución es muy lento en el caso generacional. El esquema estudiado provoca problemas en este algoritmo debido a su larga longitud de definición y alto orden, como era de esperar atendiendo al teorema de los esquemas. Sin embargo, en los modelos estacionarios y celulares sí que observamos el crecimiento exponencial esperado incluso para este esquema tan adverso al tratamiento genético tradicional.

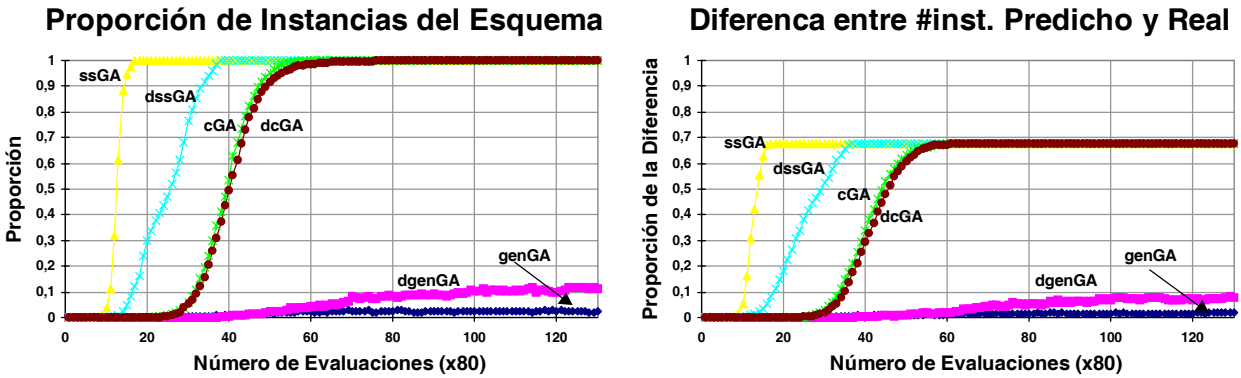


Figura 4.9. *Proporción de instancias del esquema* (izqda.) *y diferencia entre los valores predichos por el teorema de los esquemas y los valores reales* (dcha.) (*media de* 20 *ejecuciones*).

Por un lado, esto refuerza las ventajas de ambos modelos frente a los generacionales. Por otro lado, se pone de manifiesto al estudiar la diferencia entre instancias predichas y reales (Figura 4.9 dcha.) que el teorema es demasiado conservador y que son necesarias nuevas predicciones más ajustadas a la realidad y a cada modelo [Whit93a].

Como hemos observado, los resultados dependen en la mayoría de los casos de la política de migración utilizada. La siguiente sección ahonda en este aspecto estudiando cómo dicha política afecta a cada modelo para varios problemas.

4.2. Estudio Detallado de Parámetros Influyentes en los Modelos Descentralizados

De entre el conjunto de parámetros que caracterizan a un modelo distribuido la política de migración es quizás uno de los más importantes porque define el tipo de conectividad y búsqueda distribuida en general. La primera subsección profundiza en la evaluación de diferentes políticas de migración con la intención de aclarar sus ventajas/inconvenientes respectivos. Respecto al modelo celular mostraremos la importancia relativa de dos parámetros por separado. El primero es la política de reemplazo, encargada de discernir en cada generación quién sobrevive en la próxima generación. El segundo aspecto, muy importante, es la manera en la que la presión selectiva y, en general, la búsqueda, puede modificarse a través de la manipulación de la forma de la malla en un algoritmo genético celular.

4.2.1. Política de Migración en el Modelo Distribuido

Los algoritmos genéticos distribuidos incluyen parámetros de funcionamiento propios además de los tradicionales en AGs secuenciales. Entre los parámetros más importantes que guían la búsqueda de un AGP distribuido se encuentra la política de migración, y sobre todo la frecuencia de interconexión entre las islas.

En relación a esta frecuencia de migración, tradicionalmente se ha encontrado beneficioso utilizar islas “ligeramente acopladas”, es decir, que es en general mejor dejar a las islas un tiempo de aislamiento “considerable” ejecutando su modelo básico [Tane89] [Sten93] [Beld95].

Ya que las decisiones sobre este parámetro son inevitables en cualquier modelo distribuido en general, encontramos que en la literatura a menudo se mezcla un estudio sobre frecuencias de migración con otro sobre topología, operadores, etc. Nuestra intención es realizar un estudio canónico para comprender mejor lo que significa "débilmente acoplado".

En la Figura 4.10 presentamos los resultados de solucionar MMDP15 y ROS20. Estudiamos en dicho conjunto de pruebas tres parámetros por separado y comparamos su efecto sobre la búsqueda en cuanto al esfuerzo de evaluación (4.10a y 4.10b) y al porcentaje de éxitos (4.10c y 4.10d) (número de veces que se encuentra un óptimo entre las 50 pruebas independientes para cada conjunto de parámetros). Los parámetros estudiados incluyen la frecuencia de migración ζ, la técnica de selección del individuo migrado ω_S (mejor o aleatorio), y, finalmente, el tipo de evolución básica de cada isla (estado-estacionario -dssGA- frente a celular -dcGA-).

Los resultados sobre estos problemas arrojan tres conclusiones: (1) el esfuerzo de evaluación de dcGA es inferior al necesario con dssGA, (2) es mejor trabajar con una frecuencia de entre 16 y 32 generaciones y (3) es mejor enviar una copia de un individuo aleatorio que una copia del mejor individuo entre islas vecinas.

La superioridad de dcGA sobre dssGA en términos de esfuerzo de evaluación será incluso más frecuentemente encontrada en los resultados que siguen a esta sección, aunque no es cierta para cualquier problema debido a que su alta exploración tiende a ralentizar el alcance del óptimo en problemas con parámetros continuos. La calidad de la búsqueda en un sistema con disposición bidimensional de la población es en consecuencia mejor en los casos estudiados que en panmixia en cuanto al mantenimiento de la diversidad y exploración del espacio.

Los valores de 16 o 32 como los mejores intervalos de migración serán también refrendados por los resultados posteriores. Esto se debe a que suponen un buen compromiso entre acoplamiento "fuerte" y "débil" entre las islas.

Debemos recordar que se trata de múltiplos de la población, distinta en cada problema, y que por tanto estos valores representan valores distintos en problemas que usen poblaciones de distinto tamaño. Esto contrasta con la migración cada cierto número de generaciones, independientemente de la población o problema estudiado, que podemos encontrar como conclusión en otros trabajos.

Por último, seleccionar aleatoriamente es un buen criterio, ya que evitamos el "efecto conquista" en la población destino, al mismo tiempo que pasamos a la isla destino material útil elaborado en el algoritmo fuente. Además, el hecho de usar los valores de aislamiento mencionados permite integrar o rechazar rápidamente a la cadena entrante.

En general, la selección de la cadena migrada está relacionada también con la frecuencia de migración, y para la resolución de un problema concreto puede resultar más productivo migrar la mejor cadena de una población. Sin embargo, dado que nuestro estudio abarca un elevado número de modelos, parámetros y problemas, la migración de una cadena aleatoria es la que mejor se ha comportado en la mayoría de los casos. Excepto que se diga explícitamente, en los posteriores resultados migraremos una cadena aleatoria y reemplazaremos en la población destino a la peor cadena existente siempre que la cadena entrante sea mejor que ella.

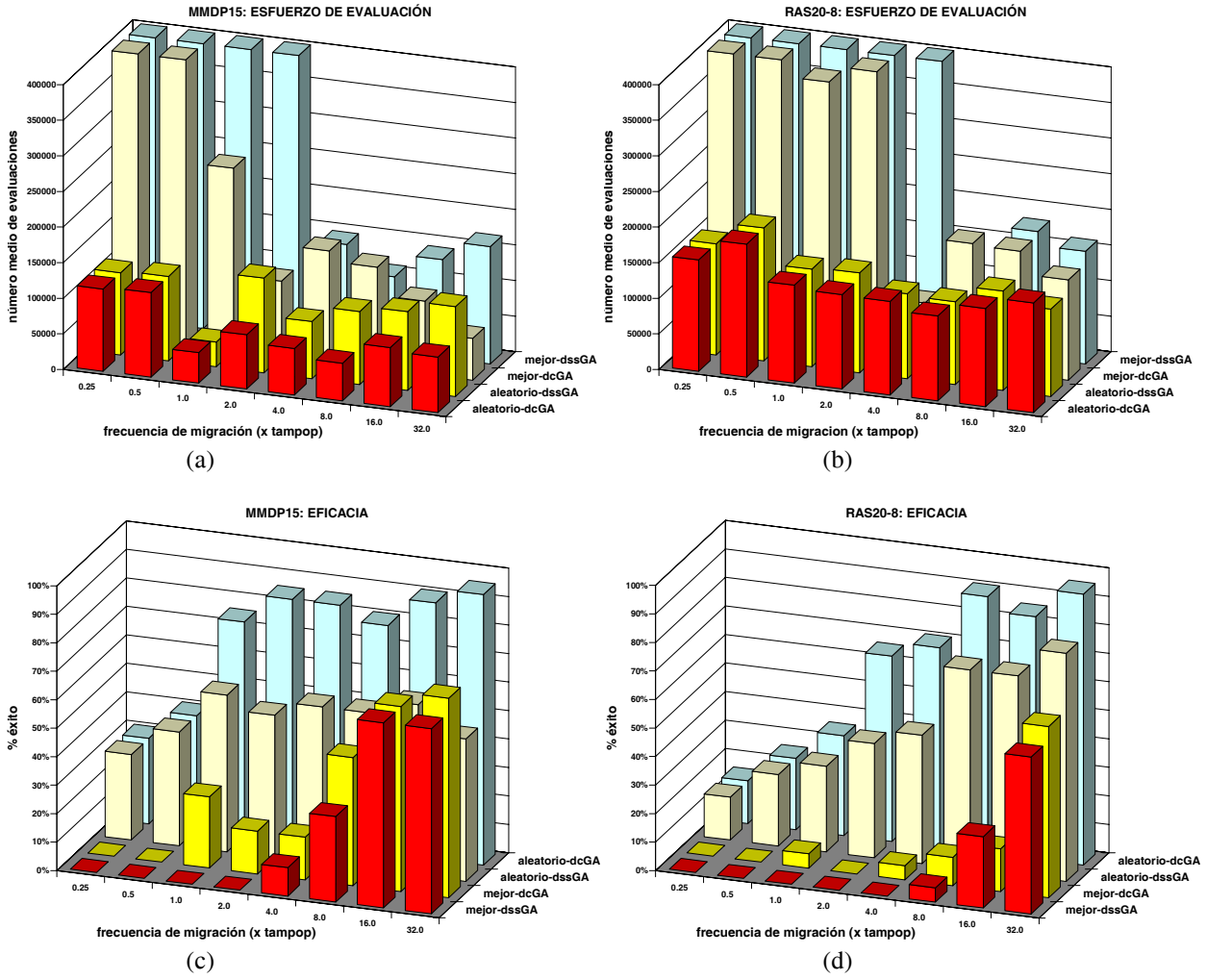


Figura 4.10. *Esfuerzo de evaluación para resolver MMDP15* (a) *y RAS20-8* (b) *y porcentaje de éxito sobre MMDP15* (c) *y RAS20-8* (d) *usando dssGA y dcGA. Mostramos la influencia de la política de migración (frecuencia y técnica de selección).*

4.2.2. Importancia de la Política de Reemplazo en el Modelo Celular

La selección (de parejas o para reemplazo) es un proceso de importancia en el modelo celular. En particular, el reemplazo utilizado determina el crecimiento/decrecimiento de la adecuación en cada punto de la rejilla y en cada paso del algoritmo, de ahí su influencia.

Una política simple consiste en reemplazar cada antigua cadena con la nueva generada en cada uno de los μ vecindarios (gránulos). A esta política no-elitista la denominaremos *always* ("siempre") y es una alternativa al reemplazo elitista consistente en sustituir la cadena antigua sólo si la nueva tiene mejor adecuación (*if_better* o "si mejor"). Esta última política es en realidad un *torneo binario* aplicado al reemplazo.

En principio, el reemplazo "siempre" facilita la exploración aunque no es elitista, mientras que el reemplazo "si mejor" es elitista y no permite decrecer la adecuación media de la población (asumiendo un problema de maximización). Véase el punto concreto de uso de estas políticas en el pseudocódigo del Capítulo 2 (Sección 2.6, Figura 2.15) en la operación `Insertar_Nuevo_Ind`. En la Tabla 4.2 presentamos el conjunto de parámetros y problemas usados para estudiar la influencia del reemplazo.

TABLA 4.2. CONJUNTO DE PARÁMETROS PARA RESOLVER LOS PROBLEMAS SPH3-32 Y RAS20-8

SPH3-32		**RAS20-8**	
Longitud cadena	3x32=96 bits	*Longitud cadena*	8x20=160 bits
Tamaño población	2x64=128	*Tamaño población*	2x90=180
p_c	1.0	p_c	1.0
p_m	0.01	p_m	0.01
max. #evaluaciones	38400	*max. #evaluaciones*	540000

Los resultados (Figura 4.11) son especialmente consistentes en todos los casos. El reemplazo "siempre" provoca los peores resultados: no encuentra ningún óptimo y no permite la mejora de la adecuación media de forma creciente.

Era de esperar que una política elitista como "si mejor" provocase mejores resultados debido a que se aplica a *cada cadena* de la población, y no a un porcentaje bajo, como es el caso de otros modelos. Por esta razón siempre utilizamos AGs celulares con reemplazo "si mejor" en nuestros posteriores análisis.

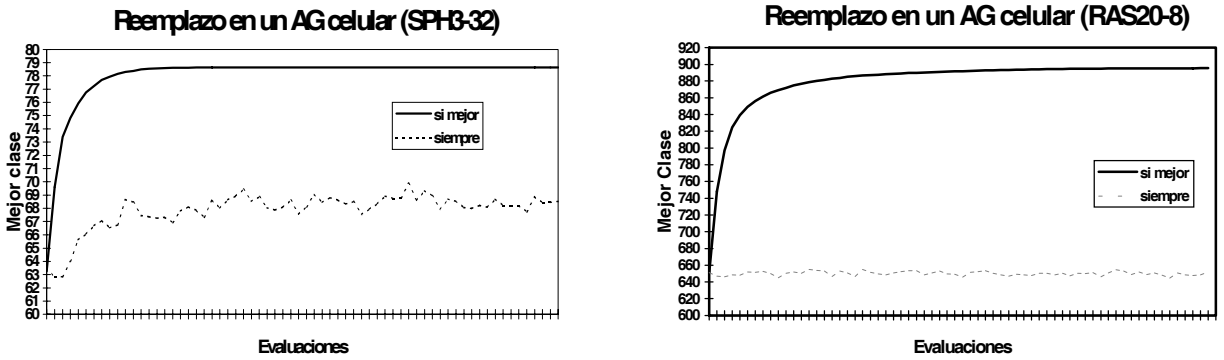


Figura 4.11. *Influencia de la política de reemplazo en un algoritmo genético celular para los problemas SPH3-32* (izquierda) *y RAS20-8* (derecha).

4.2.3. Importancia de la Forma de la Malla en el Modelo Celular

En esta sección discutimos la importancia del vecindario y la topología en un AG celular. Existen algunos trabajos en los que se sugiere la mayor eficiencia de mallas no perfectamente cuadradas (mallas delgadas). Véase por ejemplo los resultados en [MSB91] para grandes instancias de TSP, o en [Balu93] [GW93] para optimización funcional. En ellos se menciona, aunque no se estudia, el hecho de que la forma de la rejilla influencia la calidad de la búsqueda. Nuestro interés radica en caracterizar estos algoritmos como un tipo de AG con población espacialmente distribuida en forma de rejilla, sin las ventajas de un menor tiempo de ejecución provenientes de usar una máquina SIMD, pero con el mismo comportamiento algorítmico.

Existen otros factores que influencian la ejecución de un AG celular. Tal es el caso de los operadores de variación usados, sus probabilidades, etc. Pero esto no los distingue especialmente del resto de AGs. Es su selección descentralizada la principal diferencia respecto al resto, y ésta está muy influenciada por la topología y el vecindario. De hecho, ya hemos demostrado la dependencia de la presión selectiva respecto de la forma de la rejilla (Sección 4.1.2).

Nuestro objetivo es estudiar en todos los casos la influencia del ratio del AG celular en los resultados. Atendiendo a las definiciones de ratio del Capítulo 2 (Sección 2.4.2) y a las hipótesis de trabajo consecuencia de la presión selectiva, estructuramos el desarrollo en dos subsecciones. En primer lugar abordamos (Sección 4.2.3.1) un estudio comparativo del **esfuerzo de evaluación** necesario en rejillas de diferentes ratii para resolver el mismo problema. Después, la Sección 4.2.3.2 analiza la **respuesta a la selección** de algoritmos con diferentes ratii.

En todos los resultados se presenta la media de 50 ejecuciones independientes y se utilizan poblaciones lo más pequeñas posible para que los resultados sean comparables con los obtenidos con otros algoritmos. En algunas disposiciones de la malla no es posible usar todos los individuos. Por ejemplo, si la población consta de 100 individuos una rejilla de 3x33 sólo utiliza 99, aunque el efecto de este hecho es prácticamente nulo (lo mencionamos por completitud).

4.2.3.1. Eficiencia Numérica

Tras el estudio de un parámetro concreto como el tipo de reemplazo iniciamos con esta sección el estudio global sobre el comportamiento de mallas de distintos ratii en el mismo algoritmo, extendiendo la presentación a un grupo de problemas. En esta sección analizamos el esfuerzo de evaluación y el porcentaje de éxito obtenido al resolver instancias complejas de varias familias de problemas. En la Tabla 4.3 presentamos los parámetros usados.

TABLA 4.3. PARÁMETROS DE LOS ALGORITMOS

PARÁMETROS	**RAS10 (16 bits)**	**RAS20 (8 bits)**	**ROS10 (16 bits)**	**ROS20 (8 bits)**	**SPH3 (32 bits)**	**SPH6 (32 bits)**	**UMMDP6 (6 bits)**	**MMDP16 (6 bits)**
Tamaño problema	10	20	10	20	3	6	6	16
Longitud cadena	160	160	160	160	96	192	36	96
p_c	1.0	1.0	0.8	0.8	1.0	1.0	1.0	1.0
p_m	0.01	0.01	0.03	0.03	0.01	0.01	0.05	0.05

En la Figura 4.12 puede apreciarse el esfuerzo computacional en resolver cada problema. El número de evaluaciones para resolver el mismo problema es casi el mismo para cualquier forma de la rejilla (ratio) excepto para las rejillas 2x50 y 1x100. Esto se debe a que el vecindario pierde su significado en ellas y a una excesiva lentitud en la difusión de las estructuras en la población.

En general, se observa una tendencia a necesitar un mayor número de evaluaciones de las mallas más delgadas. Esto es lógico debido a su mayor incidencia en la exploración y el mantenimiento de diversidad, que son por otro lado ventajas en otro tipo de problemas de búsqueda. De hecho, comprobaremos que resulta casi la única desventaja de las mallas delgadas y además sólo en mallas excesivamente delgadas. El uso de rejillas no cuadradas y no completamente delgadas aparece como más atractivo por representar un compromiso entre exploración y explotación. La desventaja de un mayor esfuerzo es la que pretendemos solventar al ejecutar estos algoritmos de manera distribuida (dcGA).

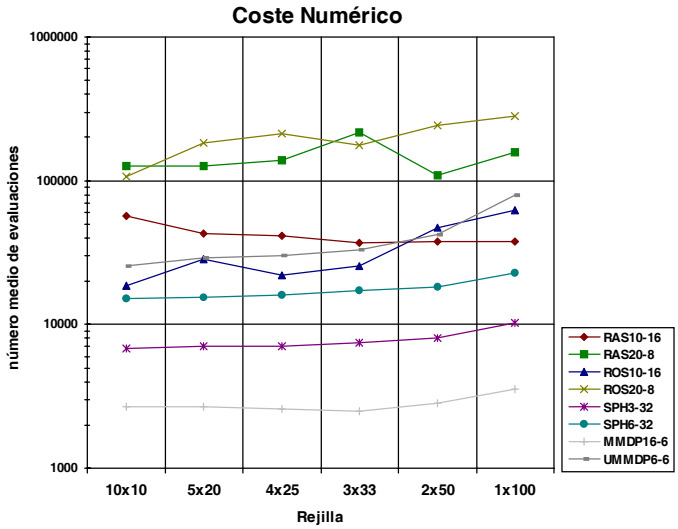


Figura 4.12. *Eficiencia: número medio de evaluaciones para resolver los problemas (50 ejcs.).*

En la Figura 4.13 podemos apreciar otro aspecto del tipo de búsqueda de un AG celular con distintos ratii: el porcentaje de éxito. En los problemas más simples este algoritmo encuentra el 100% de las veces el óptimo. Es sólo en las instancias con 10 y 20 variables de Rosenbrock y Rastrigin donde se aprecia que las rejillas con menor ratii (baja presión selectiva) producen notables incrementos en el porcentaje de éxito, compensando de esta forma el mayor número de evaluaciones que necesitan.

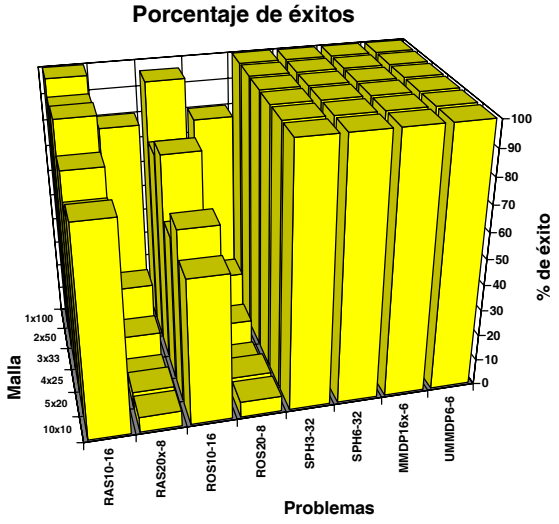


Figura 4.13. *Porcentaje de éxito resolviendo los problemas con distintas formas de la malla.*

En la Figura 4.14 presentamos dos conjuntos de resultados más detallados atendiendo a la solución de los problemas MMDP15 y RAS20-8 que lleva a cabo el AG celular usando distintas formas de la malla. A medida que la cuadrícula es más estrecha se obtienen mejores resultados. Podemos observar tanto dicho efecto como el decrecimiento del ratio del algoritmo.

Esto es lógico si pensamos que las mejores cadenas no hacen desaparecer a otras peores demasiado rápido, debido a su lenta propagación a través del eje más largo. Las peores cadenas se estancan en su vecindario próximo hasta desaparecer.

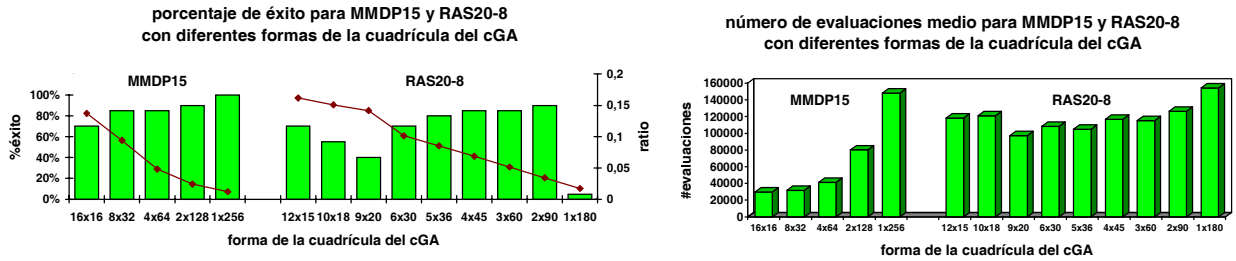


Figura 4.14. *Porcentaje de éxito* (izqda.) *y esfuerzo computacional* (dcha.) *para resolver MMDP15 y RAS20-8 con diferentes cuadrículas de un AG celular.*

Por tanto, concluimos que los valores de altura deseables son pequeños (*ratii* en [0.02 ... 0.05] o menores en grandes poblaciones). Las topologías degeneradas (altura 1, 2) funcionan de forma dependiente de la aplicación, bien para MMDP15 y mal para RAS20-8.

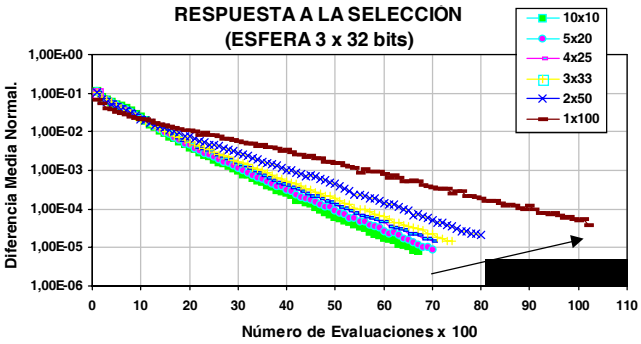
4.2.3.2. Respuesta a la Selección

La respuesta a la selección es un campo de trabajo de creciente interés ya que permite estudiar la velocidad de convergencia y otros aspectos interesantes de los algoritmos evolutivos [MS93]. El concepto clave es la llamada *respuesta a la selección*, definida como la diferencia entre la adecuación media de dos poblaciones sucesivas: $R(t+1)=S(t+1)-S(t)$.

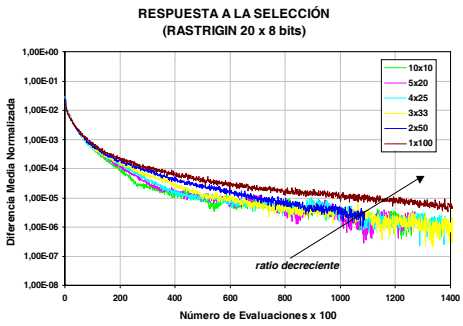
Resulta bastante complejo obtener resultados matemáticos cerrados y útiles para funciones arbitrarias en estos términos, y por el momento sólo es posible estudiar completamente funciones tan simples como la de "contar unos" [Thie97]. Sin embargo, las ramificaciones prácticas de esta teoría son fáciles de usar y arrojan resultados interesantes. En nuestro caso pretendemos estudiar la influencia de la forma de la malla y el tipo de problema en la evolución de la búsqueda.

Ya hemos mencionado la ventaja de mayor diversidad y exploración del AG celular y en especial de las mallas estrechas. Una reducción del ratio provoca una menor presión selectiva, mejor porcentaje de éxito (deseable), y mayor coste computacional (indeseable). Ya henos comprobado que el mayor número de evaluaciones del AG celular con malla delgada va acompañado efectivamente de una mayor diversidad. Pretendemos ahora estudiar si ocurre lo mismo para la mejora en la media de la adecuación mantenida durante la búsqueda (mejor respuesta a la selección).

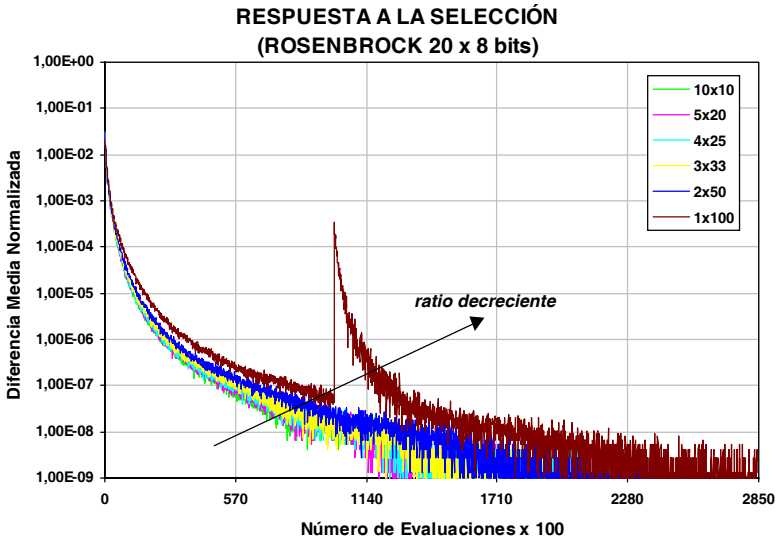
En la Figura 4.15 mostramos la respuesta a la selección de 5 problemas para distintas formas de la malla (6 ratii distintos) con una población de 100 individuos: 10x10, 5x20, 4x25, 3x33 y 1x100. Parece claro que problemas como la esfera, RAS20-8 y ROS20-8 presentan distintos tipos de respuesta a la selección para cada uno de los ratii estudiados. Esto explica por qué en ellos las rejillas más delgadas son ventajosas y mejoran la exploración, ya que las rejillas de menor ratii mantienen un decrecimiento de la mejora menos acusado que las rejillas cuadradas.



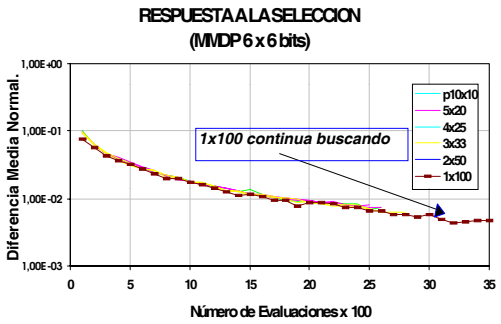
(a)



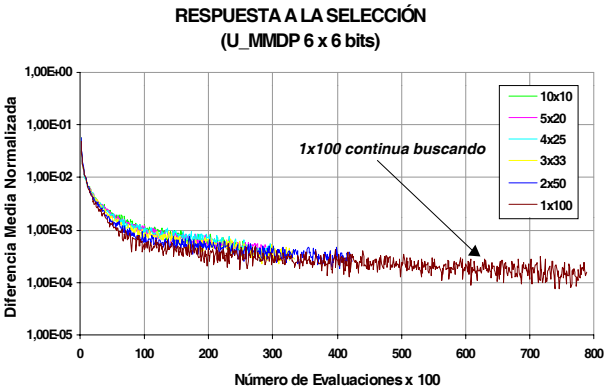
(b)



(c)



(d)



(e)

Figura 4.15. *Respuesta a la selección para 6 ratii distintos de un AG celular sobre los problemas de* (a) *SPH3-32,* (b) *RAS20-8,* (c) *ROS20-8,* (d) *MMDP6 y* (e) *ugly MMDP6.*

Sin embargo, en el caso de MMDP y su versión "fea" no se distinguen diferencias en la respuesta a la selección para los diferentes tipos de mallas. Esto explica que, ya que su respuesta a la selección es la misma, las mallas cuadradas y casi-cuadradas sean mejores puesto que son las más rápidas en converger (como ocurre para MMDP15, ..., MMDP40). Además, la mayor dificultad del problema se expresa en la Figura 4.15 como una mayor pendiente en la curva de la respuesta a la selección (rápido decremento en los incrementos de la adecuación media).

Las distintas gráficas muestran diferentes niveles de respuesta a la selección: desde los claramente distinguibles (esfera) hasta los indistinguibles (MMDP6), pasando por niveles intermedios (RAS20-8 y ROS20-8). Aunque es difícil distinguir las diferentes curvas, las flechas ayudan a comprobar el efecto de un ratio decreciente sobre la búsqueda.

En el caso de ROS20-8 (Figura 4.15c) mostramos un caso muy especial (el único observado) en el que la búsqueda conmuta de perseguir un tipo de solución a optimizar una solución distinta, entrando así en un nuevo ciclo de mejora de la media (sólo detectado para rejilla 1x100). Esto es muy infrecuente, aunque incluimos el resultado para demostrar que ofrecemos tanto los comportamientos típicos como las situaciones excepcionales.

En general, las distintas rejillas han demostrado tener ventajas relativas según el tipo de problema y respuesta a la selección. En el caso de problemas simples como el de la esfera la búsqueda de un AG celular puede resultar lenta en comparación con otros algoritmos donde la explotación está netamente presente en el modelo canónico (caso del modelo estacionario ssGA).

En algunas aplicaciones son necesarias poblaciones mucho más grandes (50x50 o 100x100) para mostrar la aparición de regiones (*clusters*, nichos, especies) y la difusión de cadenas. Esto es típico en implementaciones que usan máquinas SIMD. Sin embargo, podemos comprobar cómo este efecto está presente en poblaciones de tamaño mucho más modesto y práctico.

Una mayor diversidad no sólo tiene ventajas cualitativas sino también cuantitativas. Está demostrada la relación entre el decrecimiento de la varianza de la adecuación y el decrecimiento del número de esquemas presentes en la población [MTS93]. Esta relación refuerza la ventaja de utilizar mallas delgadas como primera alternativa en un AG celular para un nuevo problema.

4.3. Esfuerzo de Evaluación de los Modelos

Para comprobar el comportamiento relativo de los modelos secuenciales y paralelos presentamos en esta sección un estudio detallado sobre múltiples problemas. En primer lugar describimos los resultados sobre los problemas SPH3-8, MMDP5 y RAS20-8 para comparar entre sí el esfuerzo de evaluación necesario para su resolución y el porcentaje de éxitos conseguido con cada algoritmo. En esta primera evaluación usamos poblaciones extremadamente pequeñas para probar al límite las características de cada modelo.

En los modelos paralelos una copia de una cadena aleatoriamente seleccionada es enviada a la población vecina en el anillo cada $32 \cdot PoblaciónTotal$ evaluaciones paralelas (débilmente acopladas). Ofrecemos también la relación del esfuerzo de evaluación de otros algoritmos sobre dichos problemas y confirmamos gráficamente las diferentes curvas consecuencia de las diferentes presiones selectivas presentes en cada algoritmo. Finalmente, realizamos una clasificación atendiendo a los resultados obtenidos de los mejores algoritmos.

4.3.1. Resultados Usando SPH3-8

El problema de la esfera es bastante estándar y sirve como base para las comparaciones. Sin embargo, usando únicamente 3 variables, su solución con un AG es extremadamente simple y puede conseguirse en unos cuantos cientos de evaluaciones. Aunque existen algoritmos aún más eficientes para resolver este problema es interesante comparar los resultados con los proporcionados por otros modelos existentes de AGPs.

Una población de 10 cadenas es suficiente para estudiarlo (cada variable codificada en 8 bits, p_c=1.0 y p_m=0.1). Aunque es interesante comparar nuestros algoritmos entre sí y con otros algoritmos, estos resultados iniciales limitan los estudios que podemos realizar al caso no distribuido, ya que difícilmente podemos crear islas de menos de 10 individuos.

Puede observarse (Tabla 4.4) que todos los modelos resuelven el problema (100%) excepto los generaciones y el celular con reemplazo "siempre". Este resultado ya ha sido comentado y seguirá apareciendo en los siguientes estudios. En estos casos mostramos la mejor adecuación final media (el óptimo se obtiene en 78,6432). Puede observarse además la relativa lentitud del AG celular (735 evaluaciones) respecto al estado estacionario (410 evaluaciones) debido a la simplicidad del problema y a la clara ventaja de un algoritmo con alta explotación.

TABLA 4.4. RESULTADOS CON EL PROBLEMA SPH3

Modelo	Min #evaluaciones	Media #evaluaciones	Max #evaluaciones	% Éxito
*xx*GA(_,10,_,_,ss,alw)	220	428	740	100%
*xx*GA(_,10,_,_,ss,ifb)	200	410	600	100%
*xx*GA(_,10,_,_;gen,e)	23.600	28.200	32.800	(a-media 77,41) 83%
*xx*GA(_,10,_,_,gen,ne)	>200.000	>200.000	>200.000	(a-media 77,73) 0%
*xx*GA(2,5,r,20,ss,alw)	280	438	780	100%
*xx*GA(2,5,r,20,ss,ifb)	260	452	640	100%
*xx*GA(2,5,r,4,gen,e)	2.890	14.482	25.230	100%
*xx*GA(2,5,r,4,gen,ne)	>200.000	>200.000	>200.000	(a-media 77,78) 0%
*xx*GA(_,3x3,_,_,cel,alw)	38.808	38.808	38.808	(a-media 77,77) 20%
*xx*GA(_,3x3,_,_,cel,ifb)	396	735	900	100%
*xx*GA(???,3x3,_,_,cel,alw)	*Ya que la población global son 10 individuos*			
*xx*GA(???,3x3,_,_,cel,ifb)	*no podemos distribuir ni siquiera mallas de tamaño mínimo (3x3)!*			

En general, las comparaciones con otros algoritmos son injustas a menos que se utilicen idénticos conjuntos de parámetros. Por ejemplo, usando HSDGA [VSB92] sobre el mismo problema con 9 individuos se necesitan 344 evaluaciones de media para resolverlo. Este resultado es comparable al nuestro aunque nosotros usamos versiones canónicas de nuestros modelos y HSDGA usa búsqueda local y paralelismo real para conseguir este resultado. Igualmente, el sistema DGENESIS [Cant94] necesita un esfuerzo considerable para resolverlo usando un anillo o hipercubo de 16 islas (y genes de 10 bits). Quizás el resultado más claro se refiere al despegue de los algoritmos generacionales del resto por sus resultados tan negativos. La deriva genética en poblaciones tan pequeñas es un grave problema de este tipo de AG.

4.3.2. Resultados Usando MMDP5

El problema MMDP5 es considerablemente más difícil que el anterior. Los resultados (Tabla 4.5), sin embargo, confirman los anteriormente obtenidos respecto al mal comportamiento de los modelos generaciones y celular con reemplazo "siempre". El elitismo resulta casi indispensable en cualquier aplicación compleja o real tanto como está demostrando serlo en estos problemas.

Podemos observar en la tabla cómo la distribución de los modelos mejora casi siempre sus resultados. En este caso el mejor algoritmo es un modelo distribuido con islas estacionarias, ya que la población distribuida es aún pequeña para que una evolución celular pueda ser ventajosa.

TABLA 4.5. RESULTADOS CON EL PROBLEMA MMDP5

Modelo	Min #evals	Media #evals	Max #evals	Adec. Media	% Éxito
*xx*GA(_,30,_,_,ss,alw)	940	5.494	17.200	5,000	100%
*xx*GA(_,30,_,_,ss,ifb)	1.740	4.002	13.280	5,000	100%
*xx*GA(_,30,_,_,gen,e)	>300.000	>300.000	>300.000	4,504	0%
*xx*GA(_,30,_,_,gen,ne)	>300.000	>300.000	>300.000	4,404	0%
*xx*GA(2,15,r,60,ss,alw)	1.680	5.030	11.600	5,000	100%
*xx*GA(2,15,r,60,ss,ifb)	820	2.818	5.420	5,000	100%
*xx*GA(2,15,r,4,gen,e)	8.500	8.500	8.500	4,570	10%
*xx*GA(2,15,r,4,gen,ne)	>300.000	>300,000	>300.000	4,420	0%
*xx*GA(_,6x5,_,_,cel,alw)	>300.000	>300.000	>300.000	4,428	0%
*xx*GA(_,6x5,_,_,cel,ifb)	3.000	7.110	13.800	5,000	100%
*xx*GA(2,3x5,r,4,cel,alw)	>300.000	>300.000	>300.000	4,484	0%
*xx*GA(2,3x5,r,4,cel,ifb)	2.550	5.715	9.000	5,000	100%
Otros Modelos	**Min #evals**	**Media #evals**	**Max #evals**	**Adec. Media**	**% Éxito**
AG selección *disruptiva* (6 bits)	---	6.400	---	1,000	100%

Se pueden observar algunos detalles interesantes en estos resultados. El reemplazo "siempre" también empieza a ser claramente peor que el reemplazo "si mejor" en un modelo de estado estacionario (no solamente en uno celular). Por otro lado, puede observarse cómo la distribución en forma de dos islas generacionales (caso elitista) consigue solucionar algunas veces el problema. En cambio, el modelo genGA elitista secuencial siempre cae en un óptimo local, tal como ocurre también con los dos modelos generacionales no elitistas (confirmando los resultados de [GW93]).

Podemos observar además que la diferencia entre el modelo estacionario y el celular se reduce en este problema más difícil. Además ambos, en sus versiones distribuidas con sólo dos subpoblaciones ("si mejor"), consiguen reducir el esfuerzo para encontrar una solución (excepto ssGA "siempre"). Si comparamos el esfuerzo de nuestros modelos con otros modelos basados por ejemplo en usar un tipo especial de selección los resultados son muy esperanzadores, ya que en [KH96] se resuelve MMDP1 (1 subproblema, l=6) en 6.400 evaluaciones y casi todos nuestros modelos resuelven MMDP5 (5 subproblemas, l=30) en menos evaluaciones.

4.3.3. Resultados Usando RAS20-8

El problema RAS20-8 presenta una dificultad considerable para una búsqueda usando AGs. En el caso de usar 20 variables ofrecemos en la Tabla 4.6 los resultados de utilizar una población de 60 individuos y una de 180 (p_c=1.0 y p_m=0.01). En el primer caso únicamente los modelos estacionarios y celulares a veces encuentran una solución (éxito de entre el 20% y 30% de las ejecuciones). En el caso de usar 180 individuos los porcentajes mejoran, aunque únicamente el modelo celular "si mejor" consigue un porcentaje del 100%.

Las ventajas de usar un modelo distribuido (4 islas de 15 individuos o 5 de 36 en los casos de población total 60 o 180, respectivamente) son bastante notables en el caso estacionario son reemplazo "si mejor" (reducción en un factor de 1,76 al pasar de 1 a 5 islas).

Sin embargo, el modelo estacionario con reemplazo "siempre" y el celular "si mejor" empeoran ligeramente al ser distribuidos. Esta es una desventaja de los modelos síncronos presentados en esta sección debido además a que las subpoblaciones son demasiado pequeñas para conseguir ventajas al distribuir un problema con cadenas de elevada longitud.

Este caso, junto con algunos mencionados y también algún caso puntual presentado en este capítulo, demuestran que genéricamente no es posible afirmar que un modelo paralelo siempre sea mejor que otro secuencial. Un acoplamiento excesivo o un tamaño de población en las islas demasiado pequeño para el problema puede conducir a resultados en los que el impacto del paralelismo no sólo no sea beneficioso, sino incluso negativo.

En relación a otros algoritmos (distribuidos o no), los resultados obtenidos son buenos, ya que con configuraciones equivalentes GENITOR, GENITOR II y otros modelos celulares [GW93] no pueden competir con nuestro modelo celular, aunque sí con nuestras versiones estacionarias o celular distribuida (debido a la pequeña población global usada). El modelo DGENESIS obtiene unos resultados comparables, pero para 10 variables, no para 20 como estamos usando aquí. Finalmente, HSDGA sí que es bastante competitivo debido a usar menos individuos y requerir un número de evaluaciones ligeramente menor que nuestro modelo celular, aunque no debemos olvidar su elevada sofisticación en la búsqueda con operaciones nada estándares.

TABLA 4.6. RESULTADOS SOBRE RAS20

Modelo - PARAMS1	**Min #evals**	**Media #evals**	**Max #evals**	**Adec. Media**	**% Éxito**
*xx*GA(_,180,_,_,ss,alw)	88.000	112.433	158.300	0,005	30%
*xx*GA(_,180,_,_,ss,ifb)	79.800	161.033	270.300	0,003	30%
*xx*GA(_,180,_,_,gen,e)		>400.000		0,190	0%
*xx*GA(_,180,_,_,gen,ne)		>400.000		0,196	0%
*xx*GA(5,36,r,72,ss,alw)	48.000	141.500	235.000	0,003	20%
*xx*GA(5,36,r,72,ss,ifb)	38.750	91.333	158.500	0,002	30%
*xx*GA(5,36,r,2,gen,e)		>400.000		0,149	0%
*xx*GA(5,36,r,2,gen,ne)		>400.000		0,192	0%
*xx*GA(_,3x60,_,_,cel,alw)		>400.000		0,183	0%
*xx*GA(_,3x60,_,_,cel,ifb)	71.460	96.426	129.960	0,000	100%
*xx*GA(5,6x6,r,2,cel,alw)		>400.000		0,175	0%
*xx*GA(5,6x6,r,2,cel,ifb)	88.560	136.620	194.580	0,003	70%
Modelo - PARAMS2	**Min #evals**	**Media #evals**	**Max #evals**	**Adec. Media**	**% Éxito**
*xx*GA(_,60,_,_,ss,alw)	*60 individuos es una población*			0,01	0%
*xx*GA(_,60,_,_,ss,ifb)	*excesivamente pequeña*			0,01	10%
*xx*GA(_,60,_,_,gen,e)				0,2	0%
*xx*GA(_,60,_,_,gen,ne)				0,19	0%
*xx*GA(4,15,r,120,ss,alw)				0,007	10%
*xx*GA(4,15,r,120,ss,ifb)	**>300.000**			0,007	20%
*xx*GA(4,15,r,8,gen,e)				0,128	0%
*xx*GA(4,15,r,8,gen,ne)				0,197	0%
*xx*GA(_,6x10,_,_,_,alw)				0,187	0%
*xx*GA(_,6x10,_,_,_,ifb)	*Sólo ssGA con reemplazo "si mejor" y*			0,008	10%
*xx*GA(4,3x5,r,8,cel,alw)	*cGA, junto con sus versiones distribuidas*			0,185	0%
*xx*GA(4,3x5,r,8,cel,ifb)	*son capaces de resolver el problema*			0,007	30%
Otros Modelos	**Min #evals**	**Media #evals**	**Max #evals**	**Adec. Media**	**% Éxito**
GENITOR	>400.000	>400.000	>400.000		0%
GENITOR II	400.000	400.000	400.000		76,6%
Celular (Mejor+actual, ifb)	400.000	400.000	400.000		80%
DGENESIS (10 variables)	---	112.576	---	0,000	100%
HSDGA (3x33 individuos)	160	17.266	32.416	0,000	100%

4.3.4. Clasificación Preliminar sobre los Problemas Analizados

En esta sección ofrecemos una clasificación de los modelos derivados de *xx*GA atendiendo primero al porcentaje de éxito y después a su esfuerzo de evaluación (primamos la capacidad de resolver un problema y posponemos a un segundo nivel su eficiencia, cuando sea capaz de resolverlo). La Tabla 4.7 muestra para los tres problemas estudiados que el modelo estacionario y celular junto con sus versiones distribuidas (reemplazo "si mejor" en ambos casos) son los mejores. El modelo celular sólo justifica su uso en poblaciones grandes y problemas complejos. Mostramos sombreados los modelos que nunca han encontrado una solución.

En algunos casos los modelos paralelos han sido peores debido, primero, a que la distribución dispone de poblaciones relativamente pequeñas para mostrar claras ventajas, y segundo, a que el operador de migración apenas ha tenido oportunidad de mostrar sus ventajas. En los resultados de las siguientes secciones profundizaremos en estos detalles en problemas más complejos. También hemos detectado una cierta tendencia del modelo estacionario a comportarse bien en problemas donde al final de la evolución es necesario refinar el resultado, frente al modelo celular que resulta conveniente cuando la exploración es el elemento principal de la búsqueda.

TABLA 4.7. RESUMEN DE RESULTADOS PARA SPH3-8, MMDP5 Y RAS20-8

#	SPH3-8		MMDP5		RAS20-8	
1	*xx*GA(_,10,_,_,ss,ifb)	410	*xx*GA(2,15,r,60,ss,ifb)	2.818	*xx*GA(_,3x60,_,_,cel,ifb)	96.426
2	*xx*GA(_,10,_,_,ss,alw)	428	*xx*GA(_,30,_,_,ss,ifb)	4.002	*xx*GA(5,6x6,r,2,cel,ifb)	136.620
3	*xx*GA(2,5,r,20,ss,alw)	438	*xx*GA(2,15,r,60,ss,alw)	5.030	*xx*GA(5,36,r,72,ss,ifb)	91.333
4	*xx*GA(2,5,r,20,ss,ifb)	452	*xx*GA(_,30,_,_,ss,alw)	5.494	*xx*GA(_,180,_,_,ss,alw)	112.433
5	*xx*GA(_,3x3,_,_,cel,ifb)	735	*xx*GA(2,3x5,r,4,cel,ifb)	5.715	*xx*GA(_,180,_,_,ss,ifb)	161.033
6	*xx*GA(2,5,r,4,gen,e)	14.482	*xx*GA(_,6x5,_,_,cel,ifb)	7.110	*xx*GA(5,36,r,72,ss,alw)	141.500
7	*xx*GA(_,10,_,_,gen,e)	28.200	*xx*GA(2,15,r,4,gen,e)	8.500	*xx*GA(_,180,_,_,gen,e)	>400.000
8	*xx*GA(_,3x3,_,_,cel,alw)	38.808	*xx*GA(_,30,_,_,gen,e)	>300.000	*xx*GA(_,180,_,_,gen,ne)	>400.000
9	*xx*GA(_,10,_,_,gen,ne)	>200.000	*xx*GA(_,30,_,_,gen,ne)	>300.000	*xx*GA(5,36,r,2,gen,e)	>400.000
10	*xx*GA(2,5,r,4,gen,ne)	>200.000	*xx*GA(2,15,r,4,gen,ne)	>300.000	*xx*GA(5,36,r,2,gen,ne)	>400.000
11			*xx*GA(_,6x5,_,_,cel,alw)	>300.000	*xx*GA(_,3x60,_,_,cel,alw)	>400.000
12			*xx*GA(2,3x5,r,4,cel,alw)	>300.000	*xx*GA (5,6x6,r,2,cel,alw)	>400.000

En todos estos resultados preliminares no se ha hecho ningún esfuerzo por mejorar la selección de los parámetros usados, la codificación o los operadores para presentar el comportamiento básico. De haber sido así los resultados hubiesen sido notablemente mejores, pero pretendemos estudiar los problemas sin conceder ventajas a los algoritmos.

TABLA 4.8. CLASIFICACIÓN

1. dssi, dci, dssa
2. ssi, ci, ssa
3. dsuse, drwe
4. suse, rwe
5. dsusn,susn
6. ca, dca
7. drwn, rwn

Los resultados anteriores confirman la clasificación preliminar (Tabla 4.8). La distribución permite, bien encontrar un óptimo cuando la versión secuencial no lo consigue, bien disminuir el esfuerzo de cómputo. Sin embargo, aún es difícil distinguir claramente entre los modelos estacionario y celular (puestos 1 y 2).

4.4. Eficiencia y Ganancia de Velocidad en los Modelos Paralelos Distribuidos

En esta sección estudiamos los modelos paralelos distribuidos dssGA y dcGA desde múltiples puntos de vista. En la primera subsección presentamos exhaustivamente su comportamiento sobre el problema de la esfera generalizado, utilizando 16 variables de 32 bits cada una (l=512 bits). En todos los resultados se utiliza una población global de $512/n_{proc}$ individuos, p_c=1.0 y p_m=1/l. En la segunda subsección se abordan problemas de mayor complejidad que permitan aclarar y dar un contexto real a los resultados iniciales.

Las ejecuciones independientes se han obtenido con una versión paralela distribuida ejecutada en n_{proc} máquinas en una red local de procesadores idénticos (SUN UltraSparc1) comunicados por ATM. Puesto que la frecuencia de migración afecta al comportamiento de los algoritmos, cada resultado se presenta al menos para 3 valores del tiempo de aislamiento: fuerte acoplamiento (1), acoplamiento medio (2/4/16 según el problema) y casi-aislamiento (4/16/32 según el problema). Además, estudiamos tanto los modelos síncronos **s** como los asíncronos **a**.

4.4.1. Un Estudio Exhaustivo sobre el Problema SPH16-32

Realizamos en esta sección un análisis del esfuerzo numérico (Sección 4.4.1.1), el efecto de la sincronización entre las islas (Sección 4.4.1.2) y la ganancia de velocidad -*speedup*- (Sección 4.4.1.3) medidos sobre el problema SPH16-32 para trazar la línea base para futuras comparaciones. Nuestro objetivo es profundizar en numerosos aspectos de la ejecución paralela distribuida tanto del modelo estacionario como celular a través de diferentes parametrizaciones y tipos de ejecución.

4.4.1.1. Esfuerzo de Evaluación

Respecto al esfuerzo de evaluación para resolver SPH16-32, los resultados muestran claras conclusiones (Figura 4.16). Debido a su simplicidad, las frecuencias altas de envío de emigrantes (similar a una única población) resultan en un menor número de pasos para solucionar el problema. Este resultado es excepcional y sólo se encuentra en problemas simples. En estos casos la única ventaja del AGP distribuido es un tiempo de ejecución menor que el secuencial.

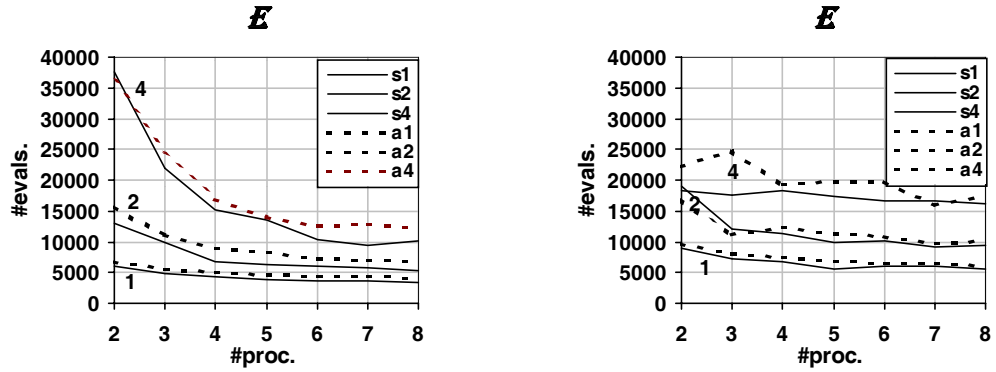


Figura 4.16. *Número de evaluaciones para dssGA* (izqda.) *y dcGA* (dcha.). *Se muestran tres frecuencias de migración* -1, 2, 4- *para versiones síncronas* -s- *y asíncronas* -a-.

Numéricamente el modelo dssGA es más sensible al uso de un mayor número de procesadores (lo que resulta ventajoso) mientras que dcGA es más estable y presenta un esfuerzo similar ligeramente reducido con el número de procesadores. La razón de nuevo radica en la simplicidad del problema para cGA, pues sólo en problemas complejos se justifica su uso.

4.4.1.2. Algoritmos Síncronos vs. Asíncronos

Para realizar una comparación justa los algoritmos síncronos y asíncronos ejecutan exactamente el mismo tipo de búsqueda con la diferencia de que los primeros acomodan a los emigrantes en ciertos pasos concretos periódicamente distribuidos durante la búsqueda, mientras que los segundos lo hacen en cuanto reciben al emigrante. En la figura anterior (4.16) puede observarse la similitud del esfuerzo de evaluación resultante.

Sin embargo, el tiempo de ejecución cambia dramáticamente entre ambas versiones dando lugar en el caso asíncrono a algoritmos más eficientes (Figura 4.17). En el problema SPH16-32 esto se comprueba más claramente para un menor número de procesadores (1,2,3,4), ya que para 5 o más procesadores todas las versiones distribuidas son muy rápidas en tiempo real.

La imposición de sincronización es una clara desventaja para todas las versiones estudiadas de algoritmos paralelos distribuidos [AT99b]. Este hecho se confirmará en las siguientes secciones sobre problemas más complejos y en términos de ganancias de velocidad en la próxima sección.

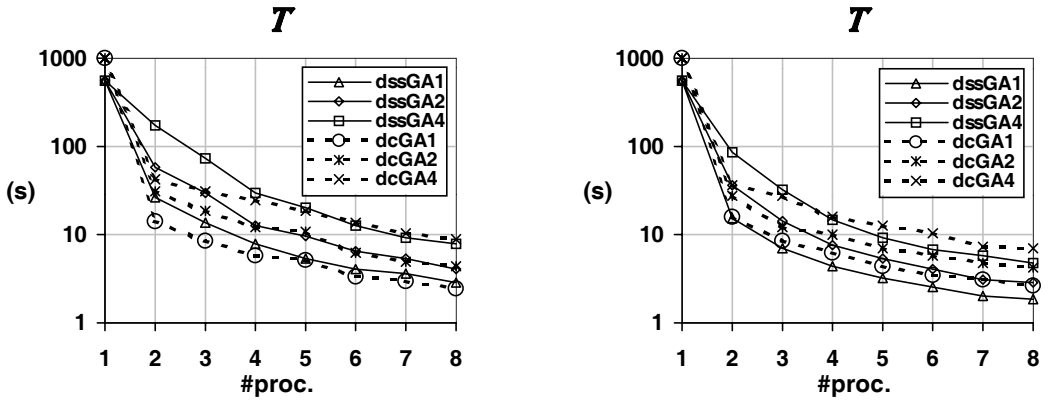


Figura 4.17. *Tiempo para resolver SPH16-32 con dssGA y dcGA. Se muestran dos comparaciones separadas: para los algoritmos síncronos* (izqda.) *y asíncronos* (dcha.).

Las diferencias en tiempo de computación de los modelos dssGA y dcGA no son muy significativas porque el problema es simple y se resuelve en todos los casos con considerable rapidez. La tendencia en ambos casos es una clara reducción del tiempo de ejecución con el número de procesadores y una clara ventaja de sus versiones asíncronas sobre las síncronas.

4.4.1.3. Ganancias en Velocidad (*Speedup*)

En esta sección presentamos la relación entre el tiempo de ejecución de los algoritmos secuenciales respecto a sus versiones distribuidas usando un número variable de procesadores. Como ya comentamos en el primer capítulo, las versiones distribuidas reducen no sólo el tiempo de ejecución, sino el número de evaluaciones necesarias para encontrar una solución.

Esto implica que el criterio de parada en todos los casos debe ser alcanzar el óptimo, y no otro [CG97] [HBBK97]. Además, la reducción en ambos aspectos nos hace esperar ganancias superlineales como las de otros autores con otros modelos de AGs paralelos [Beld95] [Shon93].

En la Figura 4.18 mostramos el *speedup* de dssGA y dcGA sobre el problema SPH16-32. Podemos concluir en todos los casos que las ganancias son claramente superlineales, ya que, por ejemplo, el tiempo en resolver el problema con 8 procesadores es menor que el tiempo del modelo secuencial equivalente en un factor bastante mayor que 8. Esto se debe a las ventajas de una población descentralizada y una mejor diversidad durante la búsqueda. Puede observarse que la ganancia es menor para los intervalos de migración elevados (como es en este caso el valor 4).

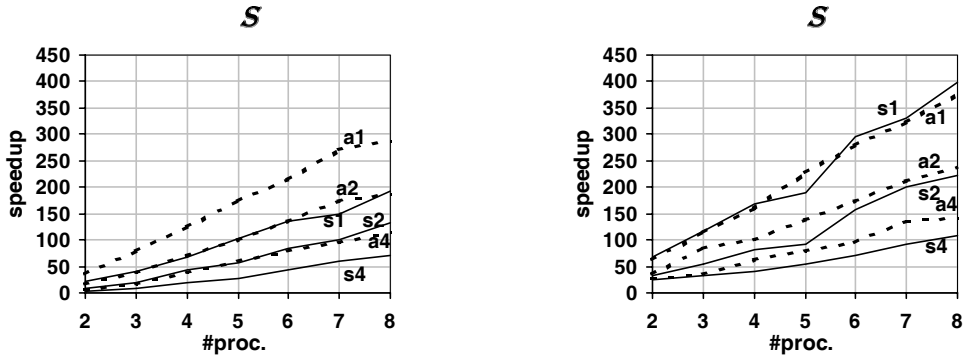


Figura 4.18. *Speedup para dssGA* (izqda.) *y dcGA* (dcha.).

En la sección anterior ya mostramos cómo las versiones con islas celulares (dcGA) eran a veces mejores en tiempo real para este problema. Ahora añadimos que son mejores en cuanto a conseguir ganancias en el tiempo a medida que aumenta el número de procesadores (compare las gráficas izquierda y derecha de la Figura 4.18, especialmente para los modos síncronos).

Pero no sólo esto, sino que si analizamos la ganancia (*speedup*) numérica (relación entre número de evaluaciones en secuencial y paralelo) vemos que también el modelo dcGA es mejor que dssGA incluso en este problema simple (Figura 4.19).

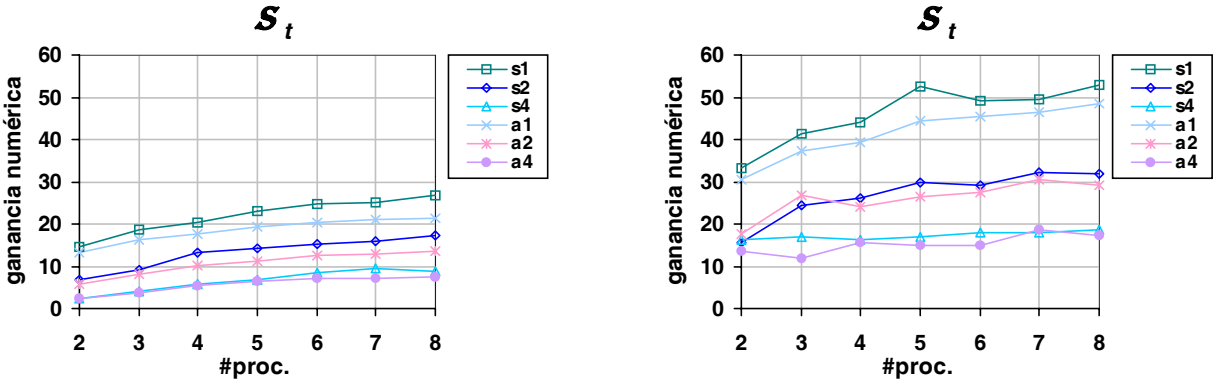


Figura 4.19. *Ganancias numéricas para dssGA* (izqda.) *y dcGA* (dcha.).

De los resultados presentados en esta sección se desprende igualmente la similitud de la ganancia numérica entre los modelos síncronos y asíncronos, no así la ganancia en tiempo, ya que los modelos asíncronos siempre son mejores en este aspecto que los síncronos (Figura 4.18).

Finalmente, presentamos la ganancia en tiempo de los modelos distribuidos respecto a la versión también distribuida con frecuencia de migración 4 (Ecuación 4.2). Puede comprobarse en la Figura 4.20 que tanto dssGA como dcGA son considerablemente mejores que el modelo distribuido casi aislado para un pequeño número de procesadores (2,3,4). Para más de 4 procesadores la ganancia respecto a la versión casi aislada disminuye y casi se estabiliza.

$$I\left(n_{proc}\right) = \frac{T_{idle}}{T^{\zeta}_{n_{proc}}} \tag{4.2}$$

Al comparar con la versión paralela pero casi aislada las ganancias están en rangos más razonables y pequeños. Sin embargo, esta "ganancia" únicamente nos informa de cómo mejora/empeora con el número de procesadores el algoritmo atendiendo a distintas frecuencias de migración (no es una ganancia como tradicionalmente se la entiende). Igualmente, podemos concluir que para más de 4 o 5 procesadores las ganancias de usar un determinado intervalo de migración son constantes. En este aspecto, dssGA decrece más que dcGA, quien resulta algo inestable, aunque mostrando curvas ligeramente ascendentes a medida que usamos más procesadores (ventaja).

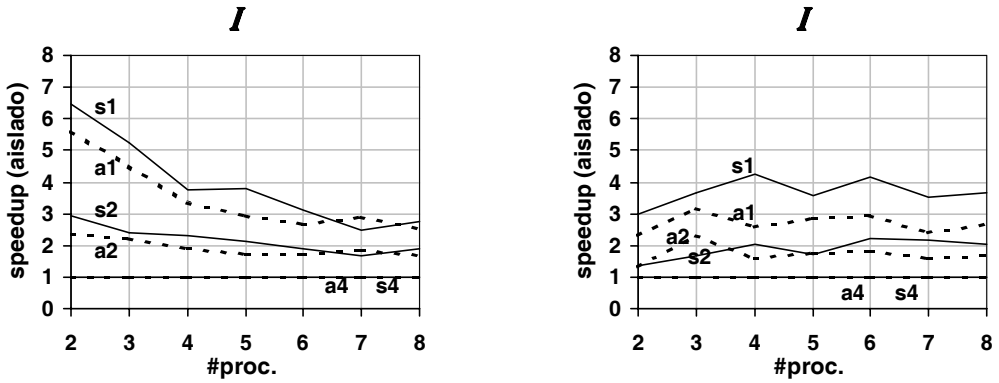


Figura 4.20. *Speedup respecto a una evolución aislada para dssGA* (izqda.) *y dcGA* (dcha.).

4.4.2. Extensión del Análisis a Problemas más Complejos

Resumiendo los resultados anteriores podemos concluir varios hechos. En primer lugar, las versiones asíncronas han sido siempre más rápidas que sus contrapartidas síncronas. Este resultado viene siendo confirmado con otros modelos de AGPs [MHK93] [MTS93] [HBBK97].

Igualmente, tanto en términos de ganancia (numérica o en tiempo) como en tiempo real dcGA con frecuencia proporciona mejores resultados, indicando que es posible incluso una aplicación de este modelo a problemas relativamente simples. Esto, unido a las ventajas demostradas sobre el resto de problemas de las secciones anteriores, hacen muy prometedor el uso de dcGA y cGA en aplicaciones complejas.

Hemos encontrado igualmente dos resultados excepcionales, ambos debidos a las características del espacio de búsqueda de SPH16-32. El primero es que los intervalos de migración pequeños son mejores. El segundo concierne a las altas ganancias obtenidas. En general, y para problemas más complejos, estas ganancias no son tan elevadas.

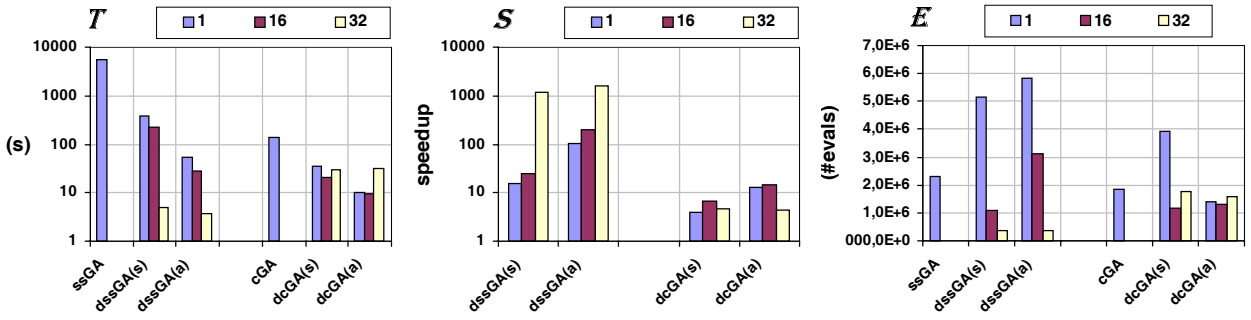
A pesar de las ventajas frecuentes de dcGA sobre dssGA no podemos recomendarlos de forma genérica. Este tipo de afirmaciones están teóricamente vetadas por el reciente teorema *No Free Lunch* (NFL) [WM97] y además tenemos evidencias [Alba92] [AAT93a-b] [AC97] de que el modelo dssGA se comporta excepcionalmente bien en dominios donde el refinamiento final de la solución (con parámetros continuos) es decisivo (incluso ante la presencia de epistasis o múltiples óptimos locales).

Un ejemplo de este tipo de dominios es el diseño de redes de neuronas usando algoritmos genéticos [AAT92] [AAT93a], en los que algoritmos distribuidos de estado estacionario como GENITOR II han resultado útiles [WH89]. El entrenamiento es una fase del aprendizaje de una red de neuronas que representa un problema excepcionalmente difícil para un AG (véase el Apéndice A para más detalles).

Por estas razones dedicamos las dos siguientes subsecciones a estudiar las conclusiones anteriores sobre este problema de considerable dificultad. Igualmente, extenderemos los resultados obtenidos sobre un problema NP-Completo como es el de la suma del subconjunto para contrastar en este nuevo campo el análisis ofrecido (de nuevo, el Apéndice A contiene los detalles sobre este problema).

4.4.2.1. Extensión del Análisis al Problema "Paridad4"

En la Figura 4.21 mostramos el tiempo real, *speedup* y esfuerzo de evaluación para entrenar la RN "Paridad4" (Apéndice A, Sección A.6) usando $512/n_{proc}$ individuos con ssGA y cGA y sus versiones distribuidas dssGA y dcGA sobre 8 procesadores (media de 100 ejecuciones independientes). La selección de uno de los padres es proporcional a la adecuación y aleatoria para elegir al otro padre (esto promociona la diversidad y la exploración). Utilizamos un cruce uniforme con p_c=1.0 (bias 0.8) y una mutación real con probabilidad p_m=0.01 (véase Capítulo 1).



(a) (b) (c)

Figura 4.21. *Tiempo real* (a)*, speedup* (b) *y esfuerzo de evaluación* (c) *entrenando "Paridad4".*

Podemos corroborar los resultados anteriores: la superioridad en tiempo real de los modelos asíncronos frente a los secuenciales y síncronos. Desde el punto de vista del número de evaluaciones es claramente perjudicial la migración tras cada generación. Se aprecia cómo 32 es el valor de la frecuencia de migración más beneficioso en la mayoría de los casos con dssGA (incluso hace que trabaje mejor que el equivalente dcGA) y similar a usar 16 en dcGA. Puede observarse en la Figura 4.21b que las ganancias son superlineales usando 8 procesadores. Los intervalos 1 y 16 son muy perjudiciales para dssGA, pero no tanto para dcGA.

Para un intervalo de migración de 1 o 16 el modelo celular es mejor que el estacionario, aunque para 32 no ocurre lo mismo. Sin embargo, debido a que existen pocos valores de adecuación posibles en la mayoría de los casos el modelo celular funciona mejor. En este problema, ya que las neuronas son binarias y que los posibles valores de adecuación son únicamente 17, la exploración del genotipo prima sobre la explotación porque la selección únicamente dispone de 17 valores distintos de adecuación, de ahí las ventajas de cGA y dcGA.

4.4.2.2. Extensión del Análisis al Problema "Tráfico"

En la Figura 4.22 presentamos la influencia de la sincronización y la frecuencia de migración usando dssGA y dcGA sobre 8 procesadores para entrenar la red "Tráfico" (Apéndice A, Sección A.7). Podemos confirmar cómo en este dominio de aplicación real, típico del aprendizaje máquina, las versiones asíncronas siguen comportándose considerablemente mejor que las síncronas (parámetros similares a los usados para "Paridad4"). Por un lado son más rápidas, y por otro lado son más insensibles al intervalo de migración (lo cual es una ventaja cuando el diseño lo realizan personas no expertas en computación evolutiva).

En este ejemplo, el uso de un algoritmo paralelo distribuido es imprescindible, ya que ninguna versión sobre un procesador con recursos equivalentes a la distribuida consiguió encontrar una solución de igual calidad (ni siquiera después de 15 o más horas de ejecución). Esto se debe a la combinación de características decepcionantes del problema abordado, en particular, alta epistasis, multimodalidad y aparición de *letales*. Debido a la alta correlación entre los pesos de una cadena, los hijos de dos cadenas de elevada adecuación pueden mostrar una adecuación mediocre, ya que el cruce mezcla trozos de cadena que no funcionan bien en el hijo [ES91].

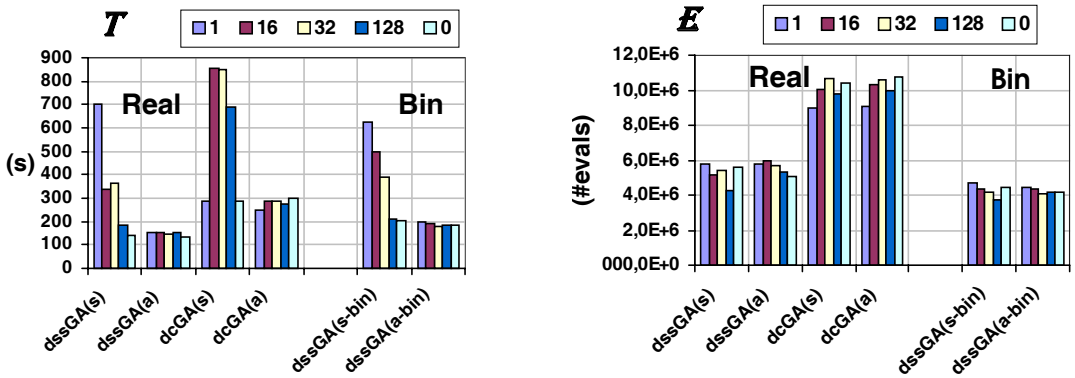


Figura 4.22. *Tiempo real* (izqda.) *y esfuerzo de evaluación* (dcha.) *para entrenar la red de neuronas "Tráfico" con dssGA y dcGA usando* 8 *procesadores. Se presentan los algoritmos síncronos* -**s**- *y asíncronos* -**a**- *para migración cada* 1*,* 16*,* 32*,*128 *gens. y aislamiento* -0-.

También se confirman los buenos resultados de utilizar altos intervalos de migración, tanto en tiempo real como en esfuerzo de evaluación. Ya que las neuronas utilizan una función de activación sigmoide todas las salidas son continuas y el problema requiere un alto refinamiento en su fase final, lo cual hace que los modelos celulares tengan una desventaja apreciable. Además, la fase de refinamiento final, lenta para todos los modelos ya que no se usa ningún operador especial para el problema, hace que los tiempos globales sean muy similares entre sí.

Finalmente, podemos comprobar cómo al codificar las cadenas en binario (8 bits por peso) el tiempo real de dssGA sube ligeramente como era de esperar [Mich92] debido a que las cadenas son mucho más largas y las operaciones del algoritmo tienden a ralentizarse respecto al caso flotante. No obstante, el efecto sobre el esfuerzo de evaluación es beneficioso debido a que el problema de búsqueda se simplifica [Bäck96]. La existencia de este tipo de aplicaciones en las que un AG celular es más lento justifica el interés por técnicas de búsqueda local para refinar más rápidamente sus individuos, como los trabajos descritos en [MSB91], [VSB92], y [Gorg97].

4.4.2.3. Extensión del Análisis al Problema de la Suma del Subconjunto

Para terminar esta extensión a problemas más complejos presentamos los resultados de resolver el problema de la suma del subconjunto en la Figura 4.23. Para ello utilizamos $512/n_{proc}$ individuos comparando ssGA y cGA y sus versiones distribuidas dssGA y dcGA usando $n_{proc} = 2$ y 8 procesadores (media de 100 ejecuciones independientes). La selección de parejas es proporcional a la adecuación para elegir ambos padres. Utilizamos un cruce DPX1 con p_c=1.0 y una mutación simple con probabilidad p_m=1/l.

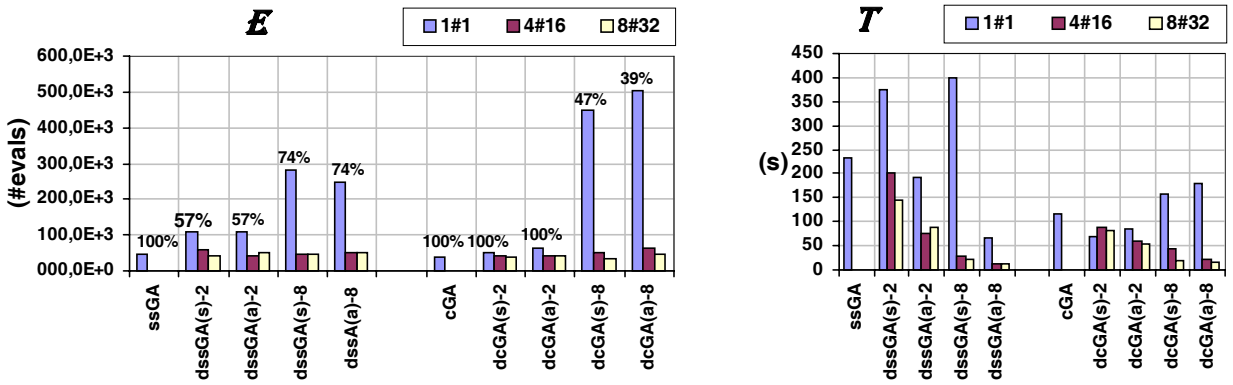


Figura 4.23. *Solución de SSS128 con los modelos estacionarios y celulares* (1, 2 *y* 8 *procesadores*): *número de evaluaciones* (izqda.)*, y tiempo real* (dcha.)*.*

En el caso de este problema NP-Completo todos los algoritmos requieren un considerable esfuerzo. El número de evaluaciones se ve influenciado muy negativamente por la frecuencia de migración 1 (dando lugar en algunos casos a un AGP distribuido peor que el modelo no distribuido). De hecho, esta frecuencia casi siempre impide encontrar una solución (notamos con el porcentaje de éxitos sobre las 100 ejecuciones cada barra de dicho intervalo de migración).

En todos los demás casos los algoritmos encontraron siempre una solución. En particular, el modelo celular sobre una y dos máquinas no fue influenciado por dicho intervalo, dando muestras de una interesante resistencia a una la elección arbitraria de un valor para dicho parámetro. Por ejemplo, para dos procesadores, dcGA obtuvo un 100% de éxitos mientras que dssGA obtuvo únicamente un 57% y además con mayor esfuerzo de evaluación.

Si analizamos el tiempo de ejecución comprobaremos que un intervalo de migración alto y una ejecución asíncrona son dos de las características más ventajosas. La Figura 4.24 ratifica las buenas ganancias de los modelos estacionarios y las casi-lineales ganancias de los modelos celulares.

A pesar de apenas rozar la ganancia lineal, las versiones celulares son las más rápidas en tiempo real, corroborando nuestra hipótesis de que dicho modelo es muy útil en problemas de elevadas necesidades de exploración.

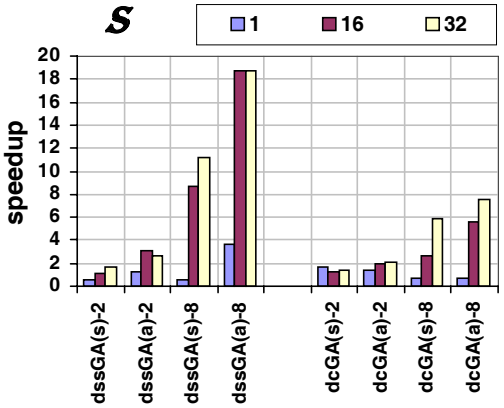


Figura 4.24. *Speedup al resolver SSS128 usando dssGA y dcGA sobre 2 y 8 procesadores.*

4.5. Crecimiento del Esfuerzo de Evaluación ante Problemas Progresivamente más Difíciles

En esta sección nuestro objetivo es estudiar el número medio de evaluaciones que un grupo de algoritmos derivables de *xx*GA necesita para encontrar una solución a medida que se enfrentan a problemas progresivamente más difíciles. En particular, trabajaremos con las versiones *xx*GA(...,ss,ifb), *xx*GA(...,cel,ifb) y *xx*GA(...,sus,e), distribuidas y no-distribuidas. Después analizaremos en mayor detalle los resultados ya presentados sobre el modelo celular por ser donde más necesidad de aportaciones existe en la actualidad.

4.5.1. Capacidad de Escalada General para Múltiples Modelos

En esta sección estudiamos la relación del crecimiento del esfuerzo con el del problema para las versiones secuenciales y distribuidas de los modelos estacionario, generacional y celular. Todas las versiones distribuidas utilizan 4 islas y migran cada 32 generaciones una cadena aleatoria. Los modelos generacionales utilizan el muestreo estocástico universal (SUS) para minimizar el error.

Permitimos en estos tests un máximo de $3 \cdot 10^5$ evaluaciones ($4{,}5 \cdot 10^5$ para Rosenbrock). Para cada familia y para cada algoritmo el conjunto de parámetros es único y constante. Esto supone que a medida que el problema crece cada algoritmo demostrará sus dificultades en resolverlo (hasta resultarle imposible en algunos casos). Los resultados pueden consultarse en la Figura 4.25 (las probabilidades usadas son las mismas que las de la Tabla 4.3).

Podemos observar varias tendencias en la Figura 4.25. El número de evaluaciones de los modelos generacionales crece de forma alarmante para saltos muy pequeños en el tamaño del problema. Esto es una gran desventaja, sobre todo indicando su dependencia de una afinación escrupulosa de los parámetros y técnicas usadas.

El modelo ssGA es ligeramente mejor en este aspecto para las versiones altas de RASxx-8 y ROSxx-8 debido a la aparición del problema de la afinación final de la solución y al pequeño número de subpoblaciones empleadas. Los resultados sobre MMDP16 son más concluyentes, ya que observamos cómo el modelo generacional rápidamente se queda fuera de la gráfica al crecer el problema y el modelo estacionario no puede resolver MMDP16. Se confirma de nuevo las ventajas de usar un modelo celular (distribuido o no).

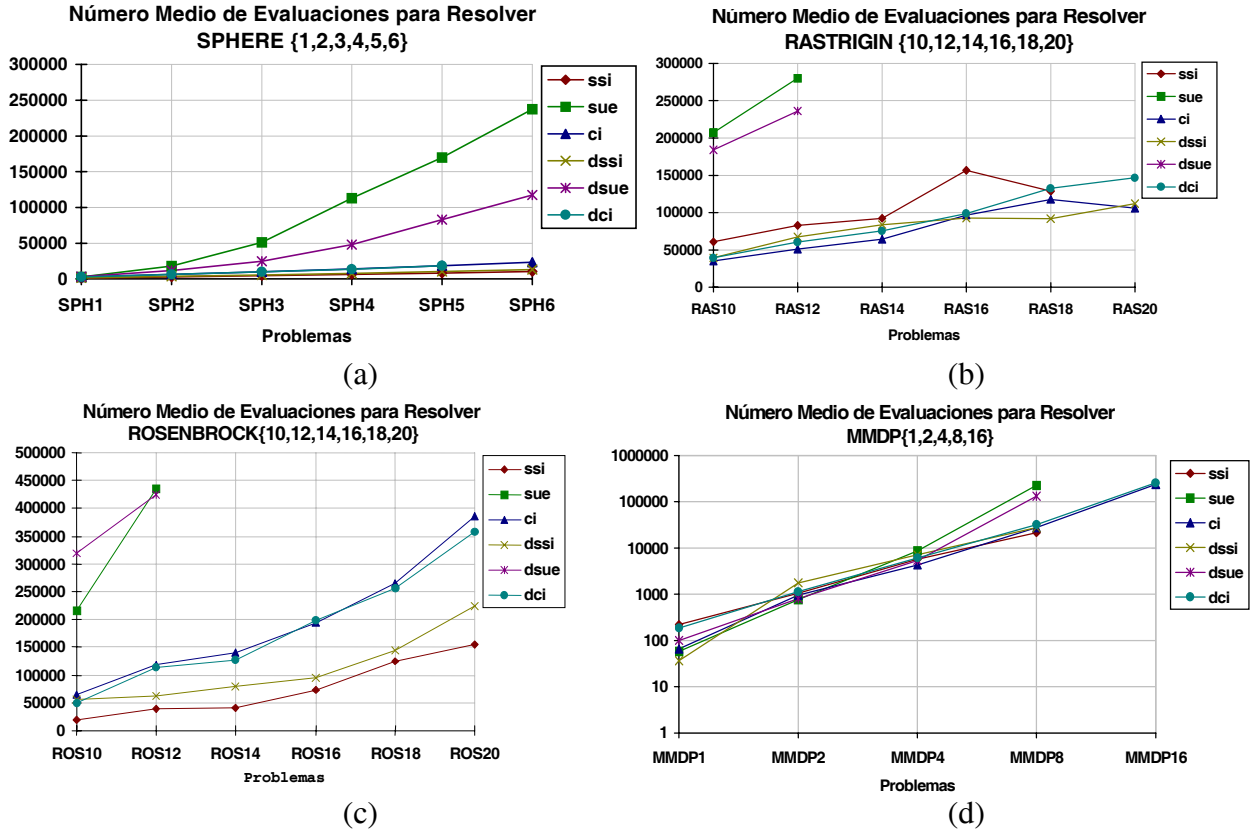


Figura 4.25. *Crecimiento del coste computacional para múltiples instancias de los problemas* (a) *SPHxx-32,* (b) *RASxx-8,* (c) *ROSxx-8 y* (d) *MMDPxx.*

Observe cómo la Figura 4.26 muestra el buen comportamiento de un AG celular (ROSxx-8) en cuanto al porcentaje de éxitos (no ya al número de evaluaciones) frente a todos los demás modelos. El porcentaje de éxitos para los modelos generacionales es bastante pobre. Con estos resultados concluimos este capítulo sobre el estudio de modelos y parámetros para pasar en el siguiente a resumir y relacionar las conclusiones alcanzadas en las pruebas numéricas realizadas.

En la siguiente sección profundizamos sobre estos detalles considerando únicamente el modelo celular. Nuestra intención es comprobar a fondo la influencia del ratio de la malla en este aspecto de la escalada del esfuerzo. La intención es aportar más datos que ayuden a conocer el comportamiento de un algoritmo genético celular usando distintas formas de malla, así como distintos problemas y rangos para el tamaño de las instancias la familia usada.

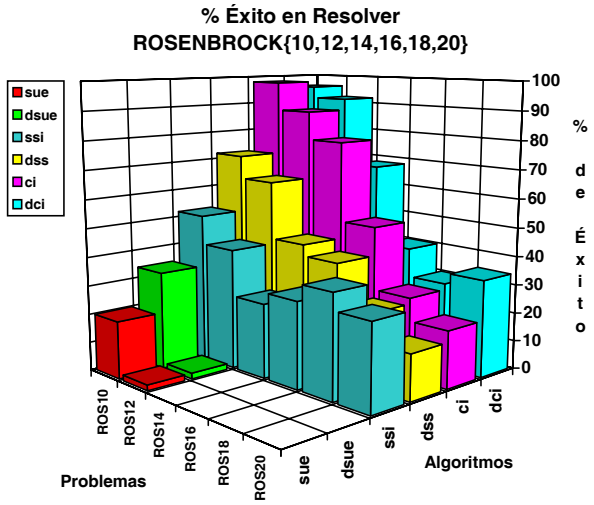


Figura 4.26. *Porcentaje de éxito en resolver múltiples instancias de la función Rosenbrock.*

4.5.2. Capacidad de Escalada en el Modelo Celular

En esta sección estudiamos la forma en que el mismo AG celular resuelve problemas de tamaño progresivamente mayor usando para todos ellos la misma parametrización. Esto da una idea de la explotación que el algoritmo hace la información presente en la población.

La Tabla 4.9 muestra los parámetros usados para cada familia. El número de evaluaciones se limita y utilizamos poblaciones tan pequeñas como sea posible típicas de aplicaciones reales. Podríamos usar poblaciones de 100x100 para dejar claros algunos conceptos como la formación de especies o nichos, pero unos resultados como esos no son útiles en la práctica, ya que el resto de algoritmos evolutivos no usan tales poblaciones.

TABLA 4.9. PARÁMETROS PARA RESOLVER SPHXX, RASXX, ROSXX Y MMDPXX

PARÁMETROS	SPH{1,2,3,4,5,6}	RAS{10,12,14,16,18,20}	ROS{10,12,14,16,18,20}	MMDP{1,2,4,8,16}
Bits por variable	32	8	8	6
Tamaño P (forma)	128 (2 x 64)	100 (2 x 50)	190 (2 x 95)	48 (2 x 24)
p_c	1.0	1.0	0.8	1.0
p_m	0.01	0.01	0.03	0.05
Max #evals.	300000	300000	450000	300000

En la Figura 4.27 trazamos el crecimiento del coste computacional para la batería de familias. De nuevo hacemos hincapié en que usamos un conjunto de parámetros constante para cada familia, de manera que el algoritmo que soluciona MMDP1 y MMDP16 (por ejemplo) es exactamente el mismo. Este tipo de estudios es poco frecuente porque normalmente cada problema requiere un cambio en el tamaño de la población, pero resulta interesante (aunque injusto y duro para los algoritmos) comprobar cómo responden los modelos estudiados.

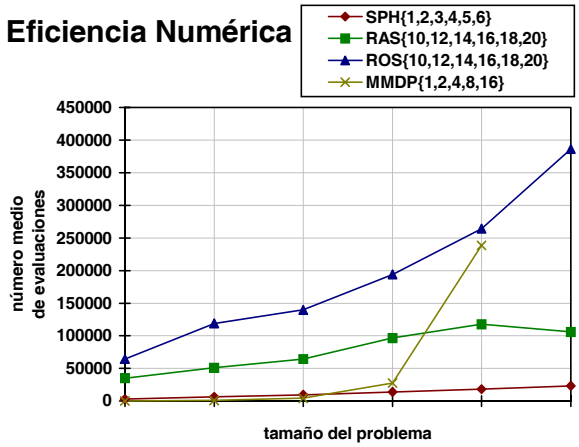


Figura 4.27. *Eficiencia* (*media de* 50 *ejecuciones*) *en resolver diferentes instancias de varias familias de problemas con un conjunto de parámetros constante en cada familia* -Tabla 4.9-.

Los problemas simples como el de la esfera son manejados adecuadamente sin grandes saltos en el esfuerzo requerido. Los problemas más complejos como ROSxx o RASxx sí que demuestran necesitar progresivamente un esfuerzo mayor para las grandes instancias, especialmente ROSxx debido la alta dependencia entre variables (epistasis).

Algo similar sucede con MMDPxx debido a que es un problema pensado para encaminar mal al algoritmo genético. El AG celular maneja bien hasta un tamaño de 8 subproblemas, mientras que para 16 el salto en el esfuerzo es considerable (dificultad notablemente mayor que para el resto de instancias: 1, 2, 4, 8).

En la Figura 4.28 podemos observar más claramente la tendencia cuando comparamos el crecimiento del esfuerzo con el crecimiento del tamaño del problema, para cada familia de problemas. El problema SPHxx es fácilmente manejado por el AG celular, mientras que ROSxx y RASxx muestran un crecimiento por encima del caso lineal de la esfera. La familia MMDPxx representa problemas de escalada para el AG celular debido a su naturaleza decepcionante (obsérvese que incluso necesitamos una escala logarítmica para el eje Y).

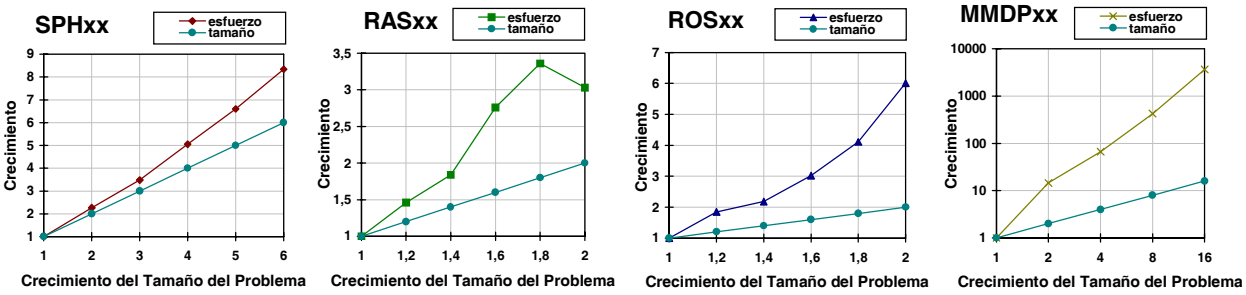


Figura 4.28. *Crecimiento del esfuerzo frente al crecimiento del tamaño del problema para cuatro familia de problemas distintos.*

Tras estos resultados podríamos pensar que el algoritmo tiene dificultades para escalar. Sin embargo lo que realmente se ha hecho es llevarlo hasta sus límites de funcionamiento para conocer su comportamiento. En la práctica es muy poco frecuente que problemas de distinto tamaño o dificultad, aunque sean de la misma familia, se resuelvan con el mismo tamaño de población. Para devolvernos la perspectiva podemos observar cómo es la relación del AG celular estudiado respecto a los más tradicionales de estado estacionario y generacional.

En la Figura 4.29 resumimos los resultados sobre la función de Rosenbrock sobre los modelos no distribuidos para apreciar más claramente cómo el AG celular es en promedio mejor respecto a las versiones secuenciales en panmixia. El porcentaje de éxito es mayor para casi todas las instancias estudiadas. A pesar de que, evidentemente, tenga limitaciones respecto al máximo tamaño que puede solucionar dada una configuración constante, su relación con los modelos de población no estructurada es bastante ventajosa. A veces incluso resuelve instancias que los otros algoritmos son incapaces de manejar.

La excepción a la superioridad de cGA se produce frente a ssGA para las instancias de 18 y 20 variables, en la que los problemas de la exploración dejan paso a los de explotación y afinación de los valores finales de las variables. Esta última característica está especialmente presente en un AG de estado estacionario [AAT93a-b].

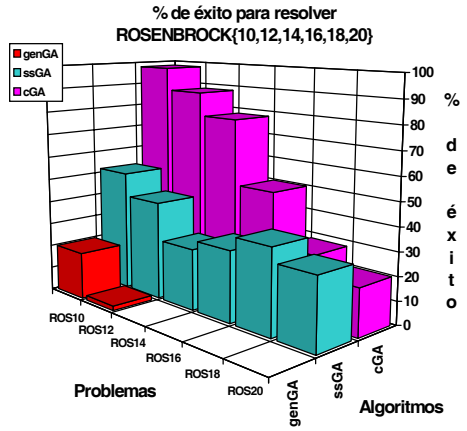


Figura 4.29. *Porcentaje de éxito* (*sobre* 50 *ejecuciones*) *en resolver diferentes instancias de la familia ROSxx-8. El algoritmo celular mejora los resultados del estacionario y del generacional.*

Para perfilar más en detalle el crecimiento del coste abordamos el especialmente difícil problema MMDP para instancias extremadamente grandes (15 a 40 subproblemas) usando una población de 1024 individuos en distintas formas de rejilla. En la Figura 4.30 izqda. observamos cómo las rejillas 32x32, 16x64 y 8x128 presentan un esfuerzo equivalente (obteniéndose 100% de éxito en *todos* los casos mostrados). Esto refuerza la ventaja de usar rejillas delgadas, ya que muestran el mismo esfuerzo en este caso que una cuadrada. Las rejillas excesivamente delgadas como 4x256 demuestran de nuevo su mayor necesidad de coste computacional.

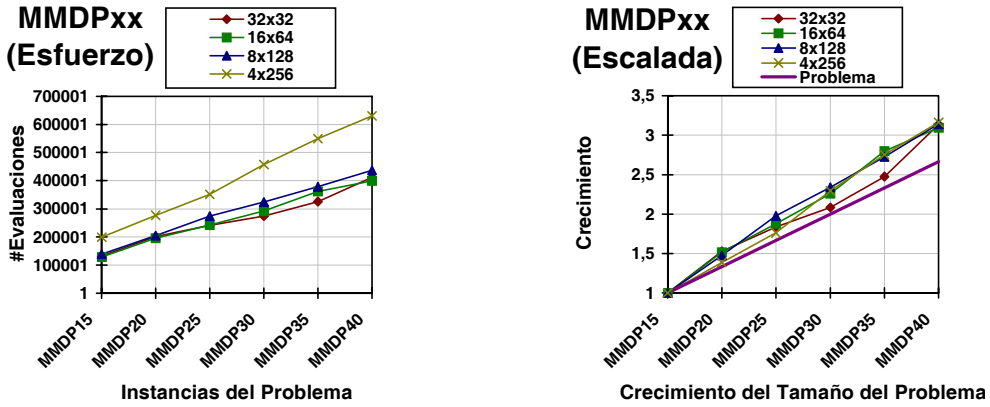


Figura 4.30. *Coste* (*#evals.*) *para obtener una solución con un AG celular* (izqda.) *y su perfil de escalada* (dcha.) *para diferentes ratii e instancias progresivamente mayores de MMDP. Los parámetros usados son* 1024 *individuos en total,* p_c=1.0 (*DPX1*), p_m=1/*l* (20 *ejecuciones*).

Respecto a la escalada del esfuerzo podemos ver cómo en poblaciones mayores que en los casos anteriores el crecimiento del coste computacional está mucho más cercano al crecimiento del problema. Una ligera ventaja puede notarse en el crecimiento de la malla cuadrada. Puesto que en términos absolutos las mallas delgadas han resultado mejores en porcentaje de éxitos y las mallas cuadradas en coste para resolver el problema podemos preguntarnos sobre qué problemas es de esperar que la forma de la malla sea mejor, o simplemente, tenga un efecto más decisivo. Quizás la respuesta a la selección sea la solución, tal como ya presentamos en la Sección 4.2.3.

5
Conclusiones y Trabajo Futuro

En este capítulo ofrecemos un resumen de las conclusiones alcanzadas en los capítulos anteriores. Asimismo, presentamos sus relaciones y algunos de los trabajos futuros de mayor interés a la luz de estas conclusiones.

Para ello, la Sección 5.1 contiene un sumario de los resultados más sobresalientes, así como la relación entre ellos. La Sección 5.2 discute brevemente los detalles más importantes respecto a la implementación de algoritmos genéticos y evolutivos paralelos, distribuidos o no. Las ventajas e inconvenientes de las técnicas y algoritmos estudiados se evalúan en la Sección 5.3, mientras que la Sección 5.4 contiene las conclusiones sobre los procesos de desarrollo y resultados de este trabajo. Para terminar, la Sección 5.5 plantea algunas cuestiones de interés para la investigación futura, algunas de los cuales ya están en marcha y han permitido dar solidez a las aportaciones.

5.1. Resumen de Resultados

Comenzamos este capítulo haciendo un resumen de los resultados obtenidos en el capítulo anterior desde un nivel de abstracción mayor al numérico. Para empezar, hemos observado cómo el *estudio unificado* permite resaltar las ventajas e inconvenientes de los distintos modelos secuenciales y paralelos que pueden extraerse de *xx*GA.

Del estudio *canónico* realizado hemos concluido la validez de nuestra caracterización de presión selectiva para cada modelo no distribuido: generacional, estado estacionario y celular. Este último modelo resalta sobremanera debido a que su *presión selectiva es modificable* fácilmente y permite así adaptarse al problema concreto. *El modelo estacionario provoca una alta presión selectiva* que es, en general, ventajosa, excepto en los casos de poblaciones muy pequeñas y/o problemas donde la exploración es la principal componente. El modelo *generacional* puro (μ,μ) ha demostrado siempre, en igualdad de condiciones, *trabajar peor en eficiencia numérica y porcentaje de éxito*, por lo cual se desaconseja como primera alternativa ante un nuevo problema.

Hemos estudiado para el modelo celular la influencia de la forma de la rejilla sobre su comportamiento, concluyendo que, en general, es preferible un enfoque de baja presión (alta exploración) aunque esto lleva aparejado un mayor esfuerzo de evaluación respecto a mayores presiones. Estos conceptos se han formalizado y cuantificado para ayudar en la elección futura de parámetros sobre nuevos problemas. De esta forma, podemos confirmar resultados que otros autores sugieren en sus estudios sobre otros aspectos del modelo celular [MSB91]. Planteamos además las ventajas de *cambiar la forma de la rejilla* frente a cambiar el tipo de [SD96] [SD97], ya que ambos pueden producir resultados equivalentes y la primera alternativa es más simple.

La influencia de la *política de reemplazo* en el modelo celular es también acusada, haciendo necesario un reemplazo elitista para evitar disminuir drásticamente la presión selectiva. Hemos comprobado las ventajas del modelo celular respecto al estacionario en la mayoría de problemas estudiados, excepto cuando es muy importante la necesidad de refinamiento de la solución que provoca el algoritmo. En esta situación el algoritmo de estado estacionario se comporta mejor que el celular. Con frecuencia, tal es el caso del entrenamiento de redes de neuronas.

Podemos también concluir las ventajas innegables de la distribución paralela de cualquiera de estos modelos básicos de evolución. Únicamente no se han producido ventajas en las instancias más simples y con un número muy pequeño de poblaciones de dos problemas de optimización funcional. Debido a la importancia de la frecuencia de migración en estos modelos un mal uso de dicho valor puede resultar en un algoritmo que presente un tiempo de ejecución y un esfuerzo de búsqueda incluso peor que su equivalente secuencial (como ocurre con SSS128 y frecuencia de migración 1). Para todo el resto de casos, la distribución ha demostrado aportar en la práctica sus ventajas teóricas.

Las ventajas teóricas mencionadas son su menor tiempo de ejecución, su disminución del esfuerzo de evaluación y sus capacidades de exploración paralela desde distintas regiones del espacio de búsqueda, asentadas en la mayor diversidad y mayor explotación dentro de cada isla (por usar poblaciones más pequeñas que sus contrapartidas de población única).

Es en este aspecto donde se encuentra la principal ventaja de cualquiera de los modelos estudiados. Nos referimos al *uso de una población estructurada*. Incluso en una ejecución sobre un único procesador, un algoritmo de población estructurada puede presentar ventajas.

Adicionalmente, los resultados en una ejecución distribuida real demuestran que todos los modelos estudiados son notablemente más rápidos en tiempo de ejecución consiguiendo incluso *ganancias superlineales*. Según la dificultad del problema, dichas ganancias son más o menos acusadas, pero en muchos de los casos estudiados son superlineales. Esto se debe a que el funcionamiento distribuido no sólo mejora el tiempo de ejecución, sino que afecta al esfuerzo de evaluación necesario para encontrar una solución (reduciéndolo).

La distribución mejora la diversidad en el caso estacionario, permitiendo disminuir la convergencia prematura. En el caso generacional hace al algoritmo más eficiente y capaz de encontrar soluciones en problemas donde la versión secuencial no lo hacía. Finalmente, la distribución de islas celulares posee una alta flexibilidad y eficiencia. Esto permite que algunos problemas complejos puedan resolverse de forma eficiente, gracias a las capacidades de exploración del modelo celular y a las ventajas en eficiencia de su ejecución distribuida (quien permite ejecutar incluso mallas estrechas muy eficientemente). La *desventaja del modelo celular* aparece cuando el problema requiere un alto nivel de *refinamiento final*, ya que este tipo de algoritmo posee una débil explotación. La solución es usar operadores de búsqueda local para mejorar la explotación en cada nodo de la rejilla, como por ejemplo un cruce que explore el potencial dinástico de cada pareja del tipo de DOR [CAT98b].

Los modelos distribuidos *asíncronos* han permitido extraer conclusiones muy claras respecto a la necesidad de evitar una alta sincronización entre las islas [ZK93]. Valores de 32 para el intervalo de migración han proporcionado los mejores resultados en problemas complejos. Los problemas simples unimodales y sin epistasis parecen necesitar una evolución similar a la de una población única debido a la carencia relativa de obstáculos en el espacio de búsqueda.

Para un estudio tan extenso como el realizado en esta memoria, la *política de migración* que selecciona a un individuo de manera *aleatoria* es mejor que enviar una copia del mejor individuo, ya que esta decisión está fuertemente relacionada con el intervalo de migración, produciendo claras situaciones de "conquista" para altas frecuencias o de "no efecto" para bajos intervalos de migración. El envío de una copia de la mejor solución puede devengar mejores resultados en algún problema, pero la libertad para la selección del resto de parámetros, en particular de la frecuencia de migración, es menor.

Las ventajas del uso de subpoblaciones en forma de *anillo* son notables respecto a la flexibilidad, facilidad, y éxito del sistema resultante. Aunque otras topologías, como por ejemplo el hipercubo [Loza96] [HL97], han demostrado ser igualmente interesantes en otros estudios, la simplicidad del anillo y su fácil y directa implementación en una red de estaciones hacen de ella una topología muy adecuada para paralelización real del sistema distribuido.

Las ganancias en velocidad y la capacidad de enfrentarse a problemas de complejidad creciente con recursos limitados ratifican las ventajas de los modelos distribuidos (y también del uso de islas celulares). De hecho, muchas de las ventajas de una buena *búsqueda teórica* están presentes en el modelo *celular*, mientras que muchas de las ventajas de una *búsqueda eficiente* están presentes en el modelo *distribuido*, por lo que el modelo dcGA merece mención aparte.

5.2. Evaluación de la Implementación

En esta sección pretendemos resumir las conclusiones respecto a la implementación de sistemas evolutivos paralelos en general, y genéticos en particular. La primera conclusión es que es mucho más práctico y flexible utilizar un lenguaje orientado a objetos para dicha implementación. Otros lenguajes tienen ventajas marginales, como por ejemplo el manejo de cadenas o el paralelismo de PARLOG [Alba92] [AAT93a-b], la simplicidad y manejabilidad de Modula-2, o la generalidad y eficiencia de lenguajes como C.

Sin embargo, en ninguno de los mencionados lenguajes es fácil conseguir las ventajas conjuntas de abstracción, extensibilidad, reutilización, ... de un lenguaje orientado a objetos como C++. El manejo de un gran número de parámetros y la similitud entre el sistema paralelo genético y el código es superior al que podemos conseguir en los otros lenguajes con un esfuerzo equivalente. Es posible además la extensión a sistemas más abstractos como los *marcos*.

De entre las conclusiones más interesantes podemos distinguir la importancia de tener una población *dinámica* que pueda manejarse rápidamente para no ralentizar de manera innecesaria la ejecución del algoritmo. Asimismo, el diseño de las clases de alelos y genes debe suponer un estudio previo de su uso para no malgastar innecesariamente recursos de memoria. Los conceptos presentes en el algoritmo distribuido o paralelo en general tales como *individuo*, *población* o *problema* aparecen en muchas implementaciones existentes. Sin embargo, no ocurre lo mismo con el concepto de *operador*, quién también debiera existir como clase aparte, como sugiere la existencia de un *plan reproductor* en el algoritmo.

La importancia de usar orientación a objetos es aún mayor cuando tenemos que mezclar modelos, cambiar fácilmente vecindarios o topologías y cuando queremos ocultar el sistema de comunicación al resto del algoritmo que lo usa.

En este aspecto, aunque la implementación es más laboriosa usando interfaces de comunicación como BSD *socket*, la eficiencia resultante es el criterio que debe primar. En el caso de los marcos de programación, estas características están también presentes, y hemos demostrado el paso de nuestro diseño a marcos definiendo los distintos niveles de éstos.

De esta forma, el sistema distribuido se ha construido creando clases que oculten los detalles y diseñando un sistema de procesos que controla tanto las tareas algorítimicas como el seguimiento de la ejecución y la extracción de resultados.

La interfaz de uso es un apartado que suele descuidarse en el diseño e implementación de *software* de investigación y que a veces se plantea como proyecto aparte. Sin embargo, un sistema configurable como el implementado, ya sea gráficamente y/o por ficheros, es a nuestro entender imprescindible para ofrecer como resultado final un sistema completo y no un conjunto de código desligado e incompleto desde el punto de vista de la completitud de un trabajo en Informática.

5.3. Ventajas e Inconvenientes

Entre las ventajas del uso de modelos como los presentados podemos destacar las siguientes. Las ventajas del modelo estacionario se derivan de su capacidad de explotación, que lo hacen adecuado en multitud de problemas. Las desventajas teóricas de convergencia prematura pueden manejarse con una mayor tasa de mutación o bien usando un modelo distribuido. De hecho, en muchos problemas, la gran dimensión del espacio de búsqueda ayuda a que esta desventaja no se haga patente (consúltense los resultados de entropía para el problema de la suma del subconjunto). Ya que este modelo es básicamente elitista resiste valores para estos parámetros que convertirían a otros algoritmos, como los generacionales, en búsquedas pseudo-aleatorias.

El modelo generacional tiene a su favor un teórico mantenimiento de diversidad y exploración que en la práctica es lento en comparación con el mismo conjunto de técnicas de un modelo estacionario. Sus ventajas (las del modelo generacional) pueden encontrarse igualmente en un modelo celular que también procede por generaciones, pero que proporciona mejores resultados frecuentemente debido a la disposición bidimensional de sus individuos. Muchas técnicas se han desarrollado para los modelos generacionales permitiendo así aumentar sus ventajas. Sin embargo, la mayoría de ellas pueden también aplicarse para potenciar las ventajas de los modelos estacionarios y celulares igualmente. En la actualidad, muchos investigadores utilizan mecanismos ajustables no generacionales y muchas veces cercanos al caso estacionario.

Las desventajas del modelo celular se hacen patentes en problemas complejos donde el refinamiento final de la solución es la principal característica. Nos referimos a problemas de parámetros continuos y adecuación continua que exigen del algoritmo celular un mayor número de pasos debido a su baja-media presión selectiva. Esta presión puede modificarse según el problema, disminuyendo así la importancia de esta desventaja. Como contrapartida, la capacidad de exploración, su flexibilidad y las mejoras en eficiencia cuando se distribuye en un anillo de islas celulares los hacen muy atractivos en aplicaciones complejas.

En general, las desventajas de los modelos distribuidos obtenidos de *xx*GA son mínimas, aunque su implementación es bastante más compleja y complicada que la de algoritmos más simples que engloban un único modelo concreto.

Las ventajas de disponer de un modelo distribuido que pueda funcionar como los modelos estudiados son claras al enfrentarlo a un nuevo problema, ya que contamos con numerosas herramientas para resolver la pérdida de diversidad, mínimos locales, problemas de conquista o no efecto, ... Sobre todo, dcGA es quizás el algoritmo canónico que más fácilmente puede adaptarse a una aplicación concreta debido a su maleable presión selectiva, mantenimiento de diversidad y características secuenciales y paralelas tradicionales. Sin embargo, no es posible aconsejarlo globalmente [WM97].

La discusión sobre técnicas de implementación y la visión global proporcionada por el estudio realizado favorecen una mayor completitud de los resultados en comparación con estudios mucho más locales como los orientados a un único operador o técnica concreta. Tanto la implementación como los resultados son susceptibles de numerosas extensiones y combinaciones con estudios posteriores de un nuevo operador, codificación, etc. aplicados a cualquiera de los modelos derivables de *xx*GA.

5.4. Conclusiones

En esta tesis hemos desarrollado un estudio genérico de técnicas y algoritmos genéticos descentralizados y paralelos para después concretar y evaluar un subconjunto especialmente eficiente y práctico de algoritmos. De la clasificación y descripción unificada de algoritmos existentes surgen varias conclusiones que justifican la utilidad de este trabajo. La primera de estas conclusiones es la necesidad de usar un enfoque unificado en la propuesta, diseño, aplicación y análisis de estos algoritmos.

Por esta razón hemos presentado formalmente a los algoritmos secuenciales y paralelos como subclases de un sistema adaptativo granulado, resaltando de esta manera sus similitudes. Esta formalización también facilita la hibridación de los algoritmos evolutivos en general con técnicas dependientes del problema resuelto [CAT98b]. Igualmente, hemos aportado una visión algorítmica paralela que engloba algunas de las características más flexibles e importantes de la ejecución de estos algoritmos. Todo ello ha dado lugar a una propuesta de "plantilla" de algoritmo del que podemos extraer numerosos modelos: *xx*GA.

Hemos analizado los conceptos de diversidad (entropía), teorema de los esquemas y presión selectiva, estudiándolos sobre los distintos modelos. La presión selectiva ajustable del modelo celular resulta muy interesante como mecanismo adicional en la solución de problemas. Además, la influencia de la forma de la rejilla y/o de las tasas de migración utilizadas en los modelos descentralizados permiten modificar esta presión y mejorar la búsqueda.

En general, la presión ejercida en la malla del modelo celular debe ser baja para favorecer la explotación, lo que constituye una desventaja en problemas simples. En cualquier caso, esta dificultad puede aliviarse con el uso de islas celulares en un modelo distribuido, mejorando así la eficiencia.

Tanto los resultados obtenidos como la justificación inicial sugieren usar un algoritmo paralelo distribuido frente a uno secuencial como primera opción, debido tanto a sus prestaciones como a que pueden ejecutarse en plataformas MIMD ampliamente disponibles.

También son interesantes los resultados sobre algoritmos celulares, indicando que este tipo de disposición de la población debe ser tenido en cuenta y estudiado cuando se propongan nuevos algoritmos y operadores, ya que el impacto en ellos puede resultar más beneficioso que en otros modelos más tradicionales como el generacional o incluso el estado estacionario.

5.5. Trabajo Futuro

Como trabajo futuro es interesante el estudio de la influencia de la *heterogeneidad* [AP96] [HL97] [HLM97] sobre los algoritmos paralelos distribuidos. Las ventajas relativas de usar islas celulares y de estado estacionario conjuntamente en el mismo anillo pueden resultar determinantes en el estudio de nuevos problemas.

Igualmente interesante resulta el estudio de *nuevos operadores* en todos los modelos derivables de *xx*GA, incluyendo nuevos operadores de cruce y selección [Cott98]. Además, la ejecución de algoritmos que determinen si resulta útil seguir buscando o no, dando lugar a un sistema distribuido con *nacimientos y muertes de las islas* componentes, es una línea de trabajo interesante que puede dar lugar a algoritmos más eficientes que se adaptan al problema resuelto durante la búsqueda [AT95].

Finalmente, la extensión sobre *nuevos dominios de aplicación* y los resultados de los modelos celulares sobre problemas de función objetivo dinámica (*no-estacionarios*) sería interesante debido a su mantenimiento de la diversidad. Su uso sería una alternativa al uso de selección con distorsión [KH93] [KH97], codificaciones diploides [Gold89a] u otros mecanismos de tratamiento de la adecuación como *sharing*, *crowding* o incluso otros modelos paralelos.

Apéndice **A** Casos de Estudio

En este apéndice se encuentra la descripción y referencias necesarias para comprender los problemas abordados en esta tesis. La selección del conjunto de problemas se ha realizado atendiendo a múltiples consideraciones que detallamos a continuación.

El problema de la *Esfera Generalizada* (Sección A.1) es útil para aportar conclusiones rudimentarias que después se contrastan en otros problemas más complejos.

La función de *Rastrigin Generalizada* (Sección A.2) es típica en evaluación de algoritmos evolutivos y sirve para futuras comparaciones con otros algoritmos; resulta interesante debido a su multimodalidad masiva a medida que el número de variables crece.

En el mismo sentido, la función de *Rosenbrock Generalizada* (Sección A.3) es también típica en la evaluación de estos algoritmos, con la dificultad notable de presentar un nivel de epistasis considerable entre sus variables.

En estas tres funciones, las operaciones típicas de un algoritmo evolutivo se ponen a prueba en la búsqueda sobre espacios-problema extensos, con numerosos óptimos locales, correlación entre las variables involucradas y refinamiento de valores.

El *Problema Decepcionante Masivamente Multimodal* (Sección A.4) y su versión *Fea* (Sección A.5) representan problemas diseñados para resultar difíciles a un algoritmo evolutivo por el tipo de búsqueda que éste realiza. La versión Fea además introduce problemas adicionales para la operación de cruce.

Los problemas de *entrenamiento genético de redes de neuronas* del tipo Perceptrón Multicapa (Sección A.6 y Sección A.7) son actualmente objeto de investigación detallada en varias ramas del campo de los algoritmos evolutivos y de las redes de neuronas, por separado y conjuntamente. Los hemos elegido porque cualquiera de los dos problemas usados muestra una elevada epistasis, multimodalidad, difícil refinamiento de los valores del genotipo y por representar problemas de utilidad práctica.

Finalmente, el problema de la *Mochila* (Sección A.8) y el problema de la *Suma del Subconjunto* (Sección A.9) son problemas NP-Completos que permitirán extender a este campo nuestras aportaciones a la vez que extraemos instancias no-triviales que permitan hacer aflorar las características más interesantes de los algoritmos estudiados.

El conjunto de casos de estudio que utilizamos es especialmente diverso y heterogéneo para así conceder mayor seguridad e impacto a los resultados obtenidos.

A.1. Función Esfera Generalizada (SPH)

Esta función es típica en los estudios sobre algoritmos evolutivos y se suele utilizar como caso base. Es una función continua simple, cuya versión más extendida utiliza tres variables (SPH3). La Ecuación A.1 describe su formulación general. Cuando el problema es de minimización el óptimo es único (de valor 0) -unimodal- y está situado en el origen. Si el problema es de maximización existen 2^n óptimos situados en los extremos del intervalo.

$$f\left(x_i\big|_{i=1..n}\right)=\sum_{i=1}^{n} x_i^2 \qquad x_i \in [-5.12,5.12] \tag{A.1}$$

A.2. Función de Rastrigin Generalizada (RAS)

Esta función (Ecuación A.2) es en realidad una familia parametrizable y escalable de funciones. Presenta un enorme número de óptimos locales (multimodal), requiriendo de una afinación final considerable del valor de las variables debido a que para los valores cercanos al óptimo el término sinusoidal es el dominante. El espacio de búsqueda es bastante extenso. Las instancias más frecuentemente usadas en la literatura consideran dos variables, 10 o 20, generalmente. El óptimo está en el origen (todas las variables a 0), y es de valor 0.

$$Ras(x_i\big|_{i=1..n}) = 10\cdot n + \sum_{i=1}^{n} x_i^2 - 10\cdot\cos(2\cdot\pi x_i) \qquad x_i \in [-5.12,5.12] \tag{A.2}$$

A.3. Función de Rosenbrock Generalizada (ROS)

Esta función (Ecuación A.3) presenta una superficie de búsqueda inclinada (parabólica) con un entorno plano cercano al óptimo (lento progreso final). Su principal característica es la alta epistasis cuando se utilizan instancias de 10 o más variables. Las relaciones entre los valores de las variables determinan el valor final de la función y la hacen especialmente difícil para un algoritmo evolutivo. El óptimo está en x_i=1 para todas las variables, y es de valor 0.

$$Ros(x_i\big|_{i=1..n}) = \sum_{i=1}^{n-1} 100\cdot\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2 \qquad x_i \in [-5.12,5.12] \tag{A.3}$$

A.4. Problema Decepcionante Masivamente Multimodal (MMDP)

Este problema está específicamente diseñado para hacer resultar difícil a un algoritmo evolutivo que use codificación binaria y mutación por inversión de bits [GDH92]. En la Figura A.1 mostramos cómo el problema MMDP consta de *k* subproblemas de 6 bits, de forma que el óptimo vale *k* y está situado en los valores extremos (0 o 6), es decir, cuando cada subproblema consta de 0 o bien de 6 unos. Cada subproblema contribuye a la adecuación de la cadena (Figura A.1) según el número de unos presentes (*unitation*).

DECEPCIÓN BIPOLAR (6 bits)	
#unos	**valor subfunción**
0	1.000000
1	0.000000
2	0.360384
3	0.640576
4	0.360384
5	0.000000
6	1.000000

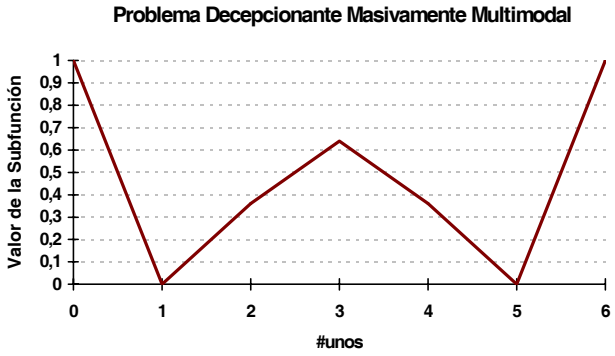


Figura A.1. *Tabla* (izqda.) *y gráfica* (dcha.) *para el problema MMDP.*

Con sólo 5 subproblemas (MMDP5) existen 32 óptimos globales (multimodal) frente a 5.153.644 óptimos locales. De hecho el número de óptimos locales es enorme y crece exponencialmente con el número de subproblemas: 22^k óptimos de los que sólo 2^k son globales. En la literatura podemos encontrar desde instancias de un único subproblema hasta 5 subproblemas [KH96]. En esta tesis resolvemos instancias de hasta 40 subproblemas. Existen problemas similares usando 4 bits y toda una teoría capaz de generarlos.

A.5. Problema Decepcionante Masivamente Multimodal. Versión Fea -*ugly*- (UMMDP)

Esta versión de MMDP incorpora otra importante característica que añade aún más dificultad al problema. Se trata de aumentar la epistasis presente. Básicamente se trata de concatenar 6 subproblemas pero colocando cada bit de cada una de las funciones componentes en posiciones separadas por 6 bits de distancia. En la Figura A.2 las posiciones con igual tono de gris contribuyen a la adecuación del mismo segmento o subproblema.

Figura A.2. *Disposición de los bits de cada subfunción para UMMDP.*

De esta manera para calcular la contribución de la primera subfunción debemos contar cuántos unos existen en las posiciones 0, 6, 12, 18, 24 y 30. Para calcular la contribución de la segunda subfunción debemos contar el número de unos en las posiciones 1, 7, 13, 19, 25 y 31, y así para todas las funciones componentes. La adecuación óptima es 6. Por tanto, este problema es masivamente multimodal, decepcionante y además de alta epistasis.

A.6. Entrenamiento de un Perceptrón Multicapa para el Problema de la Paridad4

Este problema es típico de los estudios sobre algoritmos de entrenamiento de un perceptrón multicapa [MTH89] [WH89] [WS92] [Alba93] [Roma93]. Se trata de encontrar el conjunto de pesos que hace que la red determine la paridad (número de unos) de cuatro bits de entrada. De esta forma, si el número de 1s en la entrada es impar la salida debe ser 1, y si es par la salida debe ser 0.

Para resolverlo utilizaremos una topología en tres capas (entrada-oculta-salida), cada una con 4, 4, y 1 elementos de proceso (EPs) con funciones de transferencia binaria en cada elemento de proceso. Esta arquitectura supone que las cadenas del algoritmo genético contienen $4 \cdot 4 + 4 \cdot 1 + 4 + 1 = 25$ variables (pesos más *bias* -umbrales-). La relación que debe existir entre estos pesos para resolver el problema es desconocida a priori pero muy intensa (elevada epistasis). El sistema resultante es no lineal y multimodal. Asimismo, es normal que aparezcan durante la búsqueda múltiples codificaciones distintas de igual o muy parecida adecuación, de forma que el AG se verá decepcionado por esta característica.

El objetivo es minimizar el error, extendido a todo el conjunto de patrones. En nuestro caso planteamos la función que optimizar como un problema de maximización, restando del máximo error posible el error que realmente comete la red evaluada (véase la Tabla A.1).

TABLA A.1. DATOS SOBRE LA ESTRUCTURA Y ENTRENAMIENTO DE LA RED "PARIDAD4"

Arquitectura	Patrones de Entrada	Patrones de Salida	Función de Adecuación
4 EPs de entrada ⇓ 4 EPs ocultos ⇓ 1 EP de salida	0 0 0 0	0	**ADECUACIÓN** = *MEPV - ERROR* $MEPV = 10 \cdot n \cdot p = 10 \cdot 1 \cdot 16 = 160$ $ERROR = 10 \cdot \sum_{i=1}^{p} \sum_{j=1}^{n} \left| desada_j - actual_j \right|$
	0 0 0 1	1	
	0 0 1 0	1	
	0 0 1 1	0	
	0 1 0 0	1	
	0 1 0 1	0	
	0 1 1 0	0	
	0 1 1 1	1	
	1 0 0 0	1	
	1 0 0 1	0	
	1 0 1 0	0	
	1 0 1 1	1	
	1 1 0 0	0	
	1 1 0 1	1	
	1 1 1 0	1	
	1 1 1 1	0	

En este problema multiplicamos por 10 el error máximo esperado así como la medida del error para facilitar el trabajo del operador de selección (escalado básico de la adecuación).

A.7. Entrenamiento de un Perceptrón Multicapa para la Predicción del Ruido del Tráfico Urbano

Este problema consiste en entrenar una RN que predice el nivel de ruido del tráfico urbano a partir de tres parámetros: el número de vehículos por hora que pasan por una carretera, la altura media de los edificios que bordean la carretera y la anchura de dicha carretera [CCFM93].

Se trata de realizar una traslación no lineal entre las tres entradas y la salida, todas ellas continuas en el rango [0,1]. Para resolverlo utilizamos una arquitectura en tres capas con 3, 30 y 1 elementos de proceso todos ellos con función de transferencia sigmoidal [RM89] [Kung93]. Esto da lugar a 3·30 + 30·1 + 30 + 1 = 151 variables (pesos más *bias* -umbrales-). Además de los problemas mencionados en la sección anterior, es muy acusada la necesidad de refinamiento final del conjunto de pesos. El criterio de parada es alcanzar un error medio menor a 0.006 en el resultado predicho. La función de adecuación es la misma que en el problema anterior, con MEPV=401.8.

TABLA A.2. PATRONES PARA EL ENTRENAMIENTO SUPERVISADO DEL PROBLEMA "TRÁFICO"

PALERMO (<30m)	Número de Vehículos por hora (x10000)	Altura Media de los Edificios (x100) m	Anchura de la Carretera (x100) m	Ruido Medido (x100) dB
1	0.0534	0.0990	0.2860	0.6570
2	0.2172	0.1980	0.2800	0.8240
3	0.1806	0.2450	0.2250	0.7850
4	0.1668	0.2450	0.2250	0.7700
5	0.1392	0.2450	0.2250	0.7520
6	0.1764	0.2300	0.2200	0.7750
7	0.1482	0.2300	0.2200	0.7500
8	0.0384	0.2050	0.2200	0.6370
9	0.1722	0.1485	0.2200	0.7600
10	0.1194	0.1485	0.2200	0.7140
11	0.0606	0.2700	0.2000	0.6900
12	0.0468	0.2700	0.2000	0.6620
13	0.1914	0.1600	0.2000	0.7880
14	0.2364	0.1485	0.2000	0.8250
15	0.1566	0.1485	0.2000	0.7450
16	0.2808	0.2310	0.1960	0.8230
17	0.2022	0.0660	0.1915	0.7810
18	0.1818	0.0660	0.1915	0.7720
19	0.0786	0.2500	0.1900	0.6870
20	0.0588	0.2400	0.1900	0.6770
21	0.0252	0.2150	0.1900	0.6290
22	0.0156	0.2000	0.1770	0.6140
23	0.0096	0.2500	0.1600	0.6090
24	0.1092	0.2450	0.1600	0.7010
25	0.0990	0.2450	0.1600	0.6810
26	0.0822	0.2450	0.1600	0.6750
27	0.1710	0.2000	0.1550	0.7510
28	0.1428	0.1600	0.1510	0.7300
29	0.1266	0.1600	0.1510	0.7230
30	0.1134	0.2150	0.1500	0.7070
31	0.0108	0.2145	0.1500	0.6100
32	0.2610	0.1320	0.1500	0.8040
33	0.2532	0.1980	0.1340	0.7950
34	0.1818	0.1800	0.1170	0.7510
35	0.1674	0.1800	0.1170	0.7440
36	0.2178	0.1320	0.1170	0.7760
37	0.1938	0.1320	0.1170	0.7640
38	0.1488	0.1815	0.1145	0.7360
39	0.1524	0.0825	0.1110	0.7340
40	0.2148	0.2970	0.0722	0.7620
41	0.1608	0.2970	0.0722	0.7300

A.8. El Problema de la Mochila

Dados n objetos con unos beneficios (*profits*) $\vec{p} = \{p_1, p_2, ..., p_n\}$, unos costes o pesos (*weights*) $\vec{\omega} = \{\omega_1, \omega_2, ..., \omega_n\}$ con p_i , $\omega_i \in Z^+$ (enteros positivos) $1 \le i \le n$, y una mochila de capacidad $M \in Z^+$, el problema consiste en encontrar un vector $\vec{x} = \{x_1, x_2, ..., x_n\}$ con $x_i \in \{0, 1\}$ tal que maximice la expresión $\sum_{i=1}^{n} p_i \cdot x_i$ sujeta a la restricción de que $\sum_{i=1}^{n} \omega_i \cdot x_i \le M$.

Los datos para este problema han sido obtenidos siguiendo una distribución uniforme en el rango [5, 50], con la única relación entre ellos de que la suma total de los beneficios es mayor que la suma total de los costes.

Capacidad de la mochila ..... :	425,0 (88,73% del coste total)
Número de objetos ..... :	20,0
Beneficio total ..... :	507,0
Coste total ..... :	479,0
Beneficio medio ..... :	25,4
Coste medio ..... :	24,0
Beneficio/coste medio..... :	1,2
Solución óptima ..... :	482,0

Vector de beneficios ..... :

30 15 25 20 18 32 10 20 20 40 5 12 17 31 32 25 40 47 43 25

Vector de costes ..... :

20 10 30 17 14 25 45 13 15 35 7 8 14 28 32 19 42 50 37 18

Se trata de un problema NP-Completo que únicamente utilizamos para evaluar la velocidad relativa de ejecución entre versiones de la población que utilicen apuntadores o un vector estático (Capítulo 3). Las restricciones se manejan penalizando a las cadenas no admisibles [Dora97].

A.9. El Problema de la Suma del Subconjunto (SSS)

Se trata de un problema NP-Completo consistente en encontrar, de entre un conjunto de valores $W=\{\omega_1, \omega_2, ..., \omega_n\}$, el subconjunto $V \subseteq W$ de valores que sumados se acercan más, sin superar, a un cierto valor entero C. Usamos instancias de tamaño n=128 generadas según [KBH93]. El rango de los valores es $[0,10^4]$ (y no $[0,10^3]$), ya que esto incrementa la varianza, aumentando así la dificultad de las instancias resultantes [Jela97].

Puesto que todos nuestros problemas están expresados como maximización de una función objetivo, en este caso calculamos el peso de una solución $\vec{x} = \{x_1, x_2, ..., x_n\}$ como $P(\vec{x}) = \sum_{i=1}^{n} x_i \cdot \omega_i$ y el algoritmo usado debe maximizar la función

$$f(\vec{x}) = a \cdot \left(P(\vec{x})\right) + \left(1 - a\right) \cdot \left\lfloor \left(C - 0{,}1 \cdot P(\vec{x})\right) \right\rfloor_0 \qquad \text{(A.4)}$$

donde a=1 cuando $\vec{x}$ es admisible, es decir, cuando $C - P(\vec{x}) \ge 0$, y a=0 en otro caso. Las soluciones no admisibles se penalizan con el 10% de su valor (el resultado es siempre ≥0).

Apéndice **B**
Ficheros de Configuración y UDPs

B.1. Fichero de Configuración del Sistema

<*SistConfigFile*> ::= <*WSISection*> [<*Comment*>]
<*NIslands*> [<*Comment*>]
<*ICFSection*> [<*Comment*>]
<*SyncMode*> [<*Comment*>]
{<*OptionalSections*> [<*Comment*>]}

<*OptionalSections*> ::= ε |
<*MigGap*> | <*NMigrants*> | <*Cycles*> | <*ProbConfigFile*>
<*Seeds*> | <*MigSelection*> | <*MigReplac*> | <*CheckPoints*>

<*WSISection*> ::= **[worstation_info]**
$\{<WSLine>\}_1$

<*WSLine*> ::= <*WSName*> <*Integer*> $\{<gaid>\}_1^{<Integer>}$

<*Identifier*> ::= $\{[\mathbf{a} \mid \mathbf{b} \mid ... \mid \mathbf{z} \mid \mathbf{0} \mid ... \mid \mathbf{9}]\}_1$

<*WSName*> ::= <*Identifier*>

<*Cardinal*> ::= $\{[\mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}]\}_1$

< *Cardinal* *> ::= $[\mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}]\{[\mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \mathbf{4} \mid \mathbf{5} \mid \mathbf{6} \mid \mathbf{7} \mid \mathbf{8} \mid \mathbf{9}]\}_1$

<*gaid*> ::= **ga**< *Cardinal**>

<*ICFSection*> ::= **[island_config_files]**
[**0** <*IslandConfigFile*>| <*IConfigFileList*>]

<*IConfigFileList*> ::= <*NGAs*>
$\{<gaid> <IslandConfigFile>\}_1^{<NGAs>}$

<*IslandConfigFile*> ::= <*Identifier*>

<*NGAs*> ::= < *Cardinal**>

<*NIslands*> ::= **[number_of_islands]**
<*NGAs*>

<*SyncMode*> ::= **[synchronization]**
[**sync_mode** | **async_mode**]

<*MigGap*> ::= **[migration_gap]**
<*Cardinal*> *// default 10 evals*

<*NMigrants*> ::= **[number_of_migrants]**
<*Cardinal*> *// default 1*

<*Cycles*> ::= **[cycles]**
<*Cardinal**> *// default 10 evals*

<*ProbConfigFile*> ::= **[problem_config_file]**
<*Identifier*> *// default empty*

<*Seeds*> ::= **[seeds]**
[**0** | <*SeedList*>] *// default random seeds*

<*SeedList*> ::= $\{<Cardinal>\}_1^{<NGAs>}$

<*MigSelection*> ::= **[mig_policy_for_selection]**
[**mig_random** | **mig_best**] *// default mig_random*

<*MigReplac*> ::= **[mig_policy_for_replacement]**
[**mig_least_fit** | **mig_if_better**] *// default mig_if_better*

<*CheckPoints*> ::= **[check_points]**
$\{<Checkp>\ <Cardinal>\}_1^{12}$

<*Ceckp*> ::= **ga_stats** | **sch_stats** | **proc_stats** | **min** | **max** | **avg** | **evals** |
eval_mam | **hammingd** | **best** | **worst** | **solution**

<*Comment*> ::= **//** ~~**b**~~ $\{<Identifier> | ~~\mathbf{b}~~ | <Symbol>\}_0$

<*Symbol*> ::= **!** | **“** | **·** | **$** | **%** | **&** | **/** | **(** | **(** | **)** | **=** | **?** | **¿** | **¡** | ***** | **-** | **<** | **>** | **,** | **;** | **.** | **:** | **_** | **[**
| **]** | **{** | **}** | **°** | **#** | **ª** | **¬** | **+** | **Ç** | **ñ** | **Ñ** | **á** | **é** | **í** | **ó** | **ú** | **Á** | **É** | **Í** | **Ó** | **Ú**

B.2. Fichero de Configuración de cada Isla AG

<*IslandConfigFile*> ::= <*NOpsSection*> [<*Comment*>]
<*OpsSection*> [<*Comment*>]
{<*OptionalSections*> [<*Comment*>]}

<*OptionalSections*> ::= ε |
<*ScalingSection*> | <*OutputSection*> | <*NeighbSection*>
<*NRangeSection*> | <*SeedSection*> | <*ProblemSection*>

<*NOpsSection*> ::= **[number_of_operators]**
<*NOperators*>

<*NOperators*> ::= < *Cardinal**>

<*OpsSection*> ::= **[operators]**
<*Operator*> { <*Operator*> }$_1^{<NOperators>}$

<*Operator*> ::= **rw** | **sus** | **random** | **best** | **ranking** <*Bias*> |
spx <*Prob*> | **dpx** <*Prob*> |
ux <*Prob*> <*Bias*> | **ax** <*Prob*> <*Bias*> |
mutation <*Prob*> |
<*BP*> | <*SA*> | <*GeneralOp*> |
rep_rand | **rep_least_fit** | **rep_worst_rand_pool** <*Cardinal**>

<*Bias*> ::= <*Float01*>

<*Prob*> ::= <*Float01*>

<*Float01*> ::= **1.0** | **0.**{<*Cardinal*>}

<*BP*> ::= **bp** <*Prob*> <*Cardinal*> <*Float01*> <*Float01*> [**best** | **actual**]
// probability epochs learning_rate momentun which

<*SA*> ::= **sa** <*Prob*> <*Float01*> <*Float01*> <*Function*> [**best** | **actual**]
<*Cardinal*> <*Cardinal*>
// probability Ti Tf Function which steps #changes

<*Function*> ::= <*Identifier*> *// must exist at compiletime*

<*GeneralOp*> ::= **operator** <*Prob*> <*Function*>

<*SeedSection*> ::= **[seed]** <*Cardinal*> *// problem random seed*

<*ScalingSection*> ::= <*LinearScaling*> | <*Sigma_Scaling*>

<*LinearScaling*> ::= **[linear_scaling]** <*Float01*> *// delta factor*

<*Sigma_Scaling*> ::= **[sigma_scaling]** <*Float01*> *// sigma factor*
<*OutputSection*> ::= **[output_display]**
[**nothing** | **all** | **on_change** | **steps** <*Cardinal**>] *// default "nothing"*
<*NeighbSection*> ::= <*NeighShape*> <*NeighbTopology*>
<*NeighShape*> ::= **[neighb_shape]** <*Cardinal*> <*Cardinal*> <*Cardinal*> *// x y z*
<*NeighbTopology*> ::= **[neighb_topology]**
[**grid** | **unid_ring** | **bid_ring** | **any**] *// default "any"*
<*NRangeSection*> ::= **[numeric_ranges]**
<*Float*> <*Float*>
<*Float*> ::= [**+** | **-**]<*Cardinal*>**.**<*Cardinal*>
<*ProblemSection*> ::= **[problem_config_file]**
<*Identifier*> *// default empty*

B.3. Unidades de Datos del Protocolo de Comunicación (UDPs)

FORMATO GENERAL DE UDP

<Orden>	<LongitudMsg>	<MENSAJE>
4 [char]	*1 [unsigned long]*	*MAX_MESSAGE_LENGTH*

<origen>→<destino> *<acción tomada en el destino>*

UDP DE ENVÍO/RECEPCIÓN DE DATOS DE CONFIGURACIÓN INICIALES

CONF	L	Mig_Params	Evaluacs.	Info Isla	ETP del Vecino
4 [char]	*1 [ulong]*	*estructura*	*1 [ulong]*	*estructura*	*1 entrada en TP*

monitor→isla *actualiza datos de configuración*

UDP DE NOTIFICACIÓN DE TERMINACIÓN Y UDP DE PETICIÓN DE TERMINACIÓN

ENDP	L	Condición de Terminación
4 [char]	*1 [unsigned long]*	*1 [bool] – TARGET{GENS|SOLU}*

isla→monitor *inicia secuencia de salida*

ENDP	0
4 [char]	*1 [unsigned long]*

monitor→isla *pide iniciar protocolo de salida*

UDP DE PETICIÓN Y UDP DE ENVÍO DE ESTADÍSTICAS SOBRE LA POBLACIÓN

GAST	0
4 [char]	*1 [unsigned long]*

monitor→isla *devuelve estadísticas sobre población (no actualiza)*

GAST	L	Estadísticas de la Población (POP_STATS)
4 [char]	*1 [unsigned long]*	*tamaño de la estructura de datos POP_STATS*

isla→monitor *actualiza sus estadísticas parciales*

UDP DE PETICIÓN DE UN INDIVIDUO CONCRETO

GETI	L	Posición Individuo
4 [char]	*1 [unsigned long]*	*1 [int]*

monitor→isla *devuelve una copia de dicho individuo*

UDP DE PETICIÓN Y UDP DE ENVÍO DE GENERACIÓN Y EVALUACIONES ACTUALES

GEVA	0
4 [char]	*1 [unsigned long]*

monitor→isla devuelve generación actual y número evaluaciones hechas

GEVA	L	Generación Actual	#Evaluaciones Hechas
4 [char]	*1 [unsigned long]*	*1 [unsigned long]*	*1 [unsigned long]*

isla→monitor actualiza sus estadísticas parciales

UDP DE ENVÍO DE UN INDIVIDUO Y SU ADECUACIÓN

INDI	L	Individuo Expresado	Adecuación
4 [char]	*1 [unsigned long]*	*#Parámetros [double]*	*1 [double]*

isla→monitor actualiza estadísticas parciales

isla→isla inserta en población de forma adecuada

UDP DE INFORMACIÓN DE CONEXIÓN CON EL MONITOR

INIG	L	Descriptor de Isla	PAS Transf. Datos
4 [char]	*1 [unsigned long]*	*1 [int]*	*1 [int]*

isla→monitor crea una entrada válida en la TP para dicha isla

UDP DE TERMINACIÓN DE ISLA

KILL	0
4 [char]	*1 [unsigned long]*

monitor→isla termina ejecución totalmente

UDP PARA LEER DESDE FICHERO UNA COPIA DE LA POBLACIÓN DE UNA ISLA

LOAD	0
4 [char]	*1 [unsigned long]*

monitor→isla lee desde fichero predefinido la población inicial

UDP DE PETICIÓN Y UDP DE ENVÍO DE ESTADÍSTICAS SOBRE CONSUMO DE RECURSOS

PSTA	0
4 [char]	*1 [unsigned long]*

monitor→isla *actualiza las estadísticas de proceso y las envía*

PSTA	L	Estadísticas del Proceso (Proc_Stats)
4 [char]	*1 [unsigned long]*	*tamaño de la estructura de las instancias de clase Proc_Stats*

isla→monitor *actualiza sus estadísticas parciales*

UDP PARA GUARDAR EN FICHERO UNA COPIA DE LA POBLACIÓN DE UNA ISLA

SAVE	0
4 [char]	*1 [unsigned long]*

monitor→isla *guarda en un fichero predefinido la población inicial*

UDP DE PETICIÓN Y UDP DE ENVÍO DE ESTADÍSTICAS SOBRE EL ESQUEMA EN ESTUDIO

SCHS	0
4 [char]	*1 [unsigned long]*

monitor→isla *actualiza las estadísticas sobre el esquema y las envía*

SCHS	L	Estadísticas del Esquema (SCH_STATS)
4 [char]	*1 [unsigned long]*	*tamaño de la estructura SCH_STATS*

isla→monitor *actualiza sus estadísticas sobre el esquema estudiado*

UDP PARA PETICIÓN DE LA SOLUCIÓN PARCIAL ENCONTRADA HASTA EL MOMENTO

SOLU	0
4 [char]	*1 [unsigned long]*

monitor→isla *devuelve el mejor individuo encontrado hasta el momento*

UDP PARA ARRANCAR LA BÚSQUEDA EN UNA ISLA

STRT	0
4 [char]	*1 [unsigned long]*

monitor→isla *inicia la ejecución del algoritmo en la isla*

Apéndice C
Glosario de Términos

Este apéndice contiene una lista ordenada alfabéticamente de los términos y acrónimos más usados en esta memoria, con objeto de facilitar su lectura y posteriores accesos aislados a su contenido. No se trata de un listado exhaustivo ni de definiciones rigurosas, sino de una explicación básica de conceptos para poder seguir el desarrollo de los capítulos.

Adecuación (*fitness*)	Medida de la bondad de una estructura considerada como solución a un problema. Cada individuo en un AG tiene asociado un valor de adecuación que indica la bondad de la solución que representa.
AEP (PEA)	Algoritmo Evolutivo Paralelo (*Parallel Evolutionary Algorithm*).
Alelo	Valor que puede tomar un gen, procedente de una posición de la estructura progenitora, que rige un carácter hereditario. Valor elemental de una de las posiciones de un individuo.
Algoritmo de búsqueda	Proceso para localizar una solución particular para un problema dado dentro de un conjunto de soluciones plausibles**.**
Algoritmo evolutivo	Algoritmo de búsqueda que mantiene una población de estructuras que evolucionan de acuerdo a reglas estocásticas como la selección natural (supervivencia de los más aptos), el emparejamiento (cruce) y la mutación.
Algoritmo genético (AG, GA)	Algoritmo evolutivo que mantiene una población de soluciones candidatas a una función objetivo dada. Está basado en los mecanismos de la Genética y en la selección natural de las especies y trabaja sobre cadenas de parámetros.
Algoritmo genético de grano fino (*fine-grained* o modelo con vecindad)	AG en el que la población total se divide en una gran número de subpoblaciones pequeñas sobre las que se define un esquema de vecindad. El caso extremo es tener un único individuo en cada subpoblación definida (acoplamiento fuerte).
Algoritmo genético de grano grueso (*coarse-grained* o modelo isla)	Algoritmo genético en el que la población se divide en varias subpoblaciones las cuales se mantienen relativamente aisladas unas de otras, conectadas únicamente por intercambios esporádicos de estructuras.
Algoritmo genético híbrido	Utilización conjunta de un algoritmo genético y otra técnica no genética como *backpropagation* o enfriamiento simulado. Estas técnicas se utilizan como un operador genético más, manteniendo una relación de competición, o incluyendo conocimiento sobre el problema que se está resolviendo.
Algoritmo genético paralelo (AGP o PGA)	Algoritmo genético donde el paralelismo reside en que la ejecución de algunas operaciones se solapa en el tiempo. Existen diversos modelos de paralelismo: grano grueso, grano fino, paralelismo global, mixto, ...
Algoritmos de caja negra (*black-box algorithms*)	Algoritmos que intentan optimizar una función utilizando una estrategia independiente del problema. Los algoritmos genéticos (AGs) pertenecen a esta clase de algoritmos.
Aparición de especies (s*peciation*)	Término que hace referencia a diferentes soluciones a un problema que coexisten y se generan progresivamente en paralelo.

Aprendizaje	Proceso de búsqueda de los pesos que codifican el conocimiento que deseamos imbuir en una red de neuronas.
Aprendizaje no supervisado (*clustering*)	Aprendizaje de una red de neuronas (RN) a la que sólo se le proporcionan los patrones de entrada. Las características de las entradas se descubren estadísticamente con el objetivo de generar salidas correctas.
Aprendizaje supervisado	Tipo de aprendizaje en el que la red de neuronas se entrena suministrándole los patrones de entrada y sus correspondientes patrones de salida (salida deseada).
AX	Cruce aritmético. Para cada alelo real del hijo se toma un porcentaje predefinido de cada alelo de un padre y se añade al porcentaje complementario del otro padre.
***Backpropagation* (propagación hacia atrás del error -PEA-)**	Algoritmo utilizado para entrenar una red de neuronas basado en el seguimiento descendente del gradiente en la superficie de error que la RN comete. El método está inspirado en los mínimos cuadrados -LMS- (regla delta) y encuentra los valores de todos los pesos que minimizan la función de error definida. Este método tiene el problema de caer frecuentemente en mínimos locales. La superficie de error de una red compleja está llena de colinas y valles. Debido al gradiente descendente, la RN puede verse atrapada en un mínimo local cuando hay un mínimo lo suficientemente profundo cerca. Algunos métodos probabilísticos pueden disminuir esta trampa. Otra posibilidad es incrementar el número de capas ocultas.
***Backtracking* (Retroceso)**	Método algorítmico de programación aplicable a aquellos problemas que se pueden resolver mediante una búsqueda exhaustiva con posible retroceso en el árbol de posibles soluciones, las cuales pueden ser representadas como una *n*-tupla de valores.
Bias	Es un peso más presente en todos los elementos de proceso de una RN que está conectado a un nodo ficticio cuya salida siempre está activa (1) y por tanto participa siempre en el proceso de aprendizaje funcionando como valor umbral.
Bloque básico (*Building Block*) o BB	Esquema corto, de bajo orden y alto valor medio de adecuación. Un algoritmo genético implícitamente repite y recombina BBs con la esperanza de obtener BBs cada vez mejores.
Búsqueda aleatoria	Búsqueda ciega que consiste en visitar puntos-solución seleccionados al azar.
C++	Lenguaje de programación orientado a objetos. Fue desarrollado a partir del lenguaje de programación C. Fue implementado para mejorar el lenguaje C, para apoyar la abstracción de datos y la programación orientada a objetos.
Capa	Forma en que se organizan los pesos de una red de neuronas. Genéricamente en una RN se distingue entre capa de entrada, capas ocultas y capa de salida.
Capa de entrada	Conjunto de neuronas de una RN que actúan como interfaz pasiva para recibir los datos de entrada desde el entorno.
Capa de salida	Conjunto de neuronas de una RN que actúan como interfaz activa para descargar o presentar los datos calculados por la RN al entorno.
Capa oculta	Capa cuyas entradas y salidas permanecen dentro del sistema en una RN.
Codificación (*encoding*)	En relación a los algoritmos genéticos nos referimos a las reglas que determinan la forma en que las cadenas de un AG representan las soluciones a un problema dado. En relación con el diseño de redes de neuronas mediante algoritmos genéticos nos referimos a la forma en que las cadenas contienen los pesos de la RN.
Codificación de entrada (*input encoding*)	Uno de los tipos de genotipo para el diseño de redes neuronales mediante algoritmos genéticos. Se basa en codificar juntos los pesos de entrada de cada neurona desde la capa oculta a la capa de salida En esta codificación la influencia de una neurona está en parte localizada en la cadena y en parte dispersa, pero la estructura en capas de la red de neuronas está directamente presente en las cadenas.

Codificación de entrada-salida (*input-output encoding*) Clase de genotipo para el diseño de redes de neuronas mediante algoritmos genéticos. La codificación de entrada-salida para una neurona dada codifica sus pesos de entrada seguidos por sus pesos de salida (realmente los pesos de entrada a la próxima capa). De esta forma, para dos neuronas adyacentes de una capa, *i* y *j*, colocamos juntos los pesos de entrada y salida de *i* y después los pesos de entrada y salida de *j*. En esta codificación la influencia de una neurona dada está completamente localizada en la cadena, pero no así el diseño en capas.

Combinaciones de apoyo (*supportive combinations*) Algoritmo híbrido (conjunto de algoritmos) que buscan solucionar el mismo problema. En ellas un algoritmo juega el papel de resolutor primario del problema mientras que otros algoritmos juegan el papel de apoyo.

Combinaciones de colaboración Conjunto híbrido de algoritmos en el que dos o más técnicas colaboran juntas para resolver un problema dado sin que el sistema global muestre una jerarquía entre ellas, es decir, colaboran en la búsqueda al mismo nivel.

Comodín (*wild card*) Posición del esquema en un algoritmo genético que representa al conjunto de alelos que podrían aparecer en una instancia concreta.

Conexión (peso) Cantidad numérica real (flotante) asociada al enlace que une dos nodos de una red de neuronas. Se corresponde con la intensidad de las conexiones sinápticas entre neuronas reales. Determinan lo importante que es una conexión.

Convenciones sobre competencia (*competing conventions*) Problema que surge al aplicar algoritmos genéticos para el diseño de redes de neuronas. Antes de evaluar un genotipo, primero se decodifica produciendo el fenotipo. Si esta aplicación es de muchos a uno, entonces diferentes genotipos se decodifican en el mismo fenotipo, o fenotipos equivalentes, aunque sus genotipos sean bastante distintos. Por ejemplo, la permutación de los nodos ocultos de algunas redes *feedforward* no altera su función con lo que la red exhibe el mismo *fitness* ante las mismas entradas. El algoritmo genético desconoce esta arbitrariedad de la representación y puntualmente puede verse beneficiado o perjudicado por ella.

Convergencia prematura Problema que presentan ciertos métodos de búsqueda consistente en que el algoritmo tiende hacia una estructura única que no es la óptima. En un algoritmo genético se relaciona también con la deriva genética.

Cromosoma Cada uno de los orgánulos celulares que contienen el material de la herencia biológica. Se considera una concatenación de genes. También conocido como estructura, cadena o individuo en un algoritmo evolutivo (caso haploide).

Cruce (*crossover*) Operador estocástico genético que modifica la población de individuos mediante la producción de nuevos individuos resultantes de porciones elegidas de dos padres. Produce uno o más descendientes (típicamente dos) a partir de dos padres, previamente seleccionados por algún método. Formalmente produce un intercambio de información entre los hiperplanos presentes en el genotipo dando lugar a nuevos puntos de búsqueda en la población de forma acotada con o sin efectos sobre la diversidad presente (típicamente no afecta a la diversidad).

CX (*cycle crossover*) Operador genético de cruce que construye dos descendientes de forma que, para cada uno, cada valor y su posición proviene de uno de sus progenitores. Respeta la posición absoluta de los elementos en las secuencias padre. Por ejemplo, de (1 2 3 4 5 6 7 8 9) y (4 1 2 8 7 6 9 3 5) produciría los descendientes (1 2 3 4 7 6 9 8 5) y (4 1 2 8 5 6 7 3 9).

Deriva genética Efecto de un AG por el que, aún cuando todas las estructuras de la población tienen el mismo valor de adecuación el algoritmo toma preferencia por una de ellas, a pesar de que no tiene elementos para definirse por una u otra estructura.

Descendiente Individuo resultante de una operación genética aplicada a un conjunto de progenitores.

dGAME	Entorno abierto de manipulación de algoritmos genéticos distribuidos. Fue propuesto y diseñado en LCC-UMA. Este entorno se basa en la máquina virtual *GAME* para la manipulación de información genética, en topologías de interconexión de algoritmos genéticos y en el modelo de ejecución distribuido y masivamente paralelo simulado en un entorno real multiprocesador distribuido.
Diversidad	Variedad en el contenido de los individuos de una misma población genética. Tradicionalmente se mide como distancia de Hamming entre las estructuras de una población o como la frecuencia por cada *locus* de cada alelo posible. Tmabién puded medirse calculando la entropía media de la población.
Dominio (espacio de búsqueda)	Conjunto de valores sobre el que está definida la función que mide la adecuación.
DPX (*double point crossover*)	Operador genético de doble punto de cruce. Se seleccionan dos puntos aleatoriamente en las dos cadenas padres y se intercambia el contenido entre ellos, produciendo dos cadenas descendientes.
Elemento de proceso **EP** **(nodo o neurona)**	Unidad básica de trabajo de que se compone una red de neuronas. Genera una señal de salida a partir de otra de entrada. Para ello calcula a partir de la suma de sus entradas pesadas un valor que somete a una función de transferencia (FdT) cuya salida a su vez es el argumento de la función de activación (FdA) de dicho elemento de procesamiento: *Salida=FdA(FdT(SumaEntradas))*.
Elitismo	Variante de búsqueda evolutiva en la que un porcentaje de la población actual sobrevive de forma determinista al bucle reproductor.
Enfriamiento simulado **ES** **(*simulated annealing*)**	Método de optimización combinatoria cuyo objetivo consiste en encontrar la mejor solución de entre un número finito de posibles soluciones a un problema. Para ello, un conjunto de reglas permiten modificar una configuración o estructura actual, que representa una solución parcial, para proporcionar otra estructura con un valor de menor coste esperado. Las estructuras de mayor coste no siempre se rechazan, sino que se aceptan con cierta probabilidad decreciente con el funcionamiento del algoritmo (el parámetro temperatura gobierna este proceso estocásticamente).
Entrenamiento de una red de neuronas	Proceso consistente en encontrar un conjunto de pesos para una RN que la haga comportarse como se desea. Los algoritmos genéticos pueden aplicarse como técnica para este entrenamiento.
Enumeración exhaustiva	Método de búsqueda que consiste en la evaluación de todos los puntos del dominio solución hasta encontrar un óptimo global. Se trata de un mecanismo reconocidamente seguro aunque de eficiencia muy baja en problemas reales.
Epistasis	Grado de ligazón entre los componentes de una cadena de valores.
Época (*epoch*)	Número de veces que se aplica el algoritmo de *backpropagation* (regla delta generalizada) sobre el conjunto de patrones de entrenamiento en una RN.
Escalada (*hill climbing*)	Técnica iterativa que explota la mejor solución encontrada hasta el momento e intenta mejorarla, reemplazando el punto actual por un vecino de mejor adecuación; por otro lado descuida la exploración del espacio de búsqueda.

Escalada estocástica (*stochastic hillclimbing*)
Variante probabilística de la escalada. Consiste en la búsqueda en un espacio discreto S con la intención de encontrar un estado cuyo *fitness* sea tan alto (o tan bajo) como sea posible. El algoritmo realiza esto haciendo sucesivas mejoras sobre algún estado actual $\sigma \in S$. Las mejoras locales efectuadas por el algoritmo están determinadas por la *estructura vecinal* y la función de adecuación impuesta sobre S. La estructura vecinal se puede considerar como un grafo no dirigido G sobre el conjunto de vértices S. El algoritmo intenta mejorar su estado actual σ haciendo una transición a uno de los vecinos de σ en G. En particular, el algoritmo elige un estado τ de acuerdo a una distribución de probabilidad sobre los vecinos de σ. Si el *fitness* de τ es al menos tan bueno como el de σ entonces τ se convierte en el nuevo estado actual; de otro modo se conserva σ. Este proceso se repite. El algoritmo esencialmente realiza una búsqueda aleatoria (*random walk*) en la cual los movimientos hacia un valor de adecuación menor son siempre rechazados.

Escalado
Método que se utiliza para facilitar y mejorar el trabajo del operador de selección. Esto se consigue modificando los valores de adecuación de todos los individuos de la población para hacerlos muy distintos entre sí.

Escalado lineal
Método de escalado que calcula dos valores de una ecuación lineal a y b para modificar la adecuación en la forma $f'=a \cdot f+b$.

Escalado por truncado sigma
Método de escalado que tiene la ventaja de evitar valores negativos una vez se ha modificado la adecuación. Fue ideado por Forrest y consiste en usar la desviación típica (sigma) para generar un nuevo valor: $f'=máx[0,f-(f_m-c \cdot \sigma)]$, donde c es una constante (entre 1 y 3), f_m es la adecuación media y σ es la desviación típica.

Escalado potencial (*power law scaling*)
Método de escalado en que se acentúa las diferencias entre las adecuaciones de la población elevando a cierto exponente cada uno de ellos: $f'=f^k$ (por ejemplo k=1.005).

Espaciado (*spatiality*)
Propiedad de la población en un AG por la que se asigna cada estructura a una posición específica en un entorno espacial dado, normalmente una rejilla bidimensional implementada como una matriz toroidal. En muchas ocasiones la eficacia, rango de aplicación, explotación de recursos computacionales disponibles y la eficiencia, tanto numérica como en tiempo real son mayores que en poblaciones sin estructura alguna (panmixia).

Esquema
Cadena cuyos elementos pertenecen tradicionalmente al alfabeto {0, 1, *}, que se utiliza para aludir a conjuntos de cadenas en un algoritmo genético. Simboliza la similitud existente entre las cadenas-instancias por él representadas y permite derivar una teoría para explicar el funcionamiento de los algoritmos genéticos.

Esquema evolutivo
Tipo de bucle reproductor que emplea un AG. Se caracteriza por el tipo de paso básico de avance en la evolución. Se distinguen dos tipos de esquema evolutivos: por generaciones y por efectos inmediatos (*steady-state*). Un tercer esquema evolutivo de compromiso (ajustable) define el porcentaje (*gap*) de individuos que sufren reemplazo.

Esquema evolutivo ajustable
Bucle reproductor en el que sólo participa un cierto porcentaje de la población total de que dispone el AG.

Esquema evolutivo de estado estacionario (*steady-state* o por efectos inmediatos)
Esquema evolutivo por el cual el plan reproductor obtiene uno o dos descendientes que son insertados en la población: AE-(μ+1). En este caso el paso básico de avance es la reproducción. Padres e hijos coexisten en la misma población. La complejidad de cada paso es menor que en el caso generacional.

Esquema evolutivo por generaciones	La población, llamada progenitora, se somete a tratamiento genético (selección, cruce, mutación, etc...) añadiendo los individuos así obtenidos a una población distinta, llamada descendiente: AE-(μ,λ). Una vez que se completa la población descendiente se sustituye la población progenitora por la población descendiente recién obtenida. El paso básico de avance de la evolución es la generación.
Evolución	Campo de la Biología que tiene por objeto el estudio de los cambios que han sufrido los seres vivos a lo largo del tiempo y que han dado lugar a su diversificación a partir de antepasados comunes.
Explotación avariciosa (*greedly exploit*)	Técnica heurística de búsqueda que consiste en escoger la mejor solución de una pequeña muestra inicial. Su desventaja es que no sigue buscando una solución mejor pues seguramente existen soluciones mejores en el dominio de soluciones que las elegidas de forma casual en la pequeña muestra inicial.
Expresión (genotipo → fenotipo)	Proceso que gobierna la traslación del genotipo en fenotipo (decodificación binaria-a-decimal, por ejemplo).
Fenotipo	Aspecto observable que ha sido adquirido como consecuencia del genotipo que posee y de la acción del medio ambiente (función de adecuación).
Fitness	Adecuación.
Función de error	Medida de la diferencia entre el resultado deseado y el obtenido.
Función de evaluación	Operación que permite conocer el valor de adecuación asociado a cada fenotipo.
Función de salida	Determina, en una red de neuronas, el nivel de activación basado en la entrada neta de la neurona y en su estado. Puede ser de diversa naturaleza: función umbral, sigmoidal, binaria, etc...
Función objetivo	Función original que se pretende optimizar con un algoritmo genético.
GA	Algoritmo genético (siglas inglesas).
GAME	Entorno abierto de manipulación de algoritmos genéticos. Fue propuesto y diseñado por L. Deckker y J. L. Ribeiro Filho. Tiene como objetivo la resolución de problemas mediante técnicas basadas en algoritmos genéticos. El usuario de la máquina virtual se abstrae de todo lo relacionado con el manejo de información genética a bajo nivel y así se facilita la implementación de técnicas paralelas para resolución de problemas con coste mínimo de diseño.
Gap	Fracción de la población que se reemplaza en cada generación.
Gen	Partícula elemental de un cromosoma, responsable de la transmisión hereditaria de un carácter. El conjunto de genes es el genotipo. En una técnica evolutiva representa uno de los parámetros del problema real que se desea resolver.
Generación	Paso evolutivo de toda una población en el que un conjunto de descendientes sustituye completamente a sus padres.
GENESIS	**S**istema de **I**mplementación de **B**úsqueda **Gené**tica para optimización de funciones. Se trata de una biblioteca de recursos de programación escritos en C que permite ser utilizada para cualquier problema de maximización o minimización de funciones que pueda ser abordado mediante el uso de algoritmos genéticos.
Genética	Parte de la Biología que se ocupa del estudio de la herencia biológica e intenta explicar los mecanismos y circunstancias que rigen la transmisión y variabilidad de caracteres de generación en generación.
Genotipo	Conjunto de caracteres o factores hereditarios (genes) que posee un individuo por haberlos recibido de sus progenitores (parámetros del problema codificados).
Grano grueso	Tipo de algoritmo distribuido que se caracteriza porque la proporción entre el tiempo consumido en la computación y el consumido en comunicación es alta.

Herencia	Proceso de transmisión de un conjunto de caracteres congénitos de los progenitores de una determinada especie a sus descendientes.
Herencia dominante	Tipo de herencia en la que, de los dos genes que rigen un mismo carácter, sólo uno de ellos, llamado *dominante*, se manifiesta en el fenotipo mientras que le otro, llamado *recesivo*, queda "oculto".
Herencia intermedia	Tipo de herencia en la que los dos genes que rigen un mismo carácter tienen "fuerzas similares" para manifestarse en el fenotipo, por lo que el carácter que rigen se manifiesta como una mezcla de ambos.
HX (*heuristic crossover*)	Operador de cruce utilizado en algoritmos genéticos que se aplican a problemas que satisfacen unas determinadas restricciones. Admite muchas implementaciones y un número variable de estructuras origen. Altera algunas posiciones y calcula cómo dar la estructura hijo de forma heurística.
Individuo	Representación codificada de una solución a un problema dado. Típicamente es haploide aunque puede ser diploide o *n*-ploide en general. Igualmente es típico que sea una cadena de símbolos aunque pueden encontrarse otras estructuras como árboles sintácticos o mezclas de representaciones distintas en el mismo individuo.
Ingeniería del *software*	Aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y la documentación asociada necesaria para su uso, desarrollo, operación y mantenimiento.
Intervalo de migración (*migration interval*)	Valor numérico entero que indica cada cuántos pasos tiene lugar una migración entre las islas de un AG de grano grueso distribuido.
Las Vegas, Algoritmo	Algoritmo de búsqueda inspirados en técnicas de *Backtracking* pero que asumen el riesgo de tomar una decisión aleatoria inicial que les puede impedir encontrar una solución. Reaccionan, o bien devolviendo una solución correcta, o bien admitiendo que sus decisiones aleatorias iniciales les han conducido a un punto muerto, en cuyo caso es suficiente con volver a probar sobre el mismo problema en el mismo algoritmo para tener una segunda oportunidad independiente de llegar a una solución. La probabilidad total de éxito se incrementa con la cantidad de tiempo disponible.
LD (LB o FL)	Lógica Difusa (Lógica Borrosa o *Fuzzy Logic*).
Locus	Punto determinado y específico del cromosoma donde está localizado cada alelo.
Longitud de definición de un esquema $\delta(H)$	Distancia entre la primera y la última posición fija (distinta del comodín "*") del esquema *H*.
MCDD	*Multiple Criteria Decision Making*.
Medidas de Rendimiento	Medidas que determinan la evolución de los algoritmos genéticos. Algunos ejemplos son *On_Line*, *Off_Line*, Respuesta a la Selección, ...
MEF (FSM)	Máquina de Estados Finitos (*Finite State Machine*).
Método de búsqueda basado en el cálculo	Algoritmo de búsqueda que utiliza modelos analíticos y/o numéricos del espacio de soluciones como base para la búsqueda.
Método de búsqueda eficiente	Se dice que un método de búsqueda es eficiente si el número de soluciones evaluadas antes de encontrar una solución es pequeño en relación con el tamaño del espacio de soluciones posibles.
Método de búsqueda enumerativo	Mecanismo de búsqueda sistemática a través de todo el espacio de soluciones. No es practicable directamente cuando no se conoce el espacio de soluciones o éste es infinito.
Método de búsqueda heurístico	Intenta mejorar la eficiencia del método por enumeración utilizando la información disponible sobre el problema para dirigir la búsqueda. No asegura encontrar una solución aunque ésta exista.

Método de Codificación/Decodificación	Método usado para traducir las estructuras de búsqueda entre su representación real y su representación interna, y a la inversa.
Método de los mínimos cuadrados (LMS)	Método matemático para la determinación de la curva de regresión que indica la correlación entre dos variables a través de la minimización del error cuadrático medio.
Migración	Intercambio periódico de estructuras que se produce entre las subpoblaciones vecinas (islas) de un AG de grano grueso.
Modelo centralizado (Paralelización global)	Implementación distribuida de un AG secuencial que trabaja sobre una única gran población. El sistema se organiza en forma maestro-esclavos. La topología de interconexión típica es una estrella, con el maestro en el centro. Es un mecanismo de paralelización que únicamente proporciona incremento en la velocidad de ejececución sin alterar las propiedades de búsqueda secuenciales.
Modelo de ejecución homogéneo paralelo de algoritmos genéticos	Modelo de AG paralalelo que usa los mismos parámetros en su ejecución (exceptuando la semilla aleatoria) en cada una de las islas. Desde el punto de vista *hardware* se interpreta que cada isla se ejecuta sobre máquinas del mismo tipo.
Modelo distribuido	Organización de un AG paralelo que elimina la necesidad de utilizar una población compartida. Cada procesador posee una parte de la población total y ejecuta un algoritmo básico sobre ella con intercambios intermitentes de algunos individuos.
Modelo isla (isla o *stepping stone model*)	Modelo de AG distribuido que utiliza pequeñas subpoblaciones aisladas geográficamente y los individuos pueden migrar a cualquier subpoblación con o sin restricciones (algunas referencias los distinguen como modelos distintos).
Modelo semi-distribuido	Implementación de un AG distribuido que reduce el problema de cuello de botella motivado por la existencia de un único procesador maestro. La idea es tener grupos (*clusters*) de procesadores que trabajen como en una implementación centralizada.
Modelo totalmente distribuido	Variación en la implementación distribuida de AGs en la cual los individuos nunca son transferidos entre procesadores. Sólo existe comunicación entre los procesadores en las fases de inicialización y terminación.
Momento (*momentum*)	Parámetro numérico flotante que determina la influencia de los pesos anteriores en el cálculo de los nuevos pesos en el entrenamiento de una red neuronal usando *backpropagation* (PAE).
Mutación	Cambio brusco y permanente de uno o de varios caracteres hereditarios. La mutación renueva el material genético existente en la población, generando fenotipos que quizá no hubieran aparecido de otro modo.
Mutación indiferente	Tipo de mutación que no repercute ni favorable ni desfavorablemente sobre el individuo.
Mutación progresiva	Tipo de mutación en la que el gen mutado resulta más favorable para el individuo que aquel del cual procede.
Mutación regresiva	Tipo de mutación en la que el gen mutado es desfavorable, es decir, provoca algún efecto negativo sobre el organismo.
***N*-Reinas**	Problema consistente en situar un número *N* de reinas en un tablero de ajedrez de *N* filas y *N* columnas sin que se ataquen entre sí, teniendo en cuenta que dos reinas se atacan si están situadas en la misma fila, columna o diagonal.
Operador de Migración	Operador encargado del envío de individuos entre los distintos AGPs que se encuentran interconectados entre sí. Típicamente el criterio de migración está basado en un intervalo de iteraciones tras el cual se envían copias de algunos individuos a las poblaciones vecinas y se reciben individuos de ellas a su vez.
Operador genético de cruce heurístico *n*-ario	Cruce que combina dos o más individuos de la población para dar lugar a otro presumiblemente mejor utilizando un conjunto de suposiciones estándares sobre las restricciones del problema, el porcentaje de genes modificados, etc...

Operador genético de cruce heurístico unario	Cruce basado en la idea de mejorar un individuo mediante la modificación de algunos de sus genes.
Operadores genéticos	Conjunto de operaciones a las que se ven sometidos los individuos que componen una población genética. Típicamente las más utilizados son selección, cruce, mutación y reemplazo. Todas ellas suelen incorporarse ordenadamente como parte del plan reproductor.
Orden de un esquema	Número de posiciones definidas en un esquema.
Organismo diploide	Organismo en el que cada una de sus células contiene dos juegos de cromosomas aportados respectivamente por el gameto femenino y por el gameto masculino. Cada cromosoma de un juego tiene su homólogo en el otro. Los cromosomas homólogos tienen los mismos genes y, de esta forma, cada célula contiene dos genes para regir un carácter determinado.
Organismo haploide	Organismo en el que cada una de sus células contiene en su núcleo un juego de cromosomas.
OX (*order crossover*)	Operador genético de cruce que genera dos descendientes eligiendo para cada uno de ellos un sub-recorrido de uno de los progenitores y preservando el orden relativo de los restantes valores presentes en el otro progenitor. Explota la propiedad de que el orden de los elementos (no su posición) es importante. Por ejemplo, a partir de (1 2 3 | 4 5 6 7 | 8 9) y (4 5 2 | 1 8 7 6 | 9 3) elegiría dos puntos de cruce (marcados con barras) y generaría los hijos (2 1 8 4 5 6 7 9 3) y (3 4 5 1 8 7 6 9 2).
Panmixia	Modalidad centralizada de trabajo sobre una única población en la que se considera que cada estructura puede interaccionar con cualquier otra.
Paralelismo implícito (*implicit parallelism*)	Propiedad que tienen los AGs. Consistente en actuar sobre una elevada cantidad de esquemas en paralelo (n^3) cuando realmente trabaja sobre n cadenas de valores.
Paralelización global	AGP en el que la evaluación de los individuos y la aplicación de los operadores genéticos son explícitamente paralelizados, con excepción de la fase de selección.
Parálisis	Término usado para describir un estado de estancamiento durante el proceso de entrenamiento de una red de neuronas.
PAS (SAP)	Punto de Acceso al Servicio (*Service Access Point*).
Patrón de entrenamiento	Conjunto de valores de prueba (entrada/salida o sólo entrada) que se utilizan para entrenar una red de neuronas.
Perceptron	Red de neuronas no recurrente completamente conectada.
***PGA* (siglas inglesas para AGP)**	Algoritmo genético paralelo. En algunas -muy contadas- referencias se confunde este término con un modelo celular (masivamente paralelo) desarrollado por Mühlenbein.
PMX (*partially matched crossover*)	Operador genético de cruce que produce dos descendientes eligiendo un sub-recorrido de uno de los dos progenitores y preservando el orden y la posición de tantos valores como sea posible del otro progenitor (para cada uno de ellos). Explota las similitudes en el valor y el orden -simultáneamente-. Por ejemplo, a partir de (1 2 3 | 4 5 6 7 | 8 9) y (4 5 2 | 1 8 7 6 | 9 3) elegiría dos puntos de cruce (marcados con barras) y generaría los descendientes (4 2 3 1 8 7 6 5 9) y (1 8 2 4 5 6 7 9 3).
Población	Conjunto de estructuras que representan soluciones subóptimas al problema que se está intentando resolver con un algoritmo genético.
Proceso monitor	Proceso en ejecución que se encarga de la puesta en funcionamiento, gestión y finalización de todo el sistema de búsqueda paralela.

Progenitor	Individuo al que se la aplican uno o más operadores genéticos para obtener como resultado uno o más descendientes.
Programación orientada a objetos	Metodología de programación orientada a la reutilización del *software*. Esta metodología crea modelos del mundo real mediante clases de objetos e interacciones entre las mismas.
Protocolo	Conjunto de normas que rigen el comportamiento de un sistema paralelo.
Punto de cruce	Valor entre 0 y L-1, siendo L el número de alelos del cromosoma. Delimita el fragmento de cadena que se va a intercambiar de cada padre.
Random Walk	Búsqueda aleatoria.
Ranking	Algoritmo de selección de individuos de una población en el que la probabilidad de selección es proporcional al índice del individuo en la población, ordenada según algún criterio. Por ejemplo, el índice del mejor individuo puede ser igual al tamaño de la población menos uno y el del peor, cero.
RDG (GDR)	Regla Delta Geenralizada (*Generalized Delta Rule*).
Recocido simulado	Enfriamiento simulado.
Recombinación	Cruce.
Red de neuronas (RN)	Agrupación de elementos de procesamiento conectados entre sí por enlaces con pesos en forma de grafo dirigido, organizado de tal modo que la estructura de la red sea la adecuada para el problema que se éste considerando resolver.
Red *feed-forward* (no recurrente)	Red de neuronas en la que los datos fluyen desde la entrada hacia la salida sin bucles de realimentación.
Red recurrente	Red de neuronas en la que están permitidas las conexiones hacia atrás entre los elementos de proceso que componen sus capas.
Reemplazo de individuos	Técnica empleada para mantener constante el número de individuos que componen la población genética. Típicamente se usa el reemplazo del peor individuo (con el peor valor de adecuación asociada) o el reemplazo aleatorio.
Regla de aprendizaje *Hebbian*	Es la forma más simple de modificar los pesos de una RN. Consiste en reforzar la intensidad de un enlace (peso) si dos unidades *i* y *j* están activas simultáneamente.
Regla delta	Descripción matemática formal del algoritmo de aprendizaje para redes de neuronas con funciones de activación lineales; para las no lineales se utiliza la regla delta generalizada. Consiste en utilizar la diferencia entre el valor actual y el deseado de activación de los elementos de proceso para ajustar los pesos de los enlaces.
Regla delta generalizada (RDG, GDR)	Generalización de la regla delta para funciones no lineales. También llamada algoritmo de *backpropagation* o PEA.
Roulette Wheel	Método de selección proporcional la adecuación donde la probabilidad de que un individuo sea elegido es mayor cuanto mejor es su valor de adecuación.
SBR (RBS)	Sistema Basado en Reglas (*Rule Base System*).
SC (CS)	Sistema Clasificador (*Classifier System*).
Secuencial	RN: Modelo algorítmico en el que las actualizaciones de los elementos de procesamiento que intervienen se realizan de forma simultánea. AG: algoritmo genético en que las operaciones del plan reproductor se ejecutan una tras otra.

Selección — Operador genético que guía la búsqueda hacia los lugares más prometedores del espacio de soluciones. Los individuos de más alto valor de adecuación deberían ser seleccionados frecuentemente para recombinarlos con el objeto de que sus descendientes hereden sus características. Sin embargo, una selección frecuente del mismo individuo puede provocar que toda la población esté *salpicada* con sus características, lo cual redunda en una pérdida de la diversidad. Por tanto, la clave del éxito para el operador de selección es permitir el equilibrio correcto entre la explotación y la exploración del espacio de soluciones. La selección se puede realizar de acuerdo a diferentes estrategias: el método de la ruleta (*roulette wheel*), el mecanismo *ranking*, selección aleatoria, mecanismos elitistas, etc... La selección puede usarse para elegir pareja antes de la recombinación o bien para elegir al individuo que va a ser reemplazado.

Selección elitista — Estrategia de selección que estipula que el individuo o estructura que ofrezca mejor rendimiento sobreviva de una generación a otra. Puede extenderse para hacer sobrevivir un porcentaje de individuos que pasarán a formar parte de la siguiente generación determinísticamente.

Selección natural — Teoría enunciada por Darwin como principal motor de la evolución. La selección natural es la consecuencia de la lucha por la vida y se supone que conduce a la supervivencia del más apto.

SPX (*single point crossover*) — Operador genético binario de único punto de cruce. Funciona seleccionando un punto aleatoriamente en las cadenas padres e intercambiando el contenido entre ellos para producir dos descendientes.

Steady-state — Esquema evolutivo por efectos inmediatos o de estado estacionario.

Tamaño del grano — Cuando nos referimos a un sistema paralelo, es la proporción entre el tiempo consumido en la computación y el consumido en comunicación.

Tasa de migración (*migration rate*) — Parámetro que controla cuántos individuos migran de una subpoblación a otra en un algoritmo genético distribuido. A veces se confunde con la frecuencia de migración entre islas de un AG distribuido.

Teoría de nichos y especiación (*niching and speciation theory*) — Teoría surgida de las observaciones biológicas en ambientes aislados. A menudo existen especies de animales que están adaptadas más específicamente a las peculiaridades de sus ambientes que las especies que viven en áreas de amplia extensión.

Tolerante a fallos — Propiedad de una red de neuronas consistente en producir una salida incluso en el caso de que se le presente una entrada que no ha visto nunca o que contiene ruido. En el caso de un sistema *software* dícese de la capacidad de dicho sistema para degradar sus prestaciones gradualmente conforme fallan componentes internos.

Topología — RN: indicación sobre cuántos nodos se utilizan y cómo se conectan. AGP: definición de las conexiones entre las subpoblaciones.

Topología de Interconexión — Esquema utilizado para la conexión de los distintos AGPs. Típicamente se basan en sistemas espaciales unidimensionales (ej. anillos), bidimensinales (ej. matrices, toros, etc..) o tridimensionales (ej. esferas) donde se definen vecindades para cada uno de los puntos que componen el espacio de coordenadas.

UX (*Uniform crossover*) — Cruce uniforme. Para cada posición de la cadena descendiente, cada padre aporta con cierta probabilidad (dada por el usuario) cada uno de sus genes. En la mayoría de aplicaciones se utiliza 0.5 como probabilidad (no es la prob. de aplicación).

Valor de activación — Valor resultante de la transformación que sufre la entrada neta en un elemento de proceso de una red de neuronas. En la mayoría de los casos, la activación y la entrada neta son idénticas, y los términos suelen emplearse indistintamente.

Valor de entrada neto	Valor numérico calculado sumando los valores de entrada a un nodo en una red de neuronas, ponderados (multiplicados) por sus correspondientes pesos.
Valor de salida	Es el valor que proporciona la neurona una vez aplicada la función de salida al valor de activación.
Velocidad de aprendizaje (*learning rate*)	Es una constante positiva de proporcionalidad. Determina qué porcentaje de cambio se aplica sobre los pesos de una RN durante el proceso de aprendizaje usando la regla delta generalizada.

ACRÓNIMOS

AGs Algoritmos Genéticos.
AG Algoritmo Genético.
AG Inc. Algoritmo Genético Incremental (incGA).
AG Sec. Algoritmo Genético Secuencial (seqGA).
AGP Algoritmo Genético Paralelo (PGA).
AR Avance rápido.
AX *Arithmetic Crossover*. Cruce Aritmético.
ca Algoritmo genético celular que reemplaza siempre a la cadena actualmente considerada.
CE *Computación Evolutiva*
cGA *Cellular Genetic Algorithm*.
ci Algoritmo genético celular con reemplazo de la cadena actualmente considerada sólo si la nueva generada es mejor que ella.
dcGA *Distributed Cellular Genetic Algorithm*.
dGA *Distributed Genetic Algorithm*.
DseqGA *Distribution of Sequential island GAs.*
DPX *Double Point Crossover*. Cruce de dos puntos.
EJCC *Eastern Joint Computer Conference*. Conferencia de computadores del comité del este.
ES Enfriamiento simulado. Recocido Simulado. Recristalización Simulada.
genGA *Generational Genetic Algorithm*.
ICC *Iterative Circuit Computers*. Computadoras de circuitos iterativos.
ICGA *International Conference on Genetic Algorithms*. Conferencia internacional sobre algoritmos genéticos.
LMS *Least Mean Squares*.
MSE *Mean Square Error.* Error Cuadrático Medio.
PD Programación Dinámica.
POO Programación Orientada a Objetos.
PPSN *Parallel Problem Solving from Nature*. Conferencia Internacional celebrada en Europa.
RW *Roulette Wheel*. Selección proporcional a la adecuación por ruleta.
SA *Simulated Annealing*. Enfriamiento simulado. Recocido Simulado.
SoC *String of Change*. Operador de cruce de "cadena de cambio".
SPX *Single Point (X)crossover*. Operador de cruce de un sólo punto.
SS *Steady-State*. Estado estacionario. Método de generación reemplazo de individuos.
ssa Algoritmo secuencial de estado estacionario con reemplazo de la peor cadena siempre.
ssGA *Steady State Genetic Algorithm*.
ssi Algoritmo secuencial de estado estacionario con reemplazo de la peor estructura sólo si la nueva es mejor.
SUS *Stochastic Universal Sampling*. Muestreo estocástico universal (selección).
UX *Uniform (X) Crossover*. Operador de cruce uniforme.
VAD-U Variable Aleatoria Discreta siguiendo una distribución Uniforme.
xxGA *Excess Genetic Algorithm*. Plantilla genérica de especificación de AGs paralelos.

Apéndice **D**
Resumen de la Relación con Otras Técnicas

A continuación presentamos un resumen comparativo que resalta las principales diferencias entre las siguientes técnicas:

- Algoritmos Genéticos Paralelos.
- Escalada (*Hill Climbing*).
- Redes de Neuronas.
- Enfriamiento Simulado (*Simulated Annealing*).
- Diseño de Redes de Neuronas con Algoritmos Genéticos (RNG) -aplicación-.

En todos estos paradigmas consideraremos que el objetivo es la búsqueda en algún espacio problema, de forma adaptativa y/o guiada, de modo que la información obtenida en un estado de la búsqueda se utilice para influenciar la dirección futura de la misma.

Una búsqueda adaptativa e inteligente a través de cualquier espacio de búsqueda implica comenzar con una o más estructuras (soluciones al problema o puntos) de dicho espacio, calcular su adecuación (*fitness*), y con esta información modificar (esperando mejorar) la(s) estructura(s) actual(es). La trayectoria de la búsqueda en el espacio de estructuras posibles depende de la información obtenida hasta el momento. La adaptación trae consigo el cambio de alguna estructura a fin de mejorar su comportamiento.

Consideraremos cada uno de estos paradigmas en términos de las ocho perspectivas comunes para los sistemas adaptativos descritas por Koza en [Koza92]:

- Las estructuras que sufren adaptación.
- Generación de las estructuras iniciales.
- El estado (memoria) del sistema en cada etapa.
- El criterio de terminación del proceso.
- Las operaciones que modifican las estructuras.
- La medida de adecuación usada para evaluar las estructuras.
- Los parámetros que controlan el proceso.
- El método para la designación de un resultado.

Al hacer esta comparación se ha de tener en cuenta que cada paradigma admite numerosas variantes. La descripción concreta de cada paradigma en cuanto a las ocho perspectivas, depende, por supuesto, de la variante particular que se esté utilizando. Para este resumen consideraremos versiones canónicas y apuntaremos algunas de estas variantes.

Cuando nos referimos a un **algoritmo genético paralelo** (AGP) informalmente su funcionamiento viene definido por las siguientes características:

1. El esquema de representación (esto es, el alfabeto, la longitud de las estructuras y la aplicación genotipo-a-fenotipo).
2. La medida de adecuación definida.
3. Los parámetros que controlan el algoritmo.
4. El criterio de terminación y el método para designar un resultado.

En cuanto a la **escalada** nos referimos a una escalada simple, donde la búsqueda comienza en un punto aleatorio, se examinan uno o más puntos que están a cierta distancia del punto actual, y la búsqueda continúa desde el mejor de los puntos examinados. La escalada se caracteriza por:

1. El esquema de representación (esto es, la aplicación entre el problema y los puntos en el espacio de búsqueda multidimensional).
2. La medida de la adecuación para los puntos en el espacio de búsqueda.
3. Los parámetros (es decir, el tamaño del paso y el número de puntos alternativos que considerar antes de que se dé un paso) para controlar el algoritmo.
4. El criterio de terminación y el método para designar un resultado.

Podemos considerar una red de neuronas definida en base a [RM89] [Kung93]:

1. La arquitectura de la red, es decir, el número de capas, número de elementos de proceso por capa, etc... No están incluidos aquí ni los nodos de entrada ni los de salida, pues se consideran aparte por su especial importancia.
2. La conectividad de la red, por ejemplo total o parcialmente conectada entre capas consecutivas, si la red es recurrente o no, qué conexiones de una capa a otra anterior están permitidas, etc.
3. El tipo de elemento de proceso usado, por ejemplo lineal, sigmoidal, etc.
4. El paradigma de entrenamiento, por ejemplo propagación hacia atrás del error (PAE) o *backpropagation*.
5. Los nodos de entrada y de salida de la red.
6. Los pares de entrenamiento y de generalización utilizados (suponiendo entrenamiento supervisado).
7. El criterio para medir el error, por ejemplo el error cuadrático medio (ECM o MSE).
8. Los valores de los parámetros para controlar el diseño o el uso, esto es, la velocidad de aprendizaje para *backpropagation*, rango de creación de los pesos iniciales, ...
9. El criterio para designar un resultado y terminar la ejecución, por ejemplo detener el entrenamiento una vez alcanzado cierto error.

Cuando nos referimos al **enfriamiento simulado** (*simulated annealing*) pensamos en una técnica de optimización estocástica básica. El enfriamiento simulado comienza con una única estructura específica del dominio y un método para modificar dicha estructura, definidos ambos por el usuario normalmente. Sus características son:

1. El esquema de representación para hacer corresponder los puntos individuales del espacio de búsqueda del problema a la estructura.
2. La operación para modificar las estructuras, incluyendo la técnica para la definición de los vecinos y parámetros numéricos asociados a esta operación.
3. Los parámetros para controlar el algoritmo, por ejemplo el plan de enfriamiento.
4. El criterio de terminación y el método para designar un resultado.

En cuanto al **diseño de redes con algoritmos genéticos** (red de neuronas genética o RNG) nos referimos a una combinación de AGs y RNs. El algoritmo genético se utiliza para encontrar los pesos que provocan el comportamiento deseado de la red. Esta aplicación tiene las características combinadas de ambas técnicas. Entre las más importantes se encuentran:

1. La aplicación entre los pesos de la red y el genotipo.
2. La medida de adecuación, por ejemplo el error cuadrático medio que comete la red sobre el conjunto de patrones de entrenamiento.
3. Todos los parámetros que controlan el algoritmo genético y la red de neuronas.
4. El criterio de terminación y el método para designar un resultado.

A continuación se muestra en varias tablas la comparación de los puntos mencionados.

TABLA D.1. COMPARACIÓN ATENDIENDO A LAS ESTRUCTURAS QUE SUFREN LA ADAPTACIÓN

Paradigma	Estructura que Sufre Adaptación
Algoritmo Genético Paralelo	Múltiples vectores de símbolos (binarios, reales, ...)
Escalada	Un único punto en el espacio de búsqueda
Red de Neuronas	Un único vector de pesos en el espacio de los reales
Enfriamiento Simulado	Una única estructura específica en el espacio de búsqueda
RNG	La población consta de vectores numéricos de longitud fija que representan a los pesos y *bias* de la red de neuronas

TABLA D.2. COMPARACIÓN ATENDIENDO A LA GENERACIÓN DE LAS ESTRUCTURAS INICIALES

Paradigma	Generación de Estructuras Iniciales
Algoritmo Genético Paralelo	Estructuras aleatorias o elegidas sobre el alfabeto definido
Escalada	Normalmente un punto inicial aleatorio en el espacio de búsqueda, si no, un punto que el usuario determina que será un buen punto de comienzo para la búsqueda
Red de Neuronas	Para *backpropagation*, se crea aleatoriamente un vector de pesos en un rango dado ([-0.1..+0.1] por ejemplo)
Enfriamiento Simulado	Bien una estructura inicial aleatoria o bien una estructura que se comporta bien generada por un método previo
RNG	Vectores numéricos aleatorios iniciales ([-0.1..+0.1] por ej.)

TABLA D.3 COMPARACIÓN ATENDIENDO AL ESTADO (MEMORIA) DEL SISTEMA

Paradigma	Estado (Memoria)
Algoritmo Genético Paralelo	Las subpoblaciones (islas) o vecindarios de estructuras
Escalada	El punto actual en el espacio de búsqueda
Red de Neuronas	El vector actual de pesos en el espacio de pesos admisibles
Enfriamiento Simulado	La estructura actual
RNG	La población (o subpoblaciones) de pesos

TABLA D.4. COMPARACIÓN ATENDIENDO AL CRITERIO DE TERMINACIÓN

Paradigma	Criterio de Terminación
Algoritmo Genético Paralelo	Después de un número específico de iteraciones o cuando se obtiene un resultado aceptable según cierto criterio dado
Escalada	Cuando no se produce una mejora en los puntos alternativos cercanos al punto actual (quien puede no ser un óptimo)
Red de Neuronas	Cuando no se produce mejora a partir del punto actual del espacio de pesos admisibles o se alcanza cierto valor de error
Enfriamiento Simulado	Cuando no se produce mejora y el plan de enfriamiento ha sido completamente ejecutado
RNG	Después de un número específico de iteraciones o cuando se obtiene un resultado aceptable (error tolerado máximo)

TABLA D.5. COMPARACIÓN DE LAS OPERACIONES PARA MODIFICAR LAS ESTRUCTURAS

Paradigma	Operaciones para Modificar las Estructuras
Algoritmo Genético Paralelo	Selección, cruce, mutación, migración, reemplazo, vecindad, ...
Escalada	Utiliza la información del gradiente para moverse desde el punto actual a los mejores puntos cercanos examinados (esto es, se mueve en la dirección mayor pendiente)
Red de Neuronas	Modifica los pesos del vector de pesos utilizando para ello la medida del error y la regla Delta Generalizada
Enfriamiento Simulado	El usuario define un método específico del dominio para modificar cualquier estructura. La estructura existente es provisionalmente modificada y se determina su nivel de energía. Si el nivel de energía de la modificación es mejor la modificación siempre se acepta. Si el nivel de energía de la modificación no es mejor, la modificación puede todavía ser aceptada con una cierta probabilidad determinada por la ecuación de Boltzman[1].
RNG	Selección, cruce, mutación, PAE, enfriamiento simulado, ...

TABLA D.6. COMPARACIÓN ATENDIENDO A LA MEDIDA DE LA ADECUACIÓN

Paradigma	Medida de la Adecuación
Algoritmo Genético Paralelo	Adecuación numérica (valor real) normalizada
Escalada	Energía de un punto en el espacio de búsqueda
Red de Neuronas	Error cuadrático medio
Enfriamiento Simulado	Energía de la estructura actual
RNG	Error cuadrático medio promediado para todos los patrones

[1] $p_i = \frac{1}{1 + e^{\frac{-\Delta E_i}{T}}}$; $p_i \to 1$ cuando $\Delta E_i \to \infty$

TABLA D.7. COMPARACIÓN ATENDIENDO A LOS PARÁMETROS DE CONTROL

Paradigma	Parámetros de Control	
Algoritmo Genético Paralelo	Número de subpoblaciones Máximo número de iteraciones Probabilidad de cruce Probabilidad de mutación Política de selección Política de reemplazo	Tamaño de cada subpobl. Esquema de vecindad Política de migración Sincronismo Homogeneidad ...
Escalada	Tamaño del paso Número de puntos alternativos que considerar antes de avanzar un paso	
Red de Neuronas	Número de capas de la red Número de neuronas en cada capa Umbral para cada neurona Función de transferencia y de Activación *Bias* (si existen) Si la red es recurrente o no Conexiones entre las capas de elementos de proceso Velocidad de aprendizaje y momento (para *backpropagation*) Rango para los pesos iniciales	
Enfriamiento Simulado	Plan de enfriamiento para variar la temperatura (es decir, decrementarla) conforme avanza el tiempo de proceso El número de vecinos posibles que puede producir la operación de modificación y el esquema de actualización de la estructura actualmente considerada A menudo, un tamaño de paso para controlar la distancia en el espacio de búsqueda entre la estructura actual y los posibles vecinos que se producen	
RNG	Como mínimo los parámetros de un algoritmo genético y los de una red de neuronas. Como máximo todos los parámetros que aparecen en esta tabla antes de este punto	

TABLA D.8. COMPARACIÓN ATENDIENDO AL MÉTODO DE DESIGNACIÓN DE RESULTADOS

Paradigma	Designación de Resultados
Algoritmo Genético Paralelo	La mejor estructura encontrada a lo largo de toda la evolución
Escalada	El punto actual del espacio de búsqueda en el momento de la terminación
Red de Neuronas	El vector actual de pesos en el espacio de pesos en el momento de la terminación
Enfriamiento Simulado	La estructura actual en el momento de la terminación
RNG	La mejor red encontrada hasta el momento de la terminación

Bibliografía

[Adam94] Adamidis P. (1994). "Review of Parallel Genetic Algorithms Bibliography (v1.0)". *Internal T.R., Aristotle University of Thessaloniki*, http://www.control,ee.auth.gr/~panos/papers/pga_review.ps.gz).

[Akl92] Akl S. G. (1992). *Diseño y Análisis de Algoritmos Paralelos*. RA-MA.

[Alan96] Alander J. T. (1996). *An Indexed Bibliography of Genetic Algorithms*. Report Series No. 94-1-EVONET, Draft September 20. ftp: ftp.uwasa.fi/cs/report94-1/gaEVONETbib.ps.Z.

[Alba93] Alba E. (1993). "Aplicación de los Algoritmos Genéticos para el Diseño de Redes Neuronales". Sáez deVacas F. (ed.), *Informática y Automática* 26 (2), 22-35, Madrid.

[Alda93] Aldana J. F. (1993). "A Dataflow Model for Datalog Parallel Evaluation" *Tech. Rep. Dpt Lenguajes y Ciencias de la Computación*. Univ. of Málaga.

[Anto89] Antonisse J. (1989). "A New Interpretation of Schema Notion that Overturns the Binary Encoding Constraint". In [Scha89], 86-91.

[AP96] Adamidis P., Petridis V. (1996). "Co-operating Populations with Different Evolution Behavior". *Proceedings of the Second IEEE Conference on Evolutionary Computation*, 188-191.

[ARS93] Albretch R. F., Reeves C. R., Steele N. C. (eds.) (1993). *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*. Springer-Verlag.

[AT91] Alippi C., Treleaven P. (1991). "GAME: A Genetic Algorithms Manipulation Environment". *Internal Report Department of Computer Science*, UCL.

[AT95] Alba E., Troya J. M. (1995). "Algoritmos Genéticos Incrementales". *Informe Técnico* ITI 95/15, Dpto. Lenguajes y Ciencias de la Computación.

[AT96] Alba E., Troya J. M. (1996). "Genetic Algorithms for Protocol Validation". Voigt H. M., Ebeling W., Rechenberg I., Schwefel H. P. (eds.), *Proceedings of the Parallel Problem Solving from Nature IV* I.C., Berlin. Springer-Verlag, 870-879.

[AT99a] Alba E., Troya J. M. (1999). "A Survey of Parallel Distributed Genetic Algorithms". *Complexity* (*to appear*). John Wiley & Sons.

[AT99b] Alba E., Troya J. M. (1999). "An Analysis of Synchronous and Asynchronous Parallel Distributed Genetic Algorithms with Structured and Panmictic Islands". In Zomaya A. Y., Ercal F., Olariu S. (eds.), *Proceedings of the BioSP3 Workshop* (*to appear*)*, IPPS/SPDP*, Springer-Verlag.

[Bäck96] Bäck T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

[Bäck97a] Bäck T. (ed.) (1997). *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, CA.

[Bäck97b] Bäck T. (ed.) (1997). *Proceedings of the Fourth International Conference on Evolutionary Computation*. New-York: IEEE Press.

[Bake87] Baker J. E. (1987). "Reducing Bias and Inefficiency in the Selection Algorithm". *Proceedings of the Second ICGA*, Grefenstette J. J. (ed.), LEA publishers, 14-21.

[Balu93] Baluja S. (1993). "Structure and Performance of Fine-Grain Parallelism in Genetic Search". In [Forr93], 155-162.

[BB88] Brassard G., Bratley P. (1988). *Algorithmics, Theory and Practice.* Prentice-Hall International.

[BB91] Belew r. K., Booker L. B. (eds.) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, California.

[BB93] Bianchini R., Brown C. (1993). "Parallel Genetic Algorithms on Distributed-Memory Architectures". *Technical Report 436* (revised version), May.

[Beld95] Belding T. C. (1995). "The Distributed Genetic Algorithm Revisited". In [Eshe95], 114-121.

[Bezd94] Bezdek J. C. (1994). "What is Computational Intelligence?". *Computational Intelligence: Imitating Life*. Zurada J. M., Marks R. J., Robinson Ch. J. (eds.). IEEE Press, 1-12.

[BFM97] Bäck T., Fogel D., Michalewicz Z. (eds.) (1997). *Handbook of Evolutionary Computation* (Oxford University Press).

[BG87] Bridges C. L., Goldberg D. E. (1987). "An Analysis of Reproduction and Crossover in a Binary-Coded Genetic Algorithm". In [Gref87], 9-13.

[BGKK97] Bäck T., Graaf J. M., Kok J. N., Kosters W. A. (1997). "Theory of Genetic Algorithms". *Bulletin of the European Association for Theoretical Computer Science*, no. 63, 161-192.

[BHS91] Bäck T., Hoffmeister F., Schwefel H. P. (1991). "A Survey of Evolution Strategies". In [BB91], 2-9.

[BHS97] Bäck T., Hammel U., Schwefel H. P. (1997). "Evolutionary Computation: Comments on the History and Currrent State". *IEEE Transactions on Evolutionary Computation* 1(1) 3-17.

[Boni97] Bonissone P. P. (1997). "Soft Computing: the Convergence of Emerging Reasoning Technologies". *Journal of Research in Soft Computing*, 1(1) 6-18.

[Box57] Box G. E. P. (1957). "Evolutionary Operation: A Method for Increasing Industrial Productivity". *Applied Statistics*, VI(2), 81-101.

[Brem62] Bremermann H. J. (1962). "Optimization through Evolution and Recombination". Self-Organizing Systems, Yovits M. C. et al. (eds.). Washington DC: Spartan.

[BS93] Bäck T., Schwefel H. P. (1993). "An Overview of Evolutionary Algorithms for Parameter Optimization". *Evolutionary Computation*, 1 (1), 1-23, The MIT Press.

[BS93] Bäck T., Schwefel H. P. (1993). "An Overview of Evolutionary Algorithms for Parameter Optimization". *Evolutionary Computation*, 1 (1), 1-23, The MIT Press.

[Cant94] Cantú-Paz E. (1994). "Parallel Genetic Algorithms in Distributed Memory Architectures". *Master Thesis Dissertation*, Instituto Tecnológico Autónomo de México (in spanish).

[Cant97a] Cantú-Paz E. (1997). "A Survey of Parallel GAs", *IlliGAL R. 97003*. Revised version of "A Summary of Research on Parallel Genetic Algorithms". *R. 95007*, July 1995.

[Cant97b] Cantú-Paz E. (1997). "Designing Efficient Master-Slave Parallel Genetic Algorithms". *IlliGAL report 97004* (May 1997).

[CAT96] Cotta C., Alba E., Troya J. M. (1996). "Evolutionary Design of Fuzzy Logic Controllers". IEEE Catalog Number 96CH35855, *Proceedings of the ISISC'96 Conference*, Detroit. Dearborn, MI, 127-132.

[CAT98a] Cotta C., Alba E., Troya J. M. (1998). "Un Estudio de la Robustez de los Algoritmos Genéticos Paralelos". *Revista Iberoamericana de Inteligencia Artificial*, 5 (V/98), 6-13.

[CAT98b] Cotta C., Alba E., Troya J. M. (1998). "Utilising Dinastically Optimal Forma Recombination in Hybrid Genetic Algorithms ". In Eiben A. E., Bäck T., Schoenauer M., Schwefel H.-P. (eds.), *Proceedings of the Parallel Problem Solving From Nature (PPSN) V International Conference*, Amsterdam. Springer-Verlag, 305-314.

[CCFM93] Cammarata G., Cavalieri S., Fichera A., Marletta L. (1993). "Noise Prediction in Urban Traffic by a Neural Approach". *Proceedings of the International Workshop on Artificial Neural Networks*, Mira J., Cabestany J., Prieto A. (eds.). Springer-Verlag, 611-619.

[CF96] Chipperfield A., Fleming P. (1996). "Parallel Genetic Algorithms". *Parallel and Distributed Computing Handbook*, Zomaya A. Y. H. (ed.), MacGraw-Hill, 1118-1143.

[CG97] Cantú-Paz E., Goldberg D. E. (1997). "Predicting Speedups of Idealized Bounding Cases of Parallel Genetic Algorithms". In [Bäck97a], 113-120.

[CGT90] Ceri, Gottlob, Tanca (1990). "Logic Programming and Databases". *Surveys in Computer Science*. Springer Verlag.

[CJ91] Collins R.J., Jefferson D.R. (1991). "Selection in Massively Parallel Genetic Algorithms". In [BB91], 249-256.

[Come91] Comer D. E. (1991). *Internetworking with TCP/IP, Vols 1, 2, 3*. Prentice-Hall.

[Cott98] Cotta C. (1998). *Un Estudio de las Técnicas de Hibridación y su Aplicación al Diseño de Algoritmos Evolutivos*. Tesis Doctoral. Dpto. Lenguajes y CC.CC., Univ. Málaga.

[CPRS96] Corno F., Prinetto P., Rebaudengo M., Sonza-Reorda M. (1996). "Exploiting Competing Subpopulations for Automatic Generation of Test Sequences for Digital Circuits". In [VERS96], 792-800.

[CS88] Caruna R. A., Schaffer J. D. (1988). "Representation and Hiddn Bias: Gray vs. Binary Coding for Genetic Algorithms". Proceedings of the 5th International Conference on Machine Learning, Laird J. (ed.), Morgan Kaufmann, 153-15.

[CW93] Corcoran A. L., Wainwright R. L. (1993). "LibGA: A User-Friendly Workbench for Order-Based Genetic Algorithm Research". *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing, ACM Press*.

[Davi89] Davis L. (1989). "Adapting Operator Probabilities in Genetic Algorithms". In [Scha89], 61-69.

[Davi91a] Davis L. (ed.) (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

[Davi91b] Davidor Y. (1991). "A Naturally Occuring Niche & Species Phenomenon: The Model and First Results". In [BB91], 257-263.

[DEC94] DEC (1994). *MPI: A Message-Passing Interface Standard*. Message Passing Interface Forum.

[DeJo92] DeJong K. (1992). "Learning with Genetic Algorithms: An Overview". In *Genetic Algorithms*, Buckles B. P., Petry F. E. (eds.) IEEE Computer Society Press, 30-47.

[DG89] Deb K., Goldberg D. E. (1989). "An Investigation of Niche and Species Formation in Genetic Function Optimization". In [Scha89], 42-50.

[DG91] Davis L., Grefenstette J. J. (1991). "Concerning GENESIS and OOGA". *Handbook of Genetic Algorithms,* L. Davis (ed.), New York: Van Nostrand Reinhold, 374-376.

[Domí97] Domínguez-Cobos J. F. (1997). *Paralelización de Algoritmos Genéticos Mediante el uso de Técnicas de Orientación a Objetos*. PFC197, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[Dora97] Dorado-Ruíz A. J. (1997). *Técnicas Heurísticas Distribuídas frente a Técnicas Tradicionales para la Resolución de Problemas NP-Completos*. PFC195, Dpto. L. y CC.CC., Univ. Málaga.

[DRH95] Daida J. M., Ross S. J., Hannan B. C. (1995). "Biological Symbiosis as a Metaphor for Computational Hybridization". In [Eshe95], 328-335.

[DS95] DeJong K., Sarma J. (1995). "On Decentralizing Selection Algorithms". In [Eshe95], 17-23.

[ER96] East I. R., Rowe J. (1996). "Effects of Isolation in a Distributed Population Genetic Algorithm". In [VERS96], 408-419.

[ES91] Eshelman L. J., Schaffer J. D. (1991). "Preventing Premature Convergence in GAs by Preventing Incest". In [BB91], 115-122.

[Alba92] Alba E. (1992). *Redes Neuronales Genéticas: Los Algoritmos Genéticos como Heurístico para la Optimización del Diseño de Redes Neuronales (Vol. I y II)*. PFC7, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[AA92] Alba E., Aldana J. F. (1992). "Los Algoritmos Genéticos como Heurísticos en Problemas de Optimización". *Informe Técnico ITD 92/3*, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[AAT92] Alba E., Aldana J. F., Troya J. M. (1992). "Genetic Algorithms as Heuristics for Optimizing ANN Design". *Informe Técnico ITI 92/4*, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[AAT93a] Alba E., Aldana J. F., Troya J. M. (1993). "Full Automatic ANN Design: A Genetic Approach". J. Mira J., Cabestany J., Prieto A. (eds.), IWANN'93. *New Trends in Neural Computation. Lecture Notes in Computer Science 686*, 399-404, Springer-Verlag, Sitges.

[AAT93b] Alba E., Aldana J. F., Troya J. M. (1993). "Genetic Algorithms as Heuristics for Optimizing ANN Design". In [ARS93], 683-690.

[AAT94a] Aldana J. F., Alba E., Troya J. M. (1994). "Load Balancing and Query Optimization in Dataflow Parallel Evaluation of Datalog Programs". Ni L. M. (ed.), ICPADS'94. *Proceedings of the International Conference on Parallel and Distributed Systems*, Taiwan. IEEE Computer Society Press, 682-688.

[AAT94b] Aldana J. F., Alba E., Troya J. M. (1994). "D^2: A Model for Datalog Parallel Evaluation". Alpuente M., Barbuti R., Ramos I. (eds.), GULP-PRODE'94. *Proceedings of the 1994 Joint Conference on Declarative Programming*, Vol II, 60-74.

[AAT95] Alba E., Aldana J. F., Troya J. M. (1995). "A Genetic Algorithm for Load Balancing in Parallel Query Evaluation for Deductive Relational Databases". In [PSA95], 479-482.

[AC97] Alba E., Cotta C. (1997). "Evolución de Estructuras de Datos Complejas". Sáez deVacas F. (ed.), *Informática y Automática*, 30(3), 42-60.

[AC98] Alba E., Cotta C. (1998). "The On-Line Tutorial on Evolutionary Computing". *3rd On-line World Conference on Soft Computing in Engineering Design and Manufacturing (WSC3)*, `http://www.crandfield.ac.uk/WSC3`.

[ACT96] Alba E., Cotta C., Troya J. M. (1996). "Type-Constrained Genetic Programming for Rule-Base Definition in Fuzzy Logic Controllers". Koza J. R., Goldberg, D. E., Fogel D. B. & Riolo R. L. (eds.), *Proceedings of the First Annual Conference on Genetic Programming*, Stanford Univ., Cambridge, MA. The MIT Press, 255-260.

[ACH97] Alba E., Cotta C., Herrera F. (eds.) (1997). *Actas del Primer Seminario sobre Computación Evolutiva: Teoría y Aplicaciones*. Dpto. Lenguajes y CC.CC., Univ. Málaga.

[ACT99] Alba E., Cotta C., Troya J. M. (1999). "Evolutionary Design of Fuzzy Logic Controllers Using Strongly-Typed GP". *Mathware and Soft Computing* (*to appear*).

[AD97] Alba E., Durán J. C. (1997). "Un Acercamiento Natural al Problema de las N-Reinas". Actas de la Conferencia de la Asociación Española Para la Inteligencia Artificial, 725-734.

[ESG91] Erickson J. A., Smith R. E., Goldberg D. E.. (1991). "SGA-Cube, A Simple Genetic Algorithm for nCUBE 2 Hypercube Parallel Computers". *TCGA Report No. 91005*, The Univ. of Alabama.

[Eshe93] Eshelman L. J., Schaffer J. D. (1993). "Real-Coded Genetic Algorithms and Interval-Schemata". *Foundations of GA's 2*, Morgan Kaufmann, 187-202.

[Eshe95] Eshelman L. J. (ed.) (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, CA.

[FAT94] Filho J. R., Alippi C., Treleaven P. (1994). "Genetic Algorithm Programming Environments". *IEEE Computer Journal*.

[FG97] Fogel D. B., Ghozeil A. (1997). "A Note on Representations and Variation Operators". *IEEE Transactions on Evolutionary Computation*, 1(2), 159-161.

[Foga89] Fogarty T. C. (1989). "Varying the Probability of Mutation in the Genetic Algorithm". In [Scha89], 104-109.

[Foge98] Fogel D. B. (ed.) (1998). *Evolutionary Computation. The Fossil Record (Selected Readings on the History of Evolutionary Algorithms).* IEEE Press.

[Forr93] Forrest S. (ed.) (1993). *Proceedings of the Fith International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, California.

[FOW66] Fogel L. J., Owens A. J., Walsh M. J. (1996). *Artificial Intelligence through Simulated Evolution*. Wiley.

[Frie58] Friedberg R. M. (1958). "A Learning Machine: Part I". *IBM Journal*, 2(1), 2-13.

[Garc96] García-Moreno M. (1996). *Hibridación de Técnicas Genéticas en el Entrenamiento Distribuido de Redes Perceptrón Multicapa*. PFC154, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[GBDJM94] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. (1994). *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press.

[GD91] Goldberg D. E., Deb K. (1991). "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms". *Foundations of Genetic Algorithms* 1, Rawlins G. J. E. (ed.), Morgan Kaufmann, 69-93.

[GDH92] Goldberg D. E., Deb K., Horn J. (1992). "Massively Multimodality, Deception and Genetic Algorithms". *Parallel Problem Solving from Nature* 2, Männer R., Manderick B. (eds.), North-Holland, 37-46.

[GDK91] Goldberg D. E., Deb K., Korb B. (1991). "Don't Worry, Be Messy". In [BB91], 24-30.

[Gold89a] Goldberg D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

[Gold89b] Goldberg D. E. (1989). "Sizing Populations for Serial and Parallel Genetic Algorithms". In [Scha89], 70-79.

[Gold91] Goldberg D. E. (1991). "Real-Coded Genetic Algorithms, Virtual Alphabets and Blocking". *Complex Systems* 5, 139-167.

[Gold95] Goldberg D. E., et al. (1995). "Critical Deme Size for Serial and Parallel Genetic Algorithms". *IlliGAL Report No. 95002*.

[Good96] Goodman E. D. (1996). *An Introduction to GALOPPS v3.2. TR#96-07-01*, GARAGe, I. S. Lab., Dpt. of C. S. and C. C. C. A. E. M., Michigan State University, East Lansing.

[Gorg89] Gorges-Schleuter M. (1989). "ASPARAGOS An Asynchronous Parallel Genetic Optimisation Strategy". In [Scha89], 422-427.

[Gorg97] Gorges-Schleuter M. (1997). "Asparagos96 and the Traveling Salesman Problem". In [Bäck97b], 171-174.

[GR87] Goldberg D. E., Richardson J. (1987). "Genetic Algorithms with Sharing for Multimodal Function Optimization". In [Gref87], 41-49.

[Gref84] Grefenstette J. J. (1984). "GENESIS: A System for Using Genetic Search Procedures". *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, 161-165.

[Gref87] Grefenstette J. J. (ed.) (1987). *Genetic Algorithms and Their Applications, Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates.

[Gref90] Grefenstette J. J. (1990). "A User's Guide to GENESIS. Version 5.0". *Technical Report*.

[Grua94] Gruau F. (1994). *Neural Networks Synthesis using Cellular Encoding and the Genetic Algorithm*. Ph. D. Thesis. Univ. Claude Bernard-Lyon I.

[GW93] Gordon V. S., Whitley D. (1993). "Serial and Parallel Genetic Algorithms as Function Optimizers". In [Forr93], 177-183.

[HBBK97] Hart W. E., Baden S., Belew R. K., Kohn S. (1997). "Analysis of the Numerical Effects of Parallelism on a Parallel Genetic Algorithm". *Proceedings of the Workshop on Solving Combinatorial Optimization Problems in Parallel*. IEEE (ed.). CD-ROM IPPS97.

[HL97] Herrera F., Lozano M. (1997). "Gradual Distributed Real-Coded Genetic Algorithms". Technical Report #DECSAI-97-01-03 (February 97).

[HL98] Herrera F., Lozano M., Moraga C. (1998). "Hybrid Distributed Read-Coded Genetic Algorithms". In Eiben A. E., Bäck T., Schoenauer M., Schwefel H.-P. (eds.), *Proceedings of the Parallel Problem Solving From Nature (PPSN) V International Conference*, Amsterdam. Springer-Verlag, 603-612.

[HLV95] Herrera F., Lozano M., Verdegay J. L. (1995). "Tackling Fuzzy Genetic Algorithms". *Genetic Algorithms in Engineering and Computer Science*, Winter G., Périaux J., Galán M., Cuesta P. (eds.). John Wiley & Sons, 167-189.

[HMP96] Hinterding R., Michalewicz Z., Peachy T. C. (1996). "Self-Adaptive Genetic Algorithm for Numeric Functions". In [VERS96], 420-429.

[Hoff91] Hoffmeister F. (1991). "The User's Guide to $ESC^{A}P_{A}DE$ 1.2 A Runtime Environment for Evolution Strategies". *Department of Computer Science*, University of Dortmund, GE.

[Holl75] Holland J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

[Hugh89] Hughes M. (1989). "Genetic Algorithm Workbench Documentation", Cambridge Consultants Ltd.

[Hunt95] Hunter A. (1995). "Sugal Programming Manual v2.0". *T. R.* in Univ. Sunderland, U.K.

[Ingb93] Ingber L. (1993). "Simulated Annealing: Practice versus Theory". *Mathl. Comput. Modelling*, 18 (11), 29-57.

[Jela97] Jelasity M. (1997). "A Wave Analysis of the Subset Sum Problem". In [Bäck97a], 89-96.

[JM91] Janikow C. Z., Michalewicz Z. (1991). "An Experimental Comparison of Binary and Floating Point Representation in GAs". In [BB91], 31-36.

[Jusl95] Juslstrom B. A. (1995). "What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm". [Eshe95], 81-87.

[KBH93] Khuri S., Bäck T. Heitkötter J. (1993). "An Evolutionary Approach to Combinatorial Optimization Problems". Proceedings of CSC'93.

[KGFR96] Koza J. R., Goldberg D. E., Fogel D. B., Riolo R. L. (eds) (1996). *Proceedings of the First Annual Conference on Genetic Programming*. The MIT Press.

[KH96] Kuo T., Hwang S. Y. (1996). "A Genetic Algorithm with Disruptive Selection". *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*. 26(2), 299-307.

[KH97] Kuo T., Hwang S. Y. (1997). "Using Disruptive Selection to Maintain Diversity in Genetic Algorithms". *Applied Intelligence*, Kluwer Academic Press, 7, 257-267.

[Kinn94] Kinnear K. E. (1994). *Advances in Genetic Programming*. MIT Press, Cambridge MA.

[Koza92] Koza J. R. (1992). *Genetic Programming*. Bradford Book, The MIT Press.

[Koza94] Koza J. R. (1994). *Genetic Programming II*. The MIT Press.

[Kung93] Kung S. Y. (1993). *Digital Neural Networks*. Prentice Hall.

[Kurs92] Kursawe F. (1992). "Evolution Strategies for Vector Optimization". *Preliminary Proceedings of the Tenth International Conference on Multiple Criteria Decision Making*, Tzeng G. H., Yu P. L. (eds.), 187-193. National Chiao Tung University, Taipei.

[Levi94] Levine D. (1994). "A Parallel Genetic Algorithm fot the Set Partitioning Problem". *T. R. No. ANL-94/23*, Argonne National Laboratory, Mathematics and Computer Science Division.

[Levi96] Levine D. (1996). "Users Guide to the PGAPack Parallel Genetic Algorithm Library". *T. R. ANL-95/18*, January 31.

[Levi97] Levine D. (1997). "Genetic Algorithms: A Practitioner's View". *INFORMS Journal of Computing*, 9(3), 256-259.

[LK89] Lucasius C. B., Kateman G. (1989). "Applications of Genetic Algorithms in Cheometrics". In [Scha89], 170-176.

[Loza96] Lozano M. (1996). *Aplicación de Técnicas Basadas en Lógica Difusa para Mejorar el Comportamiento de los Algoritmos Genéticos con Codificación en Coma Flotante*. Tesis Doctoral, Universidad de Granada.

[LPG94] Lin S. C., Punch III W. F., Goodman E. D. (1994). "Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach". *Proceedings of the Sixth IEEE Parallel & Distributed Processing*, 28-37.

[LR93] Louis S. L., Rawlins G. J. E. (1993). "Predicting Convergence Time for Genetic Algorithms". *Technical Report* (20 pages), January.

[Manj96] Manjón-Mostazo F. (1996). *Desarrollo de una Capa de Servicios de Comunicación para la Paralelización Transparente de Estructuras de Datos en Algoritmos Genéticos*. PFC156, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[Mare94] Maresky J. (1994). *On Efficient Communication in Distributed Genetic Algorithms*. M. S. Dissertation, Institute of Computer Science, The Hebrew University of Jerusalem.

[MC94] Mejía-Olvera M., Cantú-Paz E. (1994). "DGENESIS-Software for the Execution of Distributed Genetic Algorithms". *Proceedings of the XX Conferencia Latinoamericana de Informática,* 935-946.

[ME90] Macfarlane D., East I. (1990). "An investigation of several Parallel genetic algorithms". *Univ.of Buckingham*, MK 18 IEG, 60-67.

[MHK93] Maruyama T., Hirose T., Konagaya A. (1993). "A Fine-Grained Parallel Genetic Algorithm for Distributed Parallel Systems". In [Forr93], 184-190.

[Mich92] Michalewicz Z.(1992). *Genetic Algorithms+Data Structures=Evolution Programs.* Springer-Verlag.

[MJ91] Michalewicz Z., Janikow C. Z. (1991). "Handling Constraints in Genetic Algorithms". In [BB91], 151-157.

[MP96] Merelo J. J., Prieto A. (1996). GAGS, "A Flexible Object-Oriented Library for Evolutionary Computation". *Proceedings of MALFO*, Borrajo D., Isasi P. (eds.), 99-105.

[MS89] Manderick B., Spiessens P. (1989). "Fine-Grained Parallel Genetic Algorithms". In [Scha89], 428-433.

[MS93] Mühlenbein H., Schlierkamp-Voosen D. (1993). "Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization". *Evolutionary Computation* 1(1) 25-49, MIT Press.

[MSB91] Mühlenbein H., Schomisch M., Born J. (1991). The Parallel Genetic Algorithm as Function Optimizer. *Parallel Computing* 17, 619-632.

[MT91] Muntean T., Talbi E. G. (1991). "A Parallel Genetic Algorithm for Process-Processors Mapping". *Proceedings of the Second Symposium II. High Performance Computing*, Durán M., Dabaghi E. (eds.), 71-82. Montpellier, France: F. Amsterdam, Amsterdam.

[MTH89] Miller G. F., Todd P. M., Hegde S. U. (1989). "Designing Neural Networks using GAs". In [Scha89], 379-384.

[MTS93] Munetomo M., Takai Y., Sato Y. (1993). "An Efficient Migration Scheme for Subpopulation-Based Asynchronously Parallel GAs". *HIER-IS-9301*, Hokkaido University.

[Mühl91] Mühlenbein H. (1991). "Evolution in Time and Space - The Parallel Genetic Algorithm". *Foundations of Genetic Algorithms*, Rawlins G. J. E. (ed.), Morgan Kaufmann, 316-337.

[Muño98] Muñoz-López, A. (1998). *Diseño y Evaluación de Algoritmos Genéticos Paralelos en JAVA*. PFC248, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[NASA91] NASA - Johnson Space Center (1991). "Splicer - A Genetic Tool for Search and Optimization". *Genetic Algorithm Digest, Vol 5, Issue 17*.

[NDV97] Naudts B., Suys D., Verschoren A. (1997). "Epistasis as a Basic Concept in Formal Landscape Analysis". In [Bäck97a], 65-72.

[OHE96] Orfali R., Harkey D., Edwards J. (1996). *The Essential of Distributed Objects*. Wiley.

[OH97] Orfali R., Harkey, D. (1997). *Client/Server Programming with Java and CORBA*. Wiley Computer Publishing.

[Pedr96] Pedroso J. P. (1996). "Niche Search: An Evolutionary Algorithm for Global Optimisation". In [VERS96], 430-440.

[Peti97] Petit i Silvestre J. (1997). "PgaFrame: A Frame for Parallel Genetic Algorithms". http://www.lsi.upc.es/~jpetit/PgaFrame.

[Pew96] Pew J. A. (1996). *Instant JAVA*. The Sunsoft Press - Prentice Hall.

[PGY94] Potts J. C., Giddens T. D., Yadav S. B. (1994). "The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection". *IEEE Transactions on Systems, Man, and Cybernetics*, 24 (1), 73-86.

[PL89] Pettey C. C., Leuze M. R. (1989). "A Theoretical Investigation of a Parallel Genetic Algorithm". In [Scha89], 398-405.

[PLG87] Pettey C. C., Leuze M. R., Grefenstette J. (1987). "A Parallel Genetic Algorithm". In [Gref87], 155-161.

[PS82] Papadimitriou C. H., Steiglitz K. (1982). *Combinatorial Optimization*. Prentice Hall.

[PSA95] Pearson D. W., Steele N. C., Albretch R. F. (eds.) (1995). *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*. Springer-Verlag.

[Radc92] Radcliffe N. J. (1992). "Non-Linear Genetic Representations". *Parallel Problem Solving from Nature 2*, Elsevier Science Publishers, 259-268.

[RAT93] Ribeiro Filho J. L., Alippi C., Treleaven P. (1993). "Genetic Algorithm Programming Environments". *Parallel Genetic Algorithms: Theory & Applications*, J. Stender (ed.), IOS Press.

[Rech73] Rechenberg I. (1973). *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution*. Fromman-Holzboog Verlag, Sttutg.

[Reev93a] Reeves C. R. (ed.) (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, Oxford.

[Reev93b] Reeves C. R. (1993). "Using Genetic Algorithms with Small Populations". In [Forr93], 92-99.

[RM89] Rumelhart D. E., McClelland J. L. (1989). *Parallel Distributed Processing, Vol I*. The MIT Press.

[Robb92] Robbins G. (1992). "EnGENEer - The Evolution of Solutions". *Proceedings of the 5th Annual Seminar on Neural Networks and Genetic Algorithms*.

[Roma93] Romaniuk S. G. (1993). "Evolutionary Growth Perceptrons". In [Forr93], 334-341.

[RP97] Römke T., Petit i Silvestre J. (1997). "Programming Frames for the Efficient Use of Parallel Systems". *LSI-97-9-R*, Universidad Politécnica de Barcelona.

[RS94] Radcliffe N. J., Surry P. D. (1994). "The Reproductive Plan Language RPL2: Motivation, Architecture and Applications". *Genetic Algorithms in Optimisation, Simulation and Modelling*, Stender J., Hillebrand E., Kingdon J. (eds.). IOS Press.

[Rubi96] Rubio-delRío A. (1996). *Generación y Codificación de Imágenes Usando el Paradigma de la Programación Genética*. PFC162, Dpto. Lenguajes y CC.CC., Univ. Málaga.

[Rumb91] Rumbaugh J. (1991). *Object-Oriented Modeling and Design*. Prentice-Hall.

[Salo96] Salomon R. (1996). "The Influence of Different Coding Schemes on the Computational Complexity of Genetic Algorithms in Function Optimization". In [VERS96], 227-235.

[Scha89] Schaffer J. D. (ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, California.

[Schw81] Schwefel H. P. (1981). *Numerical Optimization of Computer Models*. Wiley, Chichester.

[Schw95] Schwefel H. P. (1995). "Evolution and Optimum Seeking". *Sixth-Generation Computer Technology Series*, Wiley New York.

[SD91] Spears W. M., De Jong K. A. (1991). "An Analysis of Multi-Point Crossover". *Foundations of Genetic Algorithms*, Rawlins G. J. E. (ed.), Morgan Kaufmann, 301-315.

[SD96] Sarma J., De Jong K. (1996). "An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms". In [VERS96], 236-244.

[SD97] Sarma J. DeJong K. (1997). "An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm". In [Bäck97a], 181-186.

[SG91] Schraudolp N. N., Grefenstette J. J. (1991). "A User's Guide to GAUCSD 1.2". *T. R. Computer Science & Engineering Department*, University of California, San Diego.

[Shon93] Shonkwiler R. (1993). "Parallel Genetic Algorithms". In [Forr93], 199-205.

[SM91] Spiessens P., Manderick B. (1991). "A Massively Parallel Genetic Algorithm". In [BB91], 279-286.

[Spea92] Spears W. M. (1992). "Crossover or Mutation?". *Proceedings of the Foundations of Genetic Algorithms Workshop*, D. Whitley (ed.), Morgan Kaufmann, 221-237.

[Sten93] Stender J. (ed.) (1993). *Parallel Genetic Algorithms: Theory and Applications*. IOS Press.

[Stro91] Strouptrup B. (1991). *The C++ Programming Language* (2nd Edition). Addison-Wesley.

[Sysw89] Syswerda G. (1989). "Uniform Crossover in Genetic Algorithms". In [Scha89], 2-9.

[Sysw91] Syswerda G. (1991). "A Study of Reproduction in Generational and Steady-State Genetic Algorithms". *Foundations of GAs* 1, Rawlins G. J. E. (ed.), Morgan Kaufmann, 94-101.

[TA91] Troya . J. M., Aldana J. F. (1991). "Extending an Object Oriented Concurrent Logic Language for Neural Network Simulations". *Proceedings of the IWANN'91, LNCS*, Springer-Verlag, 235-242.

[Take92] Takefuji, Y. (1992): *Neural Network Parallel Computing.* Kluwer Academic Publishers.

[Tane89] Tanese R. (1989). "Distributed Genetic Algorithms". In [Scha89], 434-439.

[Thie97] Thierens D. (1997). "Selection Schemes, Elitist Recombination, and Selection Intensity". In [Bäck97a], 152-159.

[Toma93] Tomassini M. (1993). "The Parallel Genetic Cellular Automata: Application to Global Function Optimization". In [ARS93], 385-391.

[TS93] Tate D. M., Smith A. E. (1993). "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms". In [Forr93], 31-37.

[TZ89] Törn A., Zilinskas A. (1989). *Global Optimization*, Vol. 350, LNCS, Springer.

[VBS93] Voigt H.-M., Born J., Santibáñez-Koref I. (1993). "Multievaluated Evolutionary Algorithms". In [Forr93], 657.

[VBT91] Voigt H. M., Born J., Treptow J. (1991). "The Evolution Machine Manual - V 2.1". *T. R.* in the Institute for Informatics and Computing Techniques, Berlin.

[VERS96] Voigt H. M., Ebeling W. Rechengerg I., Schwefel H. P. (eds.) (1996). *Proceedings of the Fourth Parallel Problem Solving from Nature*. Springer-Verlag.

[VSB92] Voigt H. M., Santibáñez-Koref I., Born J. (1992). "Hierarchically Structured Distributed Genetic Algorithms". *Proceedings of the International Conference Parallel Problem Solving from Nature, 2*, Männer R., Manderick B. (eds.), North-Holland, Amsterdan, 155-164.

[Wall95] Wall M. (1995). "Overview of Matthew's Genetic Algorithm Library". http://lancet.mit.edu/ga.

[WC97] Wah B. W., Chang Y-J. (1997). "Trace-Based Methods for Solving Nonlinear Global Optimization and Satisfiability Problems". *Journal of Global Optimization*, Kluwer Academic Press, 10(2), 107-141.

[WH89] Whitley D., Hanson T. (1989). "Optimizing Neural Networks Using Faster, More Accurate Genetic Search". In [Scha89], 391-396.

[Whit89] Whitley D. (1989). "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". In [Scha89], 116-121.

[Whit93a] Whitley D. (1993). *A Genetic Algorithm Tutorial. T.R. CS-93-103 (revised version)*, Department of Computer Science, Colorado State University (November 10).

[Whit93b] Whitley D. (1993). "Cellular Genetic Algorithms". In [Forr93], 658.

[WM97] Wolpert D. H., Macready W. G. (1997). "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation* 1(1), 67-82.

[Wrig91] Wright A. (1991). "Genetic Algorithms for Real Parameter Optimization". *Foundations of Genetic Algorithms* 1, Morgan Kaufmann, 205-218.

[WS90] Whitley D., Starkweather T. (1990). "GENITOR II: a Distributed Genetic Algorithm". *J. Expt. theor. Artif. Intelligence* 2, 189-214.

[WS92] Whitley D., Schaffer J. D. (eds.) (1992). *Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE Computer Society Press.

[WSB90] Whitley D., Starkweather T., Bogart C. (1990). "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity". *Parallel Computing* 14 , 347-361.

[YHK97] Yang J.M., Horng J.T., Kao, C.Y. (1997). "A Continuous Genetic Algorithm for Global Optimization". In [Bäck97a], 230-237.

[YG93] Yin X., Germay N. (1993). "Improving Genetic Algorithms with Sharing through Cluster Analysis". In [Forr93], 100.

[ZK93] Zeigler B. P., Kim J. (1993). "Asynchronous Genetic Algorithms on Parallel Computers". In [Forr93], 660.