Implementación del algoritmo UEGO sobre el entorno Matlab como alternativa al toolbox de optimización

Victoria Plaza Leiva
Universidad de Almería

***Abstract*— Global optimization algorithms are widely used in the scientific sector as a tool to support their own research. Matlab optimization toolbox is often used because of its ease of use and configuration and its full integration into the powerful mathematical environment. In this paper, we implement the evolutionary algorithm UEGO for Matlab as an alternative to the toolbox's algorithms. Specifically, the UEGO solution improves the one provides by Matlab genetic algorithm because it avoids be locked in local optima.**

***Resumen*— Los algoritmos de optimización global son muy usados en el sector científico como herramienta de apoyo a su investigación particular. A menudo se utiliza el toolbox de optimización de Matlab debido a su sencillez de uso y configuración y a su plena integración en el potente entorno de trabajo matemático. En este trabajo se realiza la implementación del algoritmo evolutivo UEGO para Matlab como alternativa a los algoritmos de dicho toolbox. En concreto UEGO mejora la solución del algoritmo genético de Matlab ya que evita bloquearse en óptimos locales.**

***Keywords*—Optimización global, metaheurísticas, algoritmos evolutivos, toolboox, Matlab.**

I. Introducción

LA OPTIMIZACIÓN Global es un campo de investigación que incluye teoría, métodos y aplicaciones de las técnicas de optimización, y cuyo objetivo es encontrar el óptimo global en problemas de optimización matemática donde existen muchos óptimos locales que no son óptimos globales.

La Optimización Global (OG) puede aplicarse a un amplio conjunto de problemas del mundo real. Entre tales aplicaciones se encuentran: economía de escala, cargas fijas, finanzas, estadística, optimización estructural, diseño de ingeniería, problemas de redes y transporte, problemas de diseño de chips, diseño nuclear y mecánico, diseño y control en ingeniería química, biología molecular, y otras muchas.

Formalmente, la Optimización Global busca la solución global de un modelo de optimización con restricciones. La formulación de un problema de optimización global, en forma de maximización, viene dado por:

$$\max f(x), \text{ sujeto a } x \in X \qquad (1)$$

donde X es un conjunto no vacío en $\Re^n$ y f es una función continua. Por lo tanto, el objetivo es encontrar el valor máximo f^* y todos los puntos $x^* \in X$ tales que:

$$f^* = f(x^*) \geq f(x), \ \forall x \in X \qquad (2)$$

La conversión de un problema de minimización a uno de maximización es sencilla:

$$\min f(x^*) \mid x \in X = -\max - f(x) \mid x \in X \qquad (3)$$

entonces, minimizar f(x) es equivalente a maximizar *-f(x).*

En este trabajo, el problema de optimización será tratado como problema de maximización.

En general, las técnicas clásicas de optimización tienen dificultades para tratar problemas de optimización global. Una de las principales razones por las que fallan es que pueden quedar fácilmente atrapados en un óptimo local. Además, las técnicas de optimización local no pueden generar, ni tampoco usar, la información global necesaria para encontrar el óptimo global de una función con múltiples óptimos locales. El campo de la optimización global estudia problemas del tipo (1), los cuales tienen uno o muchos óptimos globales.

Este trabajo presenta el algoritmo UEGO (Universal Global Optimization Algorithm) como método de optimización global para abordar problemas del tipo (1). La eficiencia y eficacia del método se estudiará mediante un exhaustivo estudio computacional, donde se compararán los resultados hallados por el mismo con los obtenidos mediante el toolbox de Matlab. El toolbox de OG, llamado *optimtool*, es un módulo software que se integra en el entorno Matlab y ofrece distintos algoritmos de resolución de problemas de optimización, entre los que se encuentra un algoritmo genético. Los algoritmos del toolbox a menudo se quedan atrapados en óptimos locales por lo que no devuelven como

Trabajo Final de Máster entregado en Junio 2012. Máster de Informática y Automática Industrial de la Universiad de Almería.

Victoria Plaza Leiva, Departamento de Arquitectura de Computadores y Electrónica, Universidad de Almería (e-mail: plv834@ual.es)

solución el óptimo global buscado. Se plantea UEGO como una alternativa a los algoritmos de dicho toolbox que ofrece dos ventajas principales: (a) tiene mecanismos que le permiten escapar de óptimos locales y (b) además puede devolver distintos puntos como soluciones globales y/o locales. Tal y como se verá posteriormente, los resultados obtenidos por UEGO son prometedores, encontrando el óptimo global en el 100% de los casos, en contraposición a los algoritmos de optimización que ofrece la herramienta optimtool de Matlab.

El resto del trabajo se planteará como sigue. La Sección II describirá brevemente el estado del arte en técnicas de optimización global, la Sección III se centrará en el algoritmo UEGO, su funcionamiento y configuración de parámetros. La Sección IV describirá el toolbox de optimización de Matlab y el funcionamiento de su algoritmo genético. En la Sección V se detallará el estudio computacional realizado con ambos algoritmos y sus resultados, y la última sección se destinará a las conclusiones y trabajos futuros.

II. TÉCNICAS DE OPTIMIZACIÓN GLOBAL

Existen muchas clasificaciones para las estrategias de optimización global. Un ejemplo lo podemos encontrar en [42], donde las estrategias se dividen, de acuerdo al grado de rigor con el que se aproximan al objetivo, en:

1) *Métodos incompletos*, donde se usa una heurística ingeniosa e intuitiva, pero no se garantiza que pueda escapar de un óptimo local.
2) *Métodos completos asintóticos*. La probabilidad de obtener un óptimo global es uno si se ejecutan indefinidamente. Sin embargo, no podemos asegurar que se alcance el óptimo global.
3) *Métodos completos*. Pueden alcanzar el óptimo global con certeza cuando se usan métodos exactos, y éstos se ejecutan indefinidamente. También pueden encontrar una aproximación al óptimo global, con una tolerancia determinada después de un tiempo finito.
4) *Métodos rigurosos*, donde el óptimo global se alcanza con certeza, dentro de una tolerancia dada, incluso con cálculos aproximados.

Para otras posibles clasificaciones de los métodos de optimización global véase [47], [52]. En la actualidad, es muy difícil proponer una clasificación de los algoritmos de búsqueda, ya que, normalmente, estos mezclan varias técnicas para obtener el óptimo global. En este proyecto nos vamos a centrar en la clasificación propuesta en [48], la cual se detalla en la Figura 1, y se explicará brevemente en la siguiente sección.

Inicialmente, los métodos de optimización global se dividen en *exactos* y *heurísticos*, dependiendo de si pueden garantizar o no que la solución óptima sea encontrada.

En los métodos exactos o *determinísticos* (tales como la optimización Lipschitzian o Branch and Bound), no se

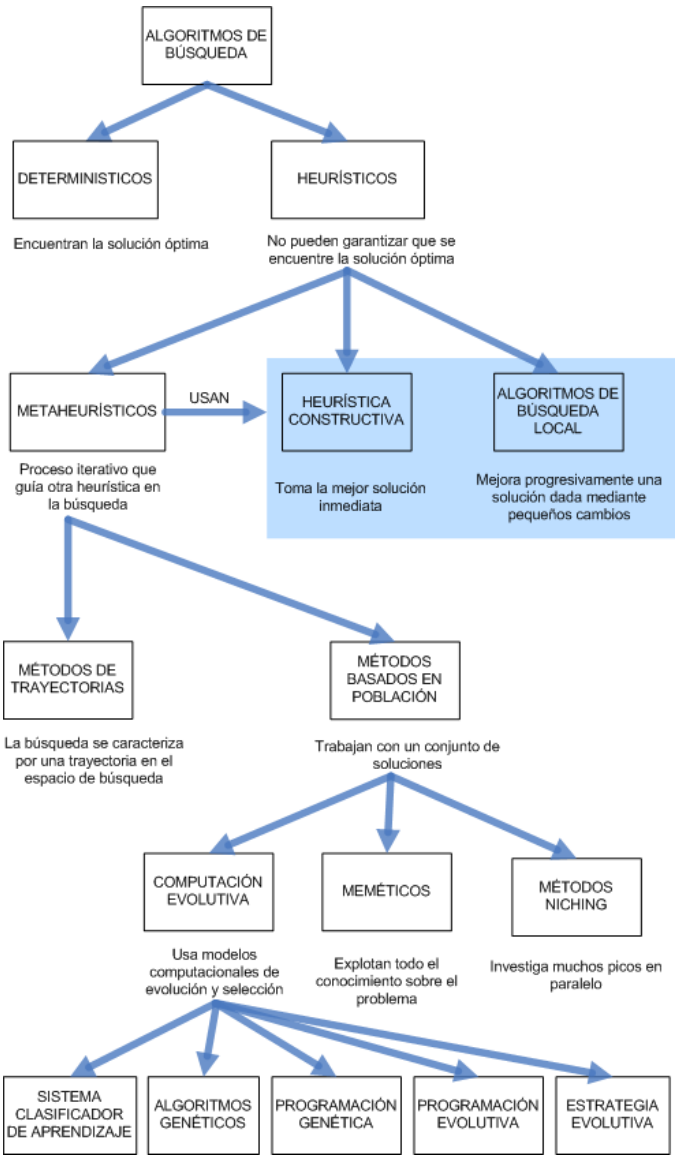

Figura 1. Taxonomía de los Algoritmos de búsqueda

incluyen factores aleatorios y se intenta encontrar, si existe, el óptimo global de forma rigurosa, es decir, explorando todo el espacio de búsqueda. Algunos de ellos convergen al óptimo global bajo algunas condiciones, pero cuando el algoritmo se detiene, después de un número finito de iteraciones, puede que no se conozca con exactitud la precisión de la solución.

Los métodos exactos, que terminan en un número finito de pasos, requieren acceso detallado a la información global del problema a resolver. Los métodos completos son más eficientes, y generalmente, combinan técnicas de bifurcación con una o varias técnicas de optimización local, análisis convexo, análisis de intervalos o programación con restricciones.

Además, los algoritmos exactos garantizan encontrar una solución óptima, para toda instancia finita de un problema. No obstante, a efectos prácticos, son muy costosos desde un punto de vista computacional. Por este motivo, los métodos heurísticos han recibido más atención en los últimos años,

sacrificando la garantía de encontrar la solución óptima por el bien de obtener una buena solución en un corto periodo de tiempo.

La *heurística* proporciona una solución útil y factible para un amplio rango de dominios de problemas y aplicaciones. Su poder proviene de su habilidad para tratar problemas complejos cuando se tiene un conocimiento pequeño o escaso del espacio de búsqueda. Son particularmente adecuados para aplicaciones de toma de decisiones y computacionalmente intratables.

Los *métodos heurísticos* o *estocásticos* pueden ser considerados como algoritmos que realizan una búsqueda aleatoria parcial de soluciones factibles, óptimas o cercanas, para un problema hasta que se cumple una condición particular de finalización o se alcance un número de iteraciones predefinidas. Los algoritmos heurísticos no pueden garantizar que el conjunto de óptimos encontrados incluye el óptimo global. Normalmente, estos algoritmos pueden encontrar una solución cuyo valor objetivo está cerca del óptimo global y lo hacen de forma rápida y fácil. Algunas veces estos algoritmos pueden ser exactos, lo que significa que encuentran la mejor solución, pero siguen siendo heurísticos, no pudiendo demostrar que sea la mejor solución.

Normalmente, las heurísticas imitan muy bien las estrategias presentes en la naturaleza. Por ejemplo, las técnicas evolutivas copian los principios aplicados a las especies para desarrollar generaciones de calidad superior; el enfriamiento simulado está basado en cómo los cristales se forman cuando los materiales se enfrían y las partículas "encuentran" una estructura que minimiza el balance de energía; etc.

En las siguientes secciones se describe la rama heurística de la Figura 1, ya que este proyecto se va a centrar en algoritmos de tipo heurístico. Dentro de los métodos heurísticos se estudiarán los métodos estocásticos basados en análisis de poblaciones como son los algoritmos genéticos o los algoritmos evolutivos como UEGO.

II.A. Algoritmos Heurísticos

En la ciencia de la computación hay dos objetivos fundamentales que los algoritmos persiguen: una buena solución (o la solución óptima), y un buen tiempo de ejecución. Una heurística es un algoritmo que abandona uno o ambos objetivos; por ejemplo, normalmente encuentran buenas soluciones, pero no hay pruebas de que la solución pueda ser arbitrariamente errónea en algunos casos; o se ejecuta razonablemente rápido, aunque no existe tampoco ninguna prueba de que siempre será así.

Puede que no estemos interesados en obtener la solución óptima del problema porque el tamaño del problema va más allá de los límites computacionales de los algoritmos conocidos; también podríamos querer resolver el problema pero realmente no merece la pena por el esfuerzo requerido, gasto de tiempo, dinero, etc, que puede suponer. En estos casos podemos utilizar una heurística que ayude a encontrar una solución factible, cercana al óptimo. La principal ventaja de los algoritmos heurísticos es que a menudo son conceptualmente sencillos y menos costosos de implementar que los algoritmos determinísticos.

Entre los algoritmos heurísticos es posible distinguir tres métodos diferentes: métodos heurísticos constructivos, métodos de búsqueda local y metaheurísticas.

II.A.1) Heurística constructiva

Los métodos heurísticos constructivos son algoritmos que generan la solución a partir de un conjunto inicial vacío, seleccionando un elemento de un conjunto de soluciones factibles. El proceso se repite hasta que se encuentra una solución completa. El ejemplo más simple de este tipo de métodos es el que se muestra a continuación:

La *Heurística Greedy* es un método que toma las decisiones en base a la información actual sin tener en cuenta las posibles consecuencias de dichas decisiones en un futuro. Es un método fácil de implementar y muy eficiente computacionalmente hablando. Los algoritmos Greedy funcionan bastante bien cuando el problema a resolver se puede dividir en pequeños subproblemas, y cuando dichos subproblemas contienen una solución óptima.

Sin embargo, pueden fallar en la búsqueda del óptimo global ya que no realizan una búsqueda exhaustiva de todos los datos. El algoritmo de Prim [46], que encuentra el árbol de expansión mínimo, así como el algoritmo de Dijkstra [14], que encuentra el camino más corto desde un origen a todos los nodos del árbol, son ejemplos muy conocidos de este tipo de heurística.

II.A.2) Métodos de búsqueda local

Un método de búsqueda local se basa en la idea de que una solución S dada puede ser mejorada realizando pequeños cambios. A las soluciones obtenidas por la modificación de S, se les llama *vecinos* de *S* y la aplicación del operador que produce un vecino *S'* se le llama *movimiento*.

Un algoritmo de búsqueda local empieza con una solución inicial y realiza movimientos de vecino a vecino incrementando el valor de la función objetivo. La principal desventaja de esta estrategia es la dificultad que presenta para escapar de un óptimo local en el que no puede encontrar ninguna solución que mejore el valor actual de la función objetivo.

El método más simple que hace uso del procedimiento de búsqueda local es el método Multistart. La mayoría de estos métodos se pueden ver como variaciones del algoritmo Multistart. Para una revisión más extensa de los métodos de dos fases, se pueden consultar las referencias de Rinnooy Kan y Timmer [27], [28].

II.B. Algoritmos Metaheurísticos

Los algoritmos metaheurísticos pueden englobarse dentro

de los algoritmos heurísticos. No obstante, y puesto que este proyecto se va a centrar en este tipo de algoritmos, se les va a destinar una sección.

Muchos algoritmos muy conocidos son considerados metaheurísticos: Optimización aleatoria, Búsqueda local, Búsqueda primero el mejor, Algoritmos genéticos, Enfriamiento simulado, Búsqueda tabú, por nombrar algunos. Las metaheurísticas son un campo activo, muy presente en la literatura, con un gran número de investigadores y de usuarios, así como multitud de aplicaciones.

Una metaheurística puede definirse como un algoritmo de alto nivel que está especializado en resolver problemas de optimización. Consiste en un proceso iterativo, que guía la búsqueda de una solución factible. Las metaheurísticas son aproximaciones, y por lo general, se aplican a problemas para los cuales no hay un algoritmo o una heurística específica que dé una solución satisfactoria, o cuando no es posible implementar dicho método.

Una buena metaheurística tiene que proporcionar un balance entre la explotación de la experiencia acumulada en la búsqueda y la exploración del espacio de búsqueda para identificar regiones con soluciones de alta calidad. El balance entre la explotación y la exploración es muy importante, por un lado para identificar rápidamente regiones del espacio de búsqueda con buenas soluciones y por otro lado, para no desaprovechar el tiempo en zonas que ya han sido exploradas o que no tienen soluciones buenas [7]. La principal diferencia entre las metaheurísticas existentes es la manera particular en la que buscan dicho balance. Las aproximaciones de las diferentes metaheurísticas se caracterizan por los diferentes aspectos que conciernen al camino de búsqueda que siguen, como el uso de poblaciones de soluciones, el número de vecinos considerados, etc. En [6] se presenta una discusión al respecto.

A continuación se describen algunas de las metaheurísticas más importantes basadas en trayectorias y estrategias basadas en poblaciones.

II.B.1) Métodos de trayectorias

Este tipo de metaheurísticas se caracteriza porque el proceso de búsqueda trata de seguir una trayectoria en el espacio. Podemos distinguir entre dos metaheurísticas: la primera sigue la trayectoria de búsqueda correspondiente a un camino cercano en el grafo, y otra permite dar grandes saltos en el grafo. Estos tipos se conocen como *enfriamiento simulado* y *búsqueda tabú*, respectivamente.

Enfriamiento simulado (Simulated Annealing, SA)

El Enfriamiento simulado (SA) es un método probabilístico que se ha usado en muchos problemas (véase [2], [9], [19], [25], [29]). Sin embargo, actualmente se usa como un componente dentro de los algoritmos metaheurísticos, más que como algoritmo de búsqueda estándar [17], [18].

El término enfriamiento (o temple) simulado procede del campo de la experimentación en física. Temple es un proceso físico en el que un cristal en estado líquido es enfriado pasando a convertirse en un cristal (estado sólido). Si el enfriamiento se hace lo suficientemente suave, el estado de energía del sólido al final del enfriamiento es el mínimo (o muy próximo a él) en el que dicho cristal puede estar.

La simulación del enfriamiento físico se puede realizar con el algoritmo *Metrópolis*, que fue ideado en las primeras etapas de la computación [34]. Este algoritmo genera secuencias de estados del sólido mediante el siguiente procedimiento: Dado un estado actual i del sólido con energía Ei, entonces el siguiente estado j se generará aplicando una pequeña perturbación al sólido, por ejemplo, moviendo una partícula. La energía del siguiente estado es Ej. Si la diferencia de energías Ei – Ej es menor o igual a cero, el estado j es aceptado como nuevo estado actual. En otro caso, el estado j se acepta con una probabilidad dada por:

$$\exp\left(-\frac{E_j - E_i}{k_b T}\right) \qquad (4)$$

donde T es la temperatura del sólido y kB es la constante de Boltzmann.

Si la temperatura se disminuye suficientemente despacio, el sólido alcanza el equilibrio térmico a cada temperatura. En el algoritmo Metrópolis este equilibrio se alcanza aplicando un número suficientemente grande de perturbaciones a cada temperatura.

El algoritmo de SA se puede ver como un proceso análogo a la simulación del enfriamiento de un sólido del algoritmo Metrópolis. La analogía entre SA (algoritmo de optimización) y el modelo físico del enfriamiento de un sólido se representa en la TABLA I [43].

Puede demostrarse que el algoritmo SA converge asintóticamente al conjunto de soluciones globalmente óptimas si se cumplen ciertas condiciones [1]. Desafortunadamente, para que la probabilidad de encontrar un óptimo global sea alta, se requieren esquemas de disminución de temperatura muy lentos y un gran número de iteraciones a cada temperatura. De hecho, SA puede ser mucho más costoso que si se hiciera una búsqueda exhaustiva en el espacio de soluciones. De todos modos, en la práctica, el algoritmo SA encuentra óptimos locales muy buenos con una carga computacional razonable.

TABLA I
ANALOGÍAS ENTRE EL MODELO FÍSICO DE ENFRIAMIENTO DE UN SÓLIDO Y EL ALGORITMO SA

Sistema físico	Problema de Optimización
Estado	Posible solución
Energía	Costo
Estado mínimo	Solución óptima
Enfriamiento rápido	Búsqueda Local
Enfriamiento delicado	Temple simulado

Búsqueda tabú

La búsqueda Tabú (BT) [21], [22] es un procedimiento adaptativo para resolver principalmente problemas de optimización, que dirige una heurística de descenso de colinas para que continúe la exploración sin confundirse por la ausencia de movimientos mejorados, y caracterizada por no caer en un óptimo local del cual previamente ha salido. En cada iteración, se aplica un movimiento admisible para la solución actual, transformándola en el vecino con el menor costo. También se permiten los movimientos hacia una nueva solución que incrementen la función costo. En ese caso, el movimiento contrario debería prohibirse a lo largo de algunas iteraciones para evitar ciclos. Estas restricciones están basadas en el mantenimiento de una función de memoria a corto plazo (lista tabú), la cual determina cuanto tiempo será forzada una restricción tabú, o alternativamente, qué movimientos son admisibles en cada iteración.

Los métodos de búsqueda tabú operan bajo el axioma de que puede construirse una vecindad para identificar soluciones adyacentes que pueden alcanzarse desde cualquier solución actual (búsqueda en la vecindad). Con cada intercambio hay asociado un valor de movimiento, que representa el cambio en el valor de la función objetivo como resultado del intercambio propuesto.

En la búsqueda del vecindario, cada solución x tiene asociado un conjunto de vecinos, $N(x) \subset X$, llamado la vecindad de x. Cada solución $x' \in N(x)$ puede alcanzarse directamente a partir de x cuando se aplica tal operación. Normalmente, en la búsqueda tabú, las vecindades se suponen simétricas, es decir, x' es un vecino de x si y sólo si x es un vecino de x'.

La función de memoria a corto plazo (lista tabú) se utiliza para indicar los movimientos que están prohibidos durante un tiempo determinado por el periodo tabú (t_l). De modo que una lista tabú guarda las t_l últimas soluciones que se han obtenido y que no se pueden repetir.

La lista tabú T podría implementarse como un conjunto de pares de soluciones (x', x), donde x es el punto de partida y x' la solución a la que se llegó. De modo que cada una de estas parejas indican que se produjo algún movimiento desde la solución x hasta la solución x' durante los últimos t_l movimientos o transiciones. Normalmente en la lista tabú se guardan las inversas de dichos movimientos.

Nótese que un movimiento es una función de un subconjunto del espacio de soluciones que se proyecta en el espacio de soluciones. En la práctica, los movimientos deben ser identificados de algún modo.

La lista Tabú a menudo restringe demasiado la búsqueda. Puede ocurrir que solo haya unos pocos movimientos en el problema, pero cada movimiento puede aplicarse a un gran número de soluciones, dando lugar a un gran número de soluciones vecinas igualmente probables. Este hecho lleva a la introducción de una *relajación* adicional a la condición de la lista tabú, llamada función *nivel de aspiración A(d, x)*. Los argumentos de entrada de esta función de nivel son: un movimiento y una solución y cuya salida es un valor expresado en las mismas unidades que la función costo. Esta función no necesita ser invariante en el tiempo. Con esta nueva función, un movimiento d de la solución x que estuviese prohibido según la lista tabú podría sin embargo, ser aceptado si se verificara el criterio de la función de nivel de aspiración:

$$f(d(x)) < A(d, x) \tag{5}$$

II.B.2) Métodos basados en población

Las metaheurísticas basadas en poblaciones (PBM) son algoritmos que trabajan con un *conjunto de soluciones* (por ejemplo, una población) al mismo tiempo [7]. A simple vista, puede parecer que la idea no aporta nada nuevo, ya que podríamos ejecutar varias veces los algoritmos anteriores e incrementar así la probabilidad de obtener el óptimo global. Pero hay un concepto fundamental que hace que los algoritmos basados en población se diferencien radicalmente de los anteriores: el concepto de *competencia* entre las soluciones de la población. La idea es simular a un proceso de evolución en el que la población compita por mantenerse durante el proceso de selección. Este proceso proporciona una forma natural para la exploración del espacio de búsqueda.

Algunos de los métodos basados en población más estudiados son la *computación evolutiva* y *optimización de colonias*. El primero de ellos se discutirá en la siguiente sección. El segundo está fuera del alcance de este proyecto, pero podemos encontrar la descripción y algunas de sus aplicaciones en [3], [15], [16], [53].

II.C. Computación evolutiva

La computación evolutiva está basada en las ideas de la selección natural, y su aplicación en un ambiente artificial. La selección natural es vista como un proceso de optimización, en el que paulatinamente los individuos de la población se van mejorando para adaptarse a su medio. Para la computación evolutiva, un individuo es una solución potencial a un problema, codificada de acuerdo con el funcionamiento del algoritmo; y el medio donde se desenvuelve lo componen lla función objetivo y las restricciones, que nos dirán qué tan apto es el individuo para sobrevivir.

En general, se llama Algoritmo Evolutivo (en lo sucesivo, AE) a cualquier procedimiento estocástico de búsqueda basado en el principio de evolución. Dicho principio tiene una doble vertiente: la finalidad última es la *supervivencia del más apto* y el modo de conseguirlo es por *adaptación al entorno*. De un modo más gráfico, los más aptos tienen más posibilidades de sobrevivir y, como resultado, más oportunidades de transmitir sus características a las siguientes generaciones.

Más concretamente, al ejecutar un AE una población de

individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después, a un proceso de selección que favorece a los mejores individuos. Cada ciclo de *transformación + selección* constituye una *generación*. Se espera del AE que, tras cierto número de generaciones, es decir, de iteraciones, el mejor individuo esté razonablemente próximo a la solución buscada. El esquema de un algoritmo evolutivo general se podría describir como aparece en el Algoritmo 1 [48][44].

Algoritmo 1: Algoritmo evolutivo.

Generar una población inicial *Pop*.
Evaluar la población inicial.
MIENTRAS que no se satisfaga el criterio de parada HACER:
    Reproducir *AuxPop*, creando una descendencia *Offsp*.
    Mutar/optimizar la descendencia
    Evaluar la Descendencia.
    Seleccionar individuos de *AuxPop* y *Offsp* para la siguiente generación
Devolver la mejor solución encontrada.

Por tanto, la Computación Evolutiva (en lo sucesivo, CE) trata de desarrollar mecanismos estocásticos de búsqueda con los que mejorar las técnicas clásicas de búsqueda determinista cuando éstas no sean buenas o ni siquiera existan. Para que la mejora sea efectiva, tales mecanismos deben ser *dirigidos*, de ahí la necesidad de introducir un procedimiento de *selección*. En definitiva, para poder emular eficientemente el proceso de evolución, un AE debe disponer de:

1) Una población de posibles soluciones debidamente representadas a través de *individuos*.
2) Un procedimiento de selección basado en la aptitud de los individuos.
3) Un procedimiento de transformación, esto es, de construcción de nuevas soluciones a partir de las disponibles actualmente.

Sobre ese esquema general se han desarrollado cuatro paradigmas fundamentales:

- *Los Algoritmos Genéticos*: Se hace evolucionar una población de enteros binarios sometiéndolos a transformaciones unarias y binarias genéricas y a un proceso de selección.
- *Los Programas Evolutivos o de Evolución:* Se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- *Las Estrategias Evolutivas:* Se hace evolucionar una población de números reales que codifican las posibles soluciones de un problema numérico y los tamaños de salto. La selección es implícita.
- *La Programación Evolutiva:* Se hace evolucionar una población de máquinas de estados finitos sometiéndolas a trasformaciones unarias.

II.C.1) Algoritmos genéticos

De entre los cuatro paradigmas principales de la Computación Evolutiva, los Algoritmos Genéticos (en lo sucesivo, AGs) ocupan el lugar central, ya que son aplicados en multitud de campos como biotecnología, computación, ingeniería, economía, química, manufactura, matemáticas, física, etc.

Se debe señalar que la principal innovación de los AGs en el dominio de los métodos de búsqueda es la adicción de un mecanismo de *selección* de soluciones. Dicha selección tiene dos vertientes: a corto plazo los mejores tienen más posibilidades de sobrevivir y a largo plazo los mejores tienen más posibilidades de tener descendencia. A causa de esto, el mecanismo de selección se desdobla en dos: uno elige los elementos que se van a transformar posteriormente (operador de *selección*), el otro operador elige los elementos que van a sobrevivir (operador de reemplazo).

En cuanto a las restantes características, cabe destacar que los AGs son métodos de búsqueda:

- *Ciega*, es decir, no disponen de ningún conocimiento específico del problema. La búsqueda se basa exclusivamente en los valores de la función objetivo.
- *Codificada*, puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.
- *Múltiple*, ya que buscan simultáneamente diferentes puntos de entre un conjunto de candidatos o dominio.
- *Estocástica*, referida tanto a las fases de selección como a las de transformación.

Todas las características enumeradas se introducen deliberadamente para proporcionar más robustez a la búsqueda, esto es, para darle más eficiencia sin perder la generalidad y viceversa (véase [23]). Goldberg [23] justifica esta afirmación del siguiente modo:

> *Los algoritmos genéticos manejan variables de decisión, o de control, representadas como cadenas con el fin de explotar similitudes entre cadenas de altas prestaciones. Otros métodos tratan habitualmente con las funciones y sus variables de control directamente.*
>
> *Los algoritmos genéticos trabajan con una población; muchos otros métodos trabajan con un único punto. De este modo, los AGs encuentran seguridad en la cantidad. Al mantener una población de puntos bien adaptados, se reduce la probabilidad de alcanzar un falso óptimo.*
>
> *Las reglas de transición de los algoritmos gen éticos son estocásticas. Hay una diferencia, no obstante, entre los operadores estocásticos de los algoritmos genéticos y otros métodos que no son más que paseos aleatorios. Los algoritmos*

genéticos usan el azar para guiar una búsqueda fuertemente explotadora. Puede parecer inusual hacer uso del azar para conseguir resultados concretos (los mejores puntos), pero hay gran cantidad de precedentes en la naturaleza.

La característica esencial de los AGs, que no se observa directamente, es su capacidad de intercambio estructurado de información en paralelo, o abreviadamente el *paralelismo implícito*. De modo intuitivo, esta característica se refiere a lo siguiente: los AGs procesan externamente cadenas de códigos, sin embargo, lo que se está procesando internamente son similitudes entre cadenas y de manera tal que al procesar cada una de las cadenas de la población se están procesando a la vez todos los patrones de similitud que contienen, que son muchos más. Es precisamente por esta propiedad que los AGs son mucho más eficaces que otros métodos de búsqueda ciega, y por tanto más robustos, con independencia de otras consideraciones.

Estructura y componentes básicos

En la ejecución de un Algoritmo Genético, una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después a un proceso de selección que favorece a los mejores individuos. Esta característica es común a todos los Algoritmos Evolutivos (AEs). Cada ciclo de selección + búsqueda constituye una generación, y se espera del AE que, al cabo de un número razonable de generaciones, el mejor individuo represente a un candidato lo suficientemente próximo a la solución buscada. Los Algoritmos Genéticos, como ejemplo fundamental de algoritmos evolutivos, siguen dicho esquema básico con ciertas peculiaridades propias.

De modo gráfico, la estructura genérica del bucle básico de un AG se sintetiza en el diagrama de la Figura 2.

Como puede observase en la Figura 2, dada una representación genética y una función de idoneidad, un algoritmos AG comienza con la inicialización de una población aleatoria (solución posible). Dicha población se mejora mediante la aplicación de los operadores de mutación, cruce y selección. El algoritmo termina cuando se encuentra un buen nivel de idoneidad o cuando se alcanza el número máximo de generaciones, en tal caso, puede o no que hayamos encontrado una buena solución. En el siguiente apartado, se describen los procedimientos básicos de un AG.

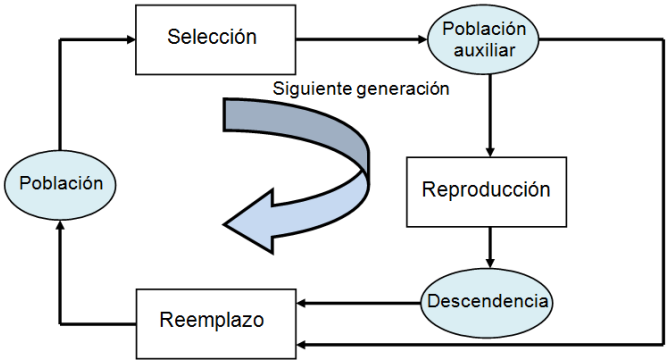

Figura 2. Ciclo básico de un GA

A la estructura de un individuo se le conoce como genotipo y a su contenido fenotipo. A la hora de programar un AG, el genotipo se implementa mediante una clase y los fenotipos a través de objetos de dicha clase. Todo el procedimiento de búsqueda está guiado exclusivamente por una función de aptitud *u(x)*, la cual se obtiene directamente a partir de la función de evaluación (o función costo) *C(x)* del correspondiente problema. Comúnmente, a los valores proporcionados por la función de evaluación se les dice *evaluaciones* o bien *aptitudes brutas*, mientras que a los valores proporcionados por la función de aptitud se les dice aptitudes a secas, y también, para distinguirlas de las aptitudes brutas, *aptitudes netas*.

Como puede verse, una población *Pop*, que consta de *n* miembros se somete a un proceso de *selección* para constituir una población intermedia *AuxPop* de *n criadores*. De dicha población intermedia se extrae un grupo reducido de individuos llamados *progenitores* que son los que efectivamente se van a reproducir. Sirviéndose de los *operadores genéticos*, los progenitores son sometidos a ciertas transformaciones de *mutación* y *recombinación* en la fase de *reproducción*, en virtud de las cuales se generan s nuevos individuos que constituyen la descendencia (*Offsp*). Este nuevo conjunto de individuos se denotará como *X*. Para formar la nueva población *Pop[t + 1]* se deben seleccionar *n* supervivientes de entre los *n+s* de la población auxiliar y la descendencia, eso se hace en la fase de *reemplazo*. Como ya se comentó, la selección se hace en dos etapas con la idea de emular las dos vertientes del Principio de Selección Natural: selección de criadores o 'selección' a secas y selección de supervivientes para la próxima generación o 'reemplazo'.

Los objetos que se someten a evolución en un AG son cadenas binarias v sobre las que se codifican los elementos x del espacio de búsqueda; ambos reciben el nombre de *individuo*, *codificado* el uno y *sin codificar* el otro. También son usuales las denominaciones 'ejemplar','muestra' o 'punto'.

Cada individuo *x* consta de *m* posiciones (*locus/i*) que son ocupadas por otros tantos atributos o *genes*, los cuales se codifican en $\ell = L_1 + \ldots + L_m$ bits. Naturalmente, si un atributo puede tomar más de dos valores o *alelos* se debe representar mediante varios bits; de hecho es inmediato comprobar que si el *j*-ésimo gen consta de a_j alelos, entonces se deben codificar mediante $L_j = \lfloor \log_2 a_j \rfloor$ bits. En este punto conviene distinguir entre la estructura de los individuos y el contenido de cada uno; la estructura se deriva de la *representación*, es igual para todos y se expresa así: los 4 primeros bits representan al primer gen, los 8 siguientes al segundo, etc.; el contenido se deriva de la *codificación* y es simplemente el entero binario con que se codifica cierto individuo.

Procedimientos básicos de un AG

Los tres procedimientos básicos de que consta un AG son: selección (de criadores), reproducción (o transformación) y reemplazo (o selección de supervivientes). Nótese que para diseñar un AG se requiere un mínimo de dos parámetros: el tamaño de la población n y el tamaño s de la descendencia X.

El proceso de selección: Consiste en muestrear, a partir de la población inicial, los n elementos de la población de criadores. El criterio concreto de muestreo depende del problema y del buen juicio del programador. Los más usados en la práctica son los muestreos por sorteo, universal y por torneos, en sus variedades conservadoras. Los muestreos deterministas se usan muy poco, entre otros motivos porque van contra la filosofía del método.

1) *Muestreo directo*: Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de "los k mejores", "los k peores", "a dedo", etc...

2) *Muestreo aleatorio simple o equiprobable*: Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra y se constituye ésta mediante ensayos de Bernuolli simples.

3) *Muestreos estocásticos:* Se asignan probabilidades de selección o *puntuaciones* a los elementos de la población base en función (directa o indirecta) de su aptitud. Por defecto, la puntuación p_i asociada al individuo x_i de la población $P = \{x_1, ..., x_2\}$ se calcula como la aptitud relativa de dicho individuo; esto es, siendo $u_1, ... , u_n$ las respectivas aptitudes se tiene que

$$p_i \overset{def}{=} \frac{u_i}{u_1 + \ldots + u_n} (\forall i = 1,\ldots,n) \qquad (6)$$

así se garantiza que $p_1 + ... + p_n = 1$, tal como corresponde a una distribución de probabilidades.

La muestra se constituye de modo estocástico haciendo uso de dichas puntuaciones.

Existen muchos mecanismos de muestreo estocástico. En concreto, al diseñar algoritmos estocásticos se usan fundamentalmente cuatro tipos de muestreo estocástico, que se describirán seguidamente. Para ilustrarlos, se considera el problema de muestrear estocásticamente dos elementos de la población $P = \{x_1, x_2, x_3, x_4\}$ siendo las puntuaciones asociadas $p_1 = 0{,}1$, $p_2 = 0{,}3$, $p_3 = 0{,}1$ y $p_4 = 0{,}5$ respectivamente.

- *Por sorteo:* Se consideran las puntuaciones estrictamente como probabilidades de elección para formar la muestra, y se constituye ésta realizando k ensayos de una variable aleatoria con dicha distribución de probabilidades.

  Dado que habitualmente sólo se dispone de un generador de números aleatorios simples (es decir, , uniformemente distribuidos entre 0 y 1), para simular un ensayo de la distribución $\{p_1 , ... , p_n\}$ se hace lo siguiente:

  a) Se calculan las puntuaciones acumuladas así:

  $$q_0 := 0$$
  $$q_i := p_1 + \ldots + p_i \quad (\forall i = 1,\ldots,n) \qquad (7)$$

  b) Se genera un número aleatorio simple r← *URand*[0, 1) .

  c) Se elige el individuo x_i que verifique:

  $$q_{i-1} < r < q_i \qquad (8)$$

  Para muestrear por sorteo k individuos, se modifican trivialmente los dos últimos pasos así:

  b) Se generan k números aleatorios simples r_j *URand*[0, 1) (∀j=1,...,k)

  c) Para cada $j = 1, ..., k$ se elige el individuo x_i que verifique

  $$q_{i-1} < r_j < q_i \qquad (9)$$

  Para el ejemplo propuesto las puntuaciones acumuladas son $q_1 = 0{,}1$ $q_2 = 0{,}4$ $q_3 = 0{,}5$ y $q_4 = 1{,}0$ respectivamente, y los $k = 2$ números aleatorios simples se han hallado como $r_1 = 0{,}02433$ y $r_2 = 0{,}51772$

  De esta manera, el primer elemento de la muestra resulta ser x_1 dado que $q_0 < r_1 < q_1$ y el segundo será x_4 puesto que $q_3 < r_1 < q_4$. Nótese que existe la posibilidad de que algunos individuos puedan ser elegidos varias veces en una muestra (los que tengan mayor puntuación o un golpe de buena suerte con los r_j) y que otros individuos pueden no ser elegidos nunca (los que tengan menor puntuación o mala suerte).

  Debido a sus buenas propiedades teóricas, este es el muestreo que se utiliza por defecto con los AGs. No obstante sus prestaciones en la práctica no están siempre a la altura de las expectativas teóricas.

- *Por restos:* A cada individuo x_i se le asignan directamente $\lfloor p_i \cdot k \rfloor$ puestos en la muestra. Seguidamente los individuos se reparten los puestos vacantes en función de sus puntuaciones. El reparto se suele hacer por sorteo (ver anterior apartado) y entonces se le llama 'muestreo estocástico por restos'.

  Para el ejemplo propuesto, a x_4 le corresponde un puesto en la muestra; el individuo que ocupe el otro puesto se elige por sorteo, es decir, se genera un número aleatorio simple $r = 0{,}39874$ y se compara con las puntuaciones acumuladas q_i, en este caso el puesto vacante le corresponde a x_2 dado que $q_1 < r < q_2$.

  Existe una variante en la que el reparto de las vacantes se hace por muestreo directo; nótese que en esta variante no hay ninguna intervención del azar, es decir, no es un método de muestreo estocástico sino directo, por este motivo se le llama 'muestreo determinista por restos', para distinguirlo del descrito anteriormente. Obsérvese que este último método de muestreo no es otra cosa que la conocida regla de repartos enteros de d'Hont.

- *Universal o por ruleta*: Es análogo al muestreo por sorteo sólo que ahora se genera un único número aleatorio simple *r* y con él se asignan todas las muestras de modo parecido a como se haría al girar una ruleta (ver Figura 3).

  Para simular una tirada de ruleta, a partir de *r*←*URand*[0, 1) se definen los siguientes números:

$$r_j := \frac{r + j - 1}{k} \quad (\forall j = 1,\ldots,k) \qquad (10)$$

  Con eso ya se determina la muestra comparando los r_j con las puntuaciones acumuladas a la manera habitual.

  Para el ejemplo propuesto, si el número aleatorio es *r*= 0,39874 entonces se tiene: r_1 = 0,19937, r_2 = 0,69937 lo que proporciona la muestra { x_2, x_4}.

  Como puede verse, el diseño del muestreo estocástico universal es muy sencillo y rápido, y en la práctica proporciona unas prestaciones análogas a las del muestreo por sorteo; por este motivo se suele usar en sustitución de éste.

  A propósito de todo esto, en la bibliografía antigua se suele llamar muestreo por ruleta (*roulette wheel sampling*) a lo que aquí se ha llamado muestreo por sorteo (*lottery sampling*), y muestreo por sorteo a lo que aquí se ha llamado muestreo universal.

- *Por torneos:* Cada elemento de la muestra se toma eligiendo el mejor de los individuos de un conjunto de *z* elementos tomados al azar de la población base; esto se repite k veces hasta completar la muestra. El parmetro *z* suele ser un entero pequeño comparado con el tamaño de la población base, normalmente 2 ó 3. Nótese que en este caso no es necesario hacer la asignación de puntuaciones.

El *proceso de reproducción*: aplicando los operadores de transformación (cruce, mutación, inversión, ...) sobre ciertos miembros de la población de criadores se obtiene una descendencia de s nuevos miembros. Este proceso se suele realizar mediante ensayos de Bernuolli; se admite a veces la variedad excluyente. No es necesario (de hecho, no es habitual) dar por adelantado el valor de *s*, normalmente sólo se da su valor esperado de modo indirecto a través de las probabilidades de aplicación de los operadores genéticos. Cuanto más grande sea el valor (real o esperado) de *s* más variará la población de una generación a otra. Salvo indicación en contra, lo común es trabajar con valores de *s* no mayores del 60% de *n*.

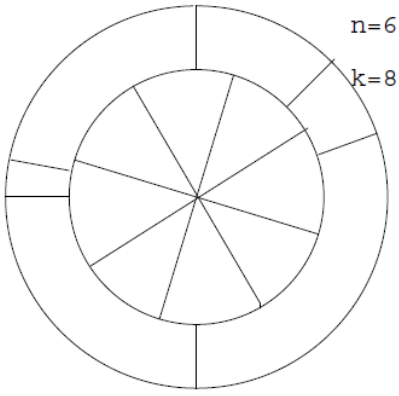

Figura 3. Muestreo por ruleta

Existen dos grupos de operadores genéticos que nunca faltan en un algoritmo genético: el cruce y la mutación.

Los operadores de cruce son el arquetipo de operadores de *recombinación*: actúan sobre parejas de individuos y normalmente originan otro par de individuos que combinan características de los progenitores. Dado que en los AGs los individuos están representados a través de cadenas binarias, el cruce se lleva a cabo por intercambio de segmentos, a la manera indicada en la Figura 4.

Los operadores de mutación, por su parte, son el arquetipo de operadores de *mutación*, dado que actúan sobre individuos solos, realizando una pequeña modificación de alguno de sus genes o en el conjunto.

El motivo de hacer esta separación es el siguiente: entendiendo que la búsqueda propiamente dicha se lleva a cabo en la fase de reproducción, resulta conveniente diferenciar los operadores de búsqueda en profundidad de los de búsqueda en anchura; los primeros se encargarán de explotar las mejores características de que disponga la población actual, los otros se encargarán de explorar nuevos dominios en busca de mejores soluciones. Desde esta perspectiva, los operadores de cruce son los que principalmente se encargan de la búsqueda en profundidad y los de mutación de la búsqueda en anchura. Así, dando mayor importancia a unos o a otros se puede ajustar el tipo de búsqueda a las necesidades del problema.

Sin necesidad de entrar de momento en mayores precisiones es evidente que la división tiene un valor práctico para diferenciar dos modos complementarios de desarrollar la reproducción: el cruce se encarga de realizar un intercambio estructurado de información, la mutación proporciona una garantía de accesibilidad para todos los puntos del espacio de búsqueda.

Existe una variante del proceso de reproducción que recibe el nombre de *reproducción operacional*; en ella cada miembro de la descendencia es producido por un solo operador genético y además estos operadores sólo pueden actuar sobre los individuos, no sobre las poblaciones o sobre los genes.

El proceso de reemplazo: A partir de los *n* miembros de la población de criadores y de los *s* miembros de la población de descendientes (*X*) se debe obtener una nueva población de *n* miembros. Para hacerlo existen varios criterios:

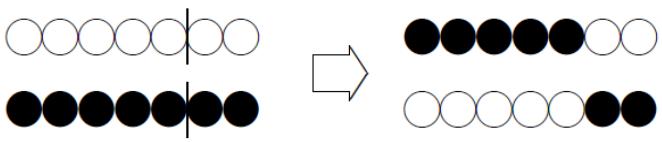

Figura 4. Cruce

1) *Reemplazo inmediato (o al vuelo*): Los *s* descendientes sustituyen a sus respectivos progenitores.

2) *Reemplazo con factor de llenado*: Los *s* descendientes sustituyen a aquellos miembros de la población de criadores que más se les parezcan.

3) *Reemplazo por inserción (o de tipo 'coma'):* Según el tamaño relativo de la descendencia respecto de la población se distinguen dos casos:

- $s \leq n$*:* Se muestrean para ser eliminados s miembros de la población de criadores (según cierto criterio; normalmente, los peores). Esos miembros serán sustituidos por los descendientes (*X*).
- $s > n$*:* Se muestrean *n* miembros de la población de descendientes y se constituye con ellos la nueva población. Nótese que de este modo cualquier individuo sólo puede vivir a lo sumo una generación.

4) *Reemplazo por inclusión (o de tipo 'más'*): Se juntan los *s* descendientes con los *n* progenitores en una sola población, y en ella se muestrean *n* miembros (normalmente, los mejores).

Naturalmente, los dos primeros criterios sólo se pueden usar cuando $s \leq n$.

II.C.2) Estrategias Evolutivas

Las estrategias evolutivas (en lo sucesivo ES, de *Evolution Strategies*) surgieron como métodos estocásticos de escalada con paso adaptativo específicamente diseñados para resolver problemas de optimización paramétrica. Con el paso del tiempo fueron incorporando procedimientos propios de la computación evolutiva hasta convertirse en un paradigma más de dicha metodología. En el presente, las ES son algoritmos evolutivos enfocados preferentemente hacia la optimización paramétrica que utilizan una representación a través de vectores reales, selección determinista y operadores específicos de mutación y cruce, el cual ocupa ahora un lugar secundario con respecto a la mutación. Las ES son el paradigma más fuerte es decir, más específico, de la CE; sin embargo, mantienen la filosofía de las 'cajas negras' y operadores ciegos.

Estrategias Evolutivas simples

Las ES simples (también llamadas ES de dos miembros) son el modelo más sencillo de su paradigma. Aunque no son métodos evolutivos propiamente dichos, contienen en su forma más pura gran parte de las ideas que incorporan las ES múltiples. Pese a su sencillez, las ES simples tienen utilidad práctica y aún se siguen utilizando versiones mejoradas para resolver 'problemas reales' de ingeniería.

Las ES simples son básicamente procedimientos estocásticos de optimización paramétrica con paso adaptativo. En la terminología del área se denotan como (1 + 1)-ES y se describen así:

> *En las (1+1)-ES evoluciona un sólo individuo haciendo uso únicamente de un operador genético, la mutación. Los individuos son bicromosomáticos y se representan a través de un par de vectores reales v := <x;s>*
>
> *El primer vector x representa a un punto del espacio de búsqueda, y el segundo, s, es un vector de desviaciones típicas que se usa al realizar la mutación. Al haber un sólo progenitor no hay selección, y el criterio de reemplazo consiste en que el descendiente reemplaza a su progenitor si y sólo si es más apto que éste; en caso contrario, el descendiente es eliminado y la 'población' permanece sin cambios.*

Concretamente, la evolución del cromosoma x se puede sintetizar así:

$$x[t+1] = \begin{cases} x[t] + N(0, s[t]) & sii \quad x[t+1] \succ x[t] \wedge x[t+1] \in X \\ x[t] & \text{en caso contrario} \end{cases} \tag{11}$$

donde N(0, S) es un vector de números aleatorios gaussianos independientes con medias 0 y desviaciones típicas S; a veces se usa también una distribución de probabilidades lognormal. Salvo indicación en contra habitualmente se acepta que

$$S[t] := \sigma[t] \ (1; \ldots ; 1) \tag{12}$$

Puede demostrarse (véase [35]) que para un número finito de óptimos y bajo ciertas condiciones de regularidad el esquema anterior converge al óptimo cuando $t \rightarrow \infty$ con probabilidad 1. No existen resultados concluyentes sobre la velocidad de convergencia, tan solo una regla semi empírica que afirma lo siguiente:

> *Hipótesis de 1/5 de éxitos: La velocidad óptima de convergencia se alcanza cuando la relación de mutaciones con éxito frente al total de mutaciones, φ[t], vale 1/5.*

A consecuencia de esa hipótesis resulta innecesario someter a evolución al segundo cromosoma, dado que ya se conoce la estrategia óptima de evolución para él: $S^o[t]$ es tal que proporciona una tasa de éxito en las mutaciones lo más próxima posible a 1/5. Eso se concreta en la siguiente ley de adaptación para el segundo cromosoma:

$$\sigma[t+1] = \begin{cases} c_d \sigma[t] & sii \quad \varphi_k[t] > 1/5; \\ \sigma[t] & sii \quad \varphi_k[t] = 1/5; \\ c_i \sigma[t] & sii \quad \varphi_k[t] < 1/5. \end{cases} \tag{13}$$

donde $\varphi_k[t]$ es el coeficiente de éxito del operador mutación durante las *k* últimas generaciones (*k* es un parámetro del método) y $c_i>1$ y $c_d<1$ son coeficientes con los que se regula la tasa de adaptación. Una parametrización típica es: $k \approx 5$ $c_d \approx 0{:}82$, $c_i = 1/c_d$. De ese modo, la búsqueda avanzará con pasos largos cuando tenga éxito y con pasos cortos en caso contrario. Nótese que la mutación propiamente dicha sólo

afecta al primer cromosoma, pues el segundo tan sólo se adapta en función del éxito que haya tenido la mutación del primero.

II.D. Algoritmos Genéticos Niching

Un AG simple representa una abstracción de un proceso elemental de la naturaleza, combinado con cuatro puntos clave: (1) presión selectiva para promocionar soluciones superiores, (2) altas tasas de "recombinación" para yuxtaponer varios componentes de soluciones para formar mejores soluciones, (3) uso de grandes poblaciones para absorber los efectos estocásticos, y (4) funciones de mutación e idoneidad para evitar quedarse atrapado en un óptimo local. La mayoría de los AG convergen rápidamente a una población, uniforme o no uniforme, de soluciones bastante buenas.

No obstante, existen muchos dominios donde no se desea una convergencia uniforme: la clasificación y el aprendizaje de máquinas, optimización de funciones multimodelo, optimización de funciones mutiobjetivo, y simulación de sistemas complejos y adaptativos. Un AG *simple* podría mejorarse si los individuos cooperasen entre sí. Sin embargo, parece difícil, debido principalmente a la dificultad de mantener a un grupo cooperativo de individuos mientras que el proceso de selección intenta elegir sólo al mejor. Otra opción podría ser decrementar la presión de la selección, lo que provocaría que la población no convergiese en el marco de tiempo establecido (número de generaciones). Pero reduciendo la presión de la selección también se reduce la velocidad y eficiencia de los algoritmos de búsqueda estocásticos (AG). Con una selección insuficiente, podemos conseguir mucha más exploración y no suficiente explotación.

Los métodos de niching extienden a los algoritmos genéticos a dominios que requieren la localización y el mantenimiento de múltiples soluciones [49]. La Figura 5 representa una función objetivo que contiene muchos óptimos globales y locales. Los algoritmos genéticos guían la población inicial a un único óptimo global, ignorando la presencia de otras soluciones globales, debido al uso del operador de selección. Este efecto, conocido como *deriva genética*, está representado en la Figura 6, la cual muestra el valor objetivo representado en la Figura 5. El color rojo oscuro corresponde con el valor objetivo más alto, mientras que el color azul oscuro, el menor valor objetivo. Los pequeños círculos en blanco representan las subpoblaciones.

Los métodos niching se han desarrollado precisamente para reducir dicha deriva genética. Mantienen la diversidad de la población, y permiten a los AGs investigar muchas cimas en paralelo. Como puede verse en la Figura 7, a diferencia de los AG, los métodos niching promueven la formación de subpoblaciones alrededor de las soluciones óptimas locales y globales. Sin embargo, previenen a los AGs de quedar atrapados en un óptimo local del espacio de búsqueda. Los algoritmos genéticos niching están basados en los mecanismos de los ecosistemas naturales. En la naturaleza, los animales compiten por sobrevivir cazando, pastando, reproduciéndonse, etc. Un nicho puede ser visto como un subespacio en un entorno con diferentes tipos de vida. Una especie es un grupo de individuos con características biológicas similares capaces de reproducirse entre ellos pero no con otros individuos de otras especies. Para cada nicho, los recursos físicos son finitos y deben ser compartidos por todos os miembros de la población del nicho. El tamaño de cada subpoblación reflejará la disponibilidad de recursos en el nicho. Comúnmente, se refiere a nicho como el óptimo del dominio.

Podemos encontrar una importante variedad de métodos niching en [6], [11], [12], [20], [24], [68], [32] .

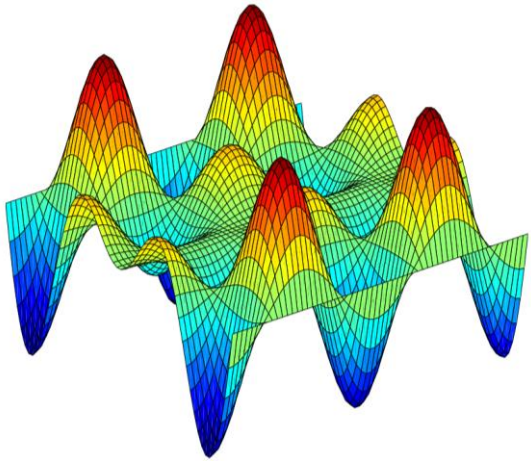

Figura 5. Función ejemplo

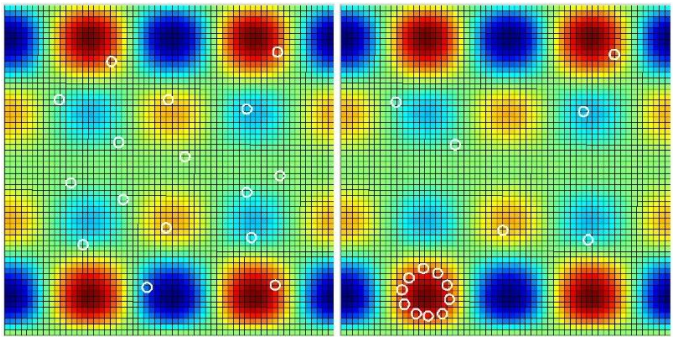

Figura 6. Efecto deriva genética: Población inicial (derecha), después de n generaciones (izquierda)

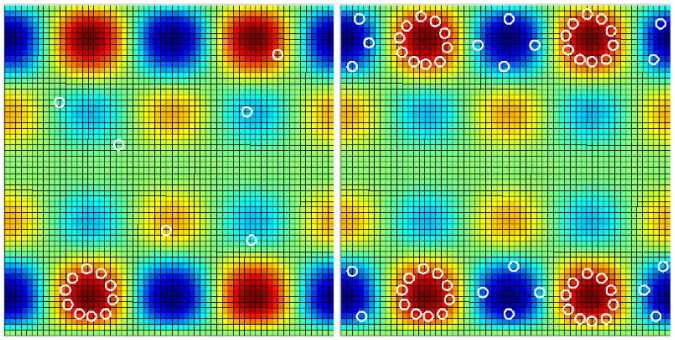

Figura 7. GA vs métodos Niching

II.E. Algoritmos Meméticos

La denominación de Algoritmos Meméticos (MAs) hace referencia a una amplia clase de *metaheurísticas* (véase sección II.B). Están basados en poblaciones de agentes. Un conjunto de agentes competitivos se encargan de la mejora de soluciones de forma individual, y raramente interaccionan entre ellos [37].

A diferencia de la Computación Evolutiva (CE), los algoritmos MAs pretenden explotar todo el conocimiento disponible sobre el problema, como una manera de mejorar la velocidad del proceso de búsqueda [38]. Los MAs se aprovechan del conocimiento del problema incorporando heurísticas preexistentes, técnicas de búsqueda local, operadores de recombinación especializados, métodos exactos de truncamiento, etc. Un factor importante es la representación adecuada de los problemas. El término *hibridación* es comúnmente usado para denotar los procesos de incorporación del conocimiento del problema [10]. Por esta razón, a veces se llama a los MAs *Algoritmos Evolutivos Híbridos*. La incorporación de conocimiento de dominio del problema no es un mecanismo opcional, sino una característica fundamental de los MAs.

La filosofía de funcionamiento se ilustra perfectamente mediante el término "memético". Acuñado por Dawkins [13], la palabra meme es una analogía a los genes en el contexto de la evolución cultural [36]. En palabras de Dawkins:

> *Ejemplos de "memes" son melodías, ideas, frases hechas, moda, la manera de construir arcos. Así los genes se propagan en el acervo genético saltando de un cuerpo a otro a través de los espermatozoides o los óvulos, "memes" se propaga en el "acervo cultural" saltando de un cerebro a otro mediante un proceso de imitación.*

Algoritmo memético básico

Un algoritmo memético comienza con el proceso de *inicialización* [38]. A diferencia de los AEs estándar, que simplemente generan un número aleatorio de soluciones, los MAs usan mecanismos más sofisticados, como por ejemplo la construcción de heurísticas [30], [51]. Un proceso iterativo nos conduce hasta la satisfacción del criterio de terminación (véase Figura 8).

La inclusión de conocimiento contribuye a la aceleración de la convergencia de la población. Por tanto, el proceso de reactivación es muy importante en los MAs. Además, si una población converge, es mejor refrescarla que mantenerla restringida a un pequeña región del dominio de búsqueda. El proceso de reactivación se puede representar de muchas maneras. Una de ellas podría ser: una pequeña fracción *p* de población se mantiene (este valor no debería ser muy alto ya que sino la población volvería a converger), y se renombra las soluciones desde el principio, como en el proceso de inicialización.El proceso generacional es la parte del

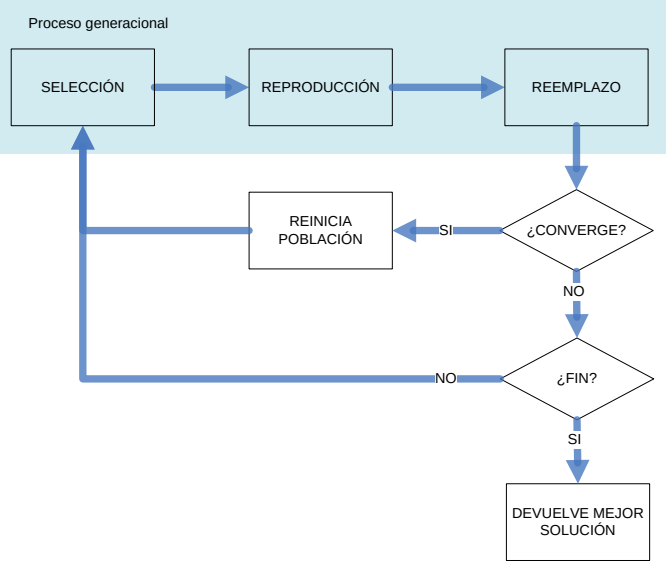

Figura 8. Estructura general de MAs

algoritmo en la cual se produce la evolución de las soluciones. Como puede verse en la Figura 8, el bloque generacional incluye tres pasos: selección, reproducción y actualización o reemplazo. El primero y el último son responsables de la competencia de los individuos de la población.

El operador de *selección* evalúa la idoneidad de los individuos de la población, usando la información proporcionada por la función de idoneidad y termina con una muestra de individuos que se reproducirán. El proceso de *actualización* mantiene a la población en un tamaño constante sustituyendo algunos individuos pre-existentes en la población original por algunos obtenidos en una nueva población (usando algún criterio específico). En la etapa de *reproducción*, se crean los nuevos individuos (o agentes) usando algún operador de reproducción (mutación, búsqueda local, etc.).

Para más detalle sobre el diseño de MAs, se remite al lector a [38], [39], [40], [41].

III. ALGORITMO UEGO

En este apartado se va a describir UEGO, un algoritmo de optimización global que presenta ciertas características típicas de los algoritmos genéticos y evolutivos. El nombre de "UEGO" viene de *Universal Evolutionary Global Optimizer*.

Este algoritmo surge a partir de GAS (Genetic Algorithm Species based), y por tanto comparten algunos conceptos, pero a su vez varía algunas propiedades con el fin de obtener un algoritmo que alcance el óptimo con mayor precisión y que presente una estructura de población mucho más sencilla para poder ser paralelizado [44].

UEGO mantiene el método de enfriamiento o disminución del radio. Esta técnica es muy eficaz para problemas de tipo radio niche las cuales tienen varios óptimos locales dispuestos de manera dispar en el espacio de búsqueda. Para este tipo de funciones no se puede determinar el radio niche

adecuadamente ya que si es demasiado pequeño la búsqueda resulta inefectiva, y si es demasiado grande aquellos óptimos locales cercanos entre sí no podrán ser distinguidos. La técnica de enfriamiento realiza la búsqueda centrándose en regiones prometedoras del espacio empezando con un radio relativamente grande que va disminuyendo a la par que progresa la búsqueda. UEGO combina esta técnica de enfriamiento añadiendo la posibilidad de escapar de óptimos locales ya que siempre destina una parte del trabajo de búsqueda al radio máximo de todo el espacio del dominio del problema.

El optimizador local de UEGO se ha separado como un módulo del resto del algoritmo con objeto de utilizar un optimizador específico para cada dominio de búsqueda, y por tanto el algoritmo puede adaptarse a un gran número de dominios de búsqueda.

Otra diferencia destacable es la simplificación del concepto especie: una especie se identifica con un individuo, y se han eliminado las dependencias entre especies.

Por tanto UEGO es un algoritmo capaz de resolver problemas de optimización multimodales donde la función objetivo tiene varios óptimos locales y de descubrir la estructura de dichos óptimos.

En UEGO cada especie intenta alcanzar un máximo local de la función objetivo, sin conocimiento de la cantidad total de máximos locales en el dominio del problema. Inicialmente existe una sola especie (especie raíz) y durante el desarrollo del algoritmo se aplican operadores genéticos: se crean nuevas especies, y se fusionan algunas de las existentes.

Conceptos básicos

El principal concepto de UEGO es el de especie, que se puede representar gráficamente como una ventana en el espacio de búsqueda. Esta ventana viene definida por su *centro* y su *radio*. El centro es una solución o punto en el espacio de búsqueda y el radio es un número positivo que está directamente relacionado con la amplitud de esta ventana. Una especie tiene un area de tracción definida por su radio, que cubre la región del espacio de búsqueda y por lo tanto múltiples soluciones. Esta definición parte de que existe un concepto de *distancia* dentro del espacio de búsqueda, para problemas reales se considerará la distancia Euclídea. El papel de esta ventana es "posicionar" al optimizador dentro de una zona o región del dominio de búsqueda. Al ser llamado desde una especie, el optimizador tiene limitada su amplitud de búsqueda, de modo que no puede avanzar más allá de los límites impuestos por el tamaño del radio. Si el optimizador encuentra una solución que es mejor que el 'centro' de la especie desde la que es llamado, entonces, esta nueva solución se convierte en el nuevo centro, de modo que la ventana se desplaza, si bien su radio se mantiene constante (vea Figura 9). Hay que destacar que en UEGO cada especie sólo viene definida por una única solución, que es el centro de la especie.

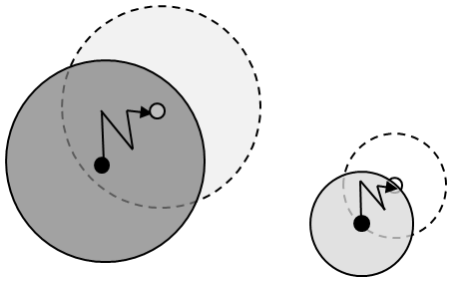

Figura 9. Actualización del centro de las especies

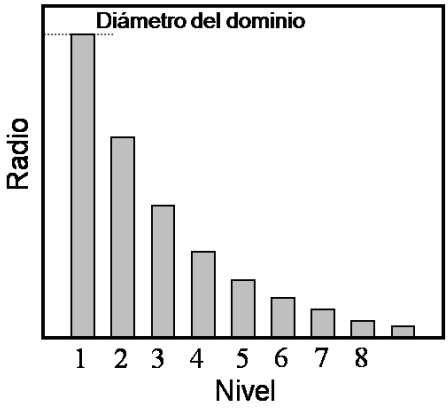

Figura 10. Disminución exponencial del radio

El radio de una especie no toma valores arbitrarios sino que se toma de una lista de radios decrecientes (*radius list*), que se calculan matemáticamente al inicio del algoritmo (véase Figura 10). De este modo el primer radio de la lista coincide con el diámetro del dominio del espacio de búsqueda inicial. Esta lista de radios preesenta tantos elementos como niveles tiene el algoritmo, de modo que si el radio de una especie creada coincide con el valor del *i*-ésimo elemento de la lista de radios, entonces, se dice que la especie se creó en el nivel (level) *i*. Cuando se crea una nueva especie se le asocia el radio correspondiente al nivel actual. Por tanto especies creadas en distintos niveles tendrán distintos radios. Cuando el radio es más pequeño (en el último nivel de creación) producirá un movimiento más lento en el espacio pero será capaz de distinguir entre óptimos locales que estén cercanos entre sí.

Durante el proceso de optimización, UEGO mantiene una lista de especies con la información de su centro y radio, pero no mantiene una relación de parentesco ni dependencia entre las especies. El algoritmo es un método que básicamente maneja esta lista de especies *species_list* (crea, borra y optimiza especies).

El número máximo de niveles viene dado por un parámetro de entrada del algoritmo llamado *levels*. A cada nivel válido i($1 \leq i \leq$ levels), se asocia un valor de un radio (R_i) y dos números que indican el número máximo de evaluaciones de la función. Uno de estos números indica el número máximo de evaluaciones que se pueden realizar en un nivel dado para

crear especies (new_i), y el otro indica el número máximo de evaluaciones que se pueden realizar en un nivel dado durante el proceso de optimización de todas las especies (n_i). También es necesario definir otro parámetro adicional que indique el número máximo de especies que pueden existir simultáneamente (el tamaño de la lista de especies); este parámetro se llama *max_spec_num*.

Parámetros

Los parámetros más importantes son aquellos relacionados con los diferentes niveles: los radios y el número de evaluaciones de la función que se utilizan en los mecanismos de creación y optimización de especies (véase Algoritmo 3). A continuación se describe un método que calcula estos parámetros a partir de unos pocos parámetros definidos por el usuario, y que son fáciles de comprender.

- *evals(N)*: El número máximo de evaluaciones de función que el usuario permite al proceso de optimización completo. Notar que el número de evaluaciones requeridas por el algoritmo puede ser menor que este valor.
- *levels(L):* El máximo número de niveles (véase Algoritmo 3), el número de distintas etapas de enfriamiento, o el número de generaciones.
- *max_spec_num(M):* El tamaño máximo de la lista de especies. El número máximo de especies que se pueden definir.
- *min_r(R_L):* El radio asociado con el último nivel (es decir, el nivel *levels*). Indica el radio mínimo que puede tener una especie.

Como se ha mencionado anteriormente, el optimizador no puede realizar pasos de búsqueda mayores que el radio de la especie donde esté trabajando. Dado un cierto número de evaluaciones, es posible medir la distancia que una especie dada se puede mover durante el proceso de optimización. Esta distancia se puede aproximar como una función del radio y del número de evaluaciones utilizando modelos matemáticos o resultados experimentales. Esto naturalmente nos lleva a la noción de velocidad que dependerá del radio de la especie, y que notaremos por *v(R):*

$$v(R_i) = \frac{\binom{n}{\frac{n-1}{2}}}{2^{n+1}} \cdot r_i \quad (i = 2,\ldots,L) \tag{14}$$

donde *n* indica la dimensión del problema a resolver. El método que selecciona los parámetros se basa en los siguientes *principios* (véase [26], [45]):

Principio de igualdad de oportunidad. En cada nivel, cada especie se mueve una cierta distancia a partir de su centro original debido a la optimización. Este principio asegura que cualquier especie recibirá un número de evaluaciones suficiente para recorrer al menos una distancia fija en cada nivel. Esta distancia común se define por $R_1 v$. Aquí se puede ver cómo el umbral *v* controla directamente la distancia máxima que puede recorrer, así que controla la estabilidad de las especies resultantes (por ejemplo, la probabilidad de que éstas representen un óptimo local). Recordar que R_1 es siempre el diámetro del espacio de búsqueda. En principio se puede formalizar:

$$\frac{v(R_i)n_i}{M} = R_1 v \quad (i = 2,\ldots,L) \tag{15}$$

Principio del decremento exponencial del radio. Este principio indica que dados el menor y mayor radio (R_L y R_1) el resto de radios se pueden expresar mediante una función exponencial:

$$R_i = R_1 \left(\frac{R_L}{R_1} \right)^{\frac{i-1}{L-1}} \quad (i = 2,\ldots,L) \tag{16}$$

Principio de oportunidad constante de creación de especies. Este principio asegura que incluso si el tamaño de la lista de especies es el máximo, hay una oportunidad de crear al menos dos especies más por cada especie existente. También hace una fuerte simplificación: el número de evaluaciones debería ser siempre un mismo valor constante.

$$new_i = 3M \quad (i = 2,\ldots,L) \tag{17}$$

Definiendo new_1=0 por simplicidad, ya que UEGO nunca usa new_1. La descomposición de *N* se obtiene de la ecuación:

$$\sum_{i=1}^{L}(n_i + new_i) = (L-1)3M + \sum_{i=1}^{L}(n_i) = N \tag{18}$$

haciendo uso de 8. También se puede utilizar una simplificación adicional; hacer $n_1 = 0$ siempre que $L > 1$. Notar que si $L = 1$, entonces UEGO se reduce al optimizador local. Si en 6 se sustituye n_i en la ecuación 9 se puede obtener la igualdad:

$$(L-1)3M + \sum_{i=1}^{L} \frac{R_1 vM}{vRi} = N \tag{19}$$

Si se usa la Ecuación (16), se puede observar que los parámetros desconocidos de la Ecuación (19) son justamente los parámetros definidos por el usuario, y debido a la monotonía de esta ecuación en cada una de las variables, uno de estos parámetros puede calcularse a partir del resto utilizando métodos numéricos efectivos. Los restantes parámetros de entrada (n_i , new_i y R_i) se pueden calcular utilizando los principios anteriores. Hay que indicar que algunas de las configuraciones impuestas por el usuario pueden no ser factibles.

Descripción del Algoritmo UEGO

La estructura del algoritmo UEGO se describe en el Algoritmo 2. A continuación se describirán cada uno de sus procedimientos.

Procedimiento Init_species_list. Este procedimiento crea una lista nueva de especies, cada una de las cuales consta de un único elemento. Esto se hace escogiendo aleatoriamente un punto del espacio de búsqueda y asociándole el primer nivel y/o radio. Este radio coincide con el diámetro del

dominio de búsqueda de la función o problema a optimizar.

Algoritmo 2: Algoritmo UEGO.

```
Init_species_list
Optimize_species(n1)
DESDE i = 2 HASTA L HACER
   Determinar Ri, newi y ni
   Create_species(newi)
   Fuse_species(Ri)
   Shorten_species list(max_spec_num)
   Optimize_species(ni/ max_spec_num)
   Fuse_species(Ri)
```

*Procedimiento Create species(*new$_i$*).* Para cada una de las especies se crea una sublista de soluciones que se escogen aleatoriamente de la ventana de la especie. Una vez creada la sublista, se van formando parejas de soluciones de modo que se puedan combinar cada una de ellas con el resto (véase Figura 11).

Cada vez que se selecciona una pareja, se calcula el punto medio de la sección que las conecta (línea que une las dos soluciones), y se calcula el valor de la función objetivo para tal punto medio. Si alguna de las soluciones evaluada tiene mejor valor objetivo que el centro de la especies, entonces éste será el nuevo centro, aunque se mantendrá el mismo radio. Además, si el valor objetivo del punto medio es peor que el de sus correspondientes puntos padres, entonces estos padres se insertan en la *species_list* (véase Figura 11) porque se corresponden con diferentes colinas que tienen un valle entre sí. A cada especie nueva insertada se le asigna el valor del nivel actual *i*.

El parámetro de entrada new_i del procedimiento *Create_species* es un límite superior del número de evaluaciones que se pueden realizar en la creación de las especies. De manera interna éste parámetro se divide entre el número actual de especies en la lista, y de este modo cada especie de la lista tiene un número fijo de evaluaciones para

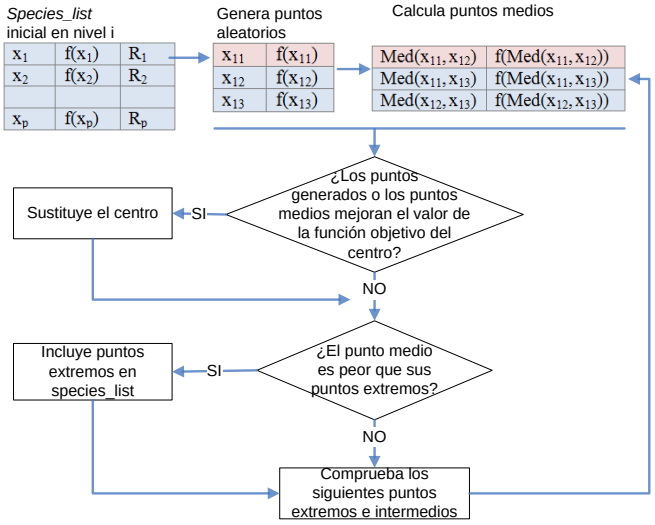

Figura 11. Procedimiento de creación de especies.

la creación de puntos nuevos igual a:

budget_per_species=new$_i$/length(species_list)

Por tanto, el número *NP* de nuevos puntos que pueden ser generados por cada especie está limitado por:

$$NP = \binom{NP}{2} \leq budget\_per\_species \qquad (20)$$

Nótese que por cada pareja de puntos, el punto medio del segmento que conecta la pareja también es evaluado. Desarrollando la ecuación anterior, el número total de nuevos puntos que pueden crearse en cada especie viene dada por:

$$NP = \frac{\sqrt{(1+8\cdot budget\_per\_species)}-1}{2} \qquad (21)$$

De esta manera cuando el número de especies de la lista es pequeño, entonces cada especie puede generar más puntos de prueba, mientras que la cantidad de generación de nuevos puntos será menor cuando la lista crezca.

Procedimiento Fuse_species(radius). En este procedimiento, si los centros de cualquier par de especies, procedentes de la lista de especies, están a una distancia inferior al radio (*radius*) dado, entonces las dos especies se funden en una sola (véase Figura 12). La especie resultante tendrá como centro aquel de las especies fundidas que tenga el mejor valor objetivo, y se le asociará como nivel (y por tanto como radio) el menor de las dos especies (el radio será el mayor). Para niveles bajos la mayoría de especies se fusionan, y para niveles altos (con radio más pequeño) sólo unas pocas especies se fusionarán, y dejarán al descubierto

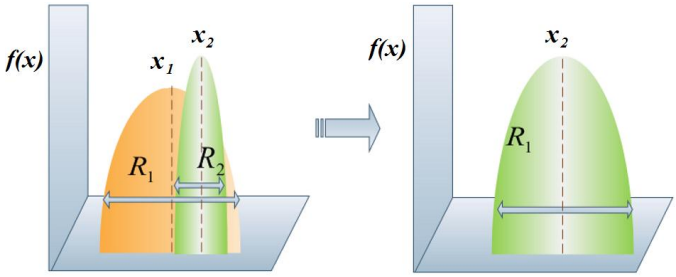

Figura 12. Procedimiento de fusión de especies.

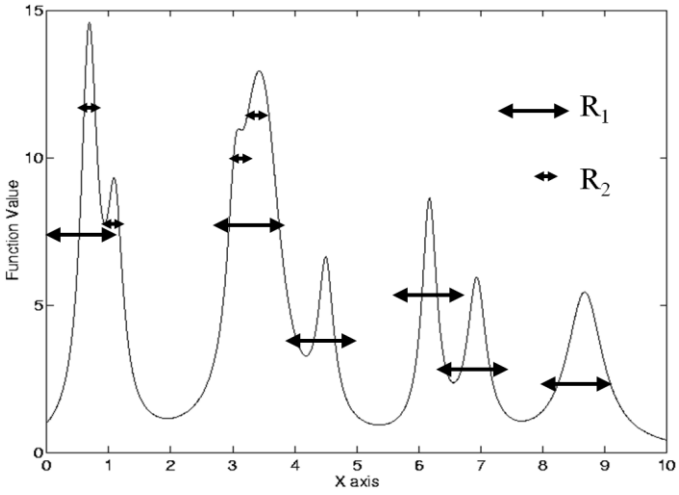

Figura 13. Distribución de especies dependiendo del radio

nuevos óptimos locales (Figura 13).

Procedimiento Shorten_species_list(max_spec_num). Si el número de especies en la lista es más grande que el máximo permitido (*max_spec_num*), se eliminan individuos de la lista hasta alcanzar dicho máximo.

Los individuos con mayor nivel son eliminados antes, guardando así, las subpoblaciones con el radio más grande. Por esta razón una subpoblación del nivel 1, cuyo radio es igual al diámetro del espacio de búsqueda, siempre existe, permitiendo que el algoritmo pueda escapar de óptimos locales. En términos de computación evolutiva se considera un mecanismo de selección donde las especies con radio mayor se eligen primero.

Procedimiento Optimize_species(evals). Este procedimiento es el encargado de optimizar cada una de las especies. La optimización de cada especie tiene asociado un número máximo de evaluaciones de la función objetivo, y toma como punto de partida el centro de la especie, moviéndose en la región definida por su radio (véase Figura 9). Si el proceso de optimización encuentra un punto solución mejor que el centro, entonces este nuevo punto pasa a ser el centro de la especie, de modo que la ventana, y por tanto la región de búsqueda, se desplazan. El radio de la especie se mantiene siempre constante.

En este procedimiento es donde se efectúan las mutaciones de los individuos para intentar producir descendientes con un mejor valor. El criterio de reemplazo es determinista. Se podría decir que el tipo de reemplazo es por inclusión, ya que este procedimiento se va quedando con las mejores soluciones (los centros de las especies). En el caso de que para algún *i* la lista de especies sea menor que el tamaño máximo permitido (*max_spec_num*), el número de evaluaciones debido a la optimización de las especies en el nivel *i* será menor que el calculado a priori n_i (véase Algoritmo 3, *Optimize_species*), por lo que el costo total del algoritmo será dependiente del problema.

III.A. Algoritmo SASS como optimizador local de UEGO

En este trabajo, se utiliza como optimizador local una estrategia de Búsqueda Estocástica con Agente único (Single Agent Stochastic Search) o abreviadamente SASS.

Como método directo, SASS no requiere propiedades especiales (diferenciabilidad, continuidad) de la función objetivo a optimizar por lo que presenta la ventaja de ser fácilmente implementable. Entre los algoritmos basados enestrategias de este tipo se pueden resaltar el algoritmo de McDonnell [31]. Éste propone propone un esquema de optimización híbrida que introduce el método de Solis y Wets [50], como una técnica SASS, dentro del paradigma de programación evolutiva.

El algoritmo de Solis y Wets tal y como lo describe McDonnell [31] se muestra en el Algoritmo 3.

Donde ξ es la varianza del tamaño de la perturbación que

Algoritmo 3: Algoritmo optimizador SASS

Parámetros de entrada: Scnt, Fcnt, ex, ct, σ_{ub}, σ_{lb}, MaxIte
Seleccionar aleatoriamente x_0 #inicializar vector búsqueda
$b_0=0$ #inicializar vector parcial
k=1, scnt=0, fcnt=0, $\sigma_0=1$
MIENTRAS k<MaxIte HACER:
  SI scnt> Scnt ENTONCES $\sigma_k=ex\cdot \sigma_{k-1}$
  SI fcnt> Fcnt ENTONCES $\sigma_k=ct\cdot \sigma_{k-1}$
  SI $\sigma_{k-1}< \sigma_{lb}$ ENTONCES $\sigma_{k=} \underline{\sigma_{ub}}$
  $\xi k=N(bk, \sigma kI)$ # generar vector aleatorio
  # gaussiano multivariado
  SI $\Phi(x_k+\xi_k)< \Phi(x_k)$ ENTONCES
    $x_{k+1}=x_k+\xi_k$
    $b_{k+1}=0.4\xi_k+0.2b_k$
    scnt=scnt+1, fcnt=0
    SI NO
      SI $\Phi(x_k -\xi_k)< \Phi(x_k)< \Phi(x_k+\xi_k)$ ENTONCES
        $x_{k+1}=x_k-\xi_k$
        $b_{k+1}=b_k-0.4\xi_k$
        scnt=scnt+1, fcnt=0
      SI NO
        $x_{k+1}=x_k$
        $b_{k+1}=0.5b_k$
        fcnt=fcnt+1, scnt=0
  k=k+1

se controla mediante el número de éxitos (*scnt*) o fallos (*fcnt*) sucesivos que se producen. Un éxito se produce cuando el nuevo punto calculado tiene un menor valor de la función objetivo Φ, y un fallo se produce en el caso contrario. Los valores numéricos (0.2 y 0.4) que aparecen en la descripción fueron propuestos por Solis y Wets. Los parámetros de usuario del algoritmo son: el número mínimo de éxitos, *Scnt*, y fallos, *Fcnt*, (para aplicar una expansión *ex*, o una contracción *ct*), y los límites superior e inferior (σ_{ub}, σ_{lb}) del valor de la desviación estándar de la perturbación aleatoria σ. McDonnell [31] usa *Scnt* = 5, *Fcnt*= 3, *ex* = 2, *ct* = 0.5, σ_{ub} = 1.0, σ_{lb} = 0.00001. El valor de los coeficientes 0.2 y 0.4 se dejan tal y como lo propusieron Solis y Wets [50]. Notar que las condiciones para modificar σ_k no son mutuamente exclusivas. La desviación estándar σ especifica el tamaño de la esfera que más probablemente contiene el vector de perturbación. El término parcial (bias) *b* localiza el centro de tal esfera basado en la dirección del éxito pasado. En esta formulación, el usuario debe elegir el número máximo de iteraciones como criterio de parada del algoritmo.

En nuestros experimentos se implementó el algoritmo SASS tal y como indica el Algoritmo 3, y se testeó utilizando un conjunto estándar de funciones test. Nuestras primeras pruebas ilustraron la habilidad de SASS para encontrar un mínimo local en unos pocos de cientos de evaluaciones, sin embargo, frecuentemente quedaba atrapado en tales mínimos locales y le resultaba imposible salir de la región de atracción

de tales mínimos. Como el criterio de parada estaba basado en el número de iteraciones, no consideraba si el algoritmo había alcanzado un mínimo global o si se ha quedado atrapado en un mínimo local, nosotros sustituimos este criterio de parada por otro basado en el número máximo de fallos consecutivos *MaxFcnt*, de modo la condición del bucle sería: "while *fcnt* < *MaxFcnt*" en lugar de "while *k* < *MaxIte*". En particular, para nuestros experimentos se hizo *MaxFcnt* = 32.

III.A. Configuración de UEGO

En [45] se hace un análisis de los efectos de los diferentes parámetros con el fin de obtener un conjunto de parámetros robusto, y se proponen algunas guías para ajustar los parámetros dependiendo del problema que se quiere resolver. Aunque en este caso el estudio se centra en el dominio contínuo, las ideas generales pueden extrapolarse al dominio discreto.

Algunas de las principales ideas se muestran a continuación:

El mecanismo de "enfriamiento" de los algoritmos evolutivos aquí tratados, asegura que si el nivel máximo (l) es alto, el algoritmo no quedará atrapado en óptimos locales. Esta propiedad es muy útil, ya que establecer un nivel máximo elevado puede asegurar un grado de robustez mayor. No obstante, tampoco conviene que éste sea extremadamente alto, pues implicaría un gran número de iteraciones en el algoritmo y la convergencia hacia la solución sería más lenta. Lo ideal es que éste oscile entre 10 y 30.

El parámetro M indica el número máximo de subpoblaciones que pueden existir y, por tanto, determina cuántas de ellas pueden evolucionar y optimizarse simultáneamente. En cada nivel, se asocia un presupuesto a cada una de las subpoblaciones que forman la lista: un número de evaluaciones para la creación y optimización de individuos, que serán (new_i / *length(species_list*)) y (n_i / M), respectivamente. Teniendo esto en cuenta, el número de evaluaciones totales invertidas en el proceso de optimización completo, será inferior al máximo permitido (N), si en algún nivel del algoritmo, el número de subpoblaciones es menor que M. Esto puede llegar a ocurrir si el número de óptimos de la función objetivo es inferior a M (sobre todo en las últimas etapas del algoritmo, donde todas las soluciones han convergido hacia los óptimos).

Cuando el número de óptimos de la función es superior o igual a M, la longitud de la lista prácticamente no se reducirá. Esto no significa que se consuman todas las evaluaciones. El presupuesto por subpoblación en el optimizador local (n_i/M), sólo es un límite superior, que se aplicará cuando éste no disponga de otro criterio de parada.

En el caso general, dónde el criterio de parada del optimizador se basa en el número de evaluaciones, el considerar un M relativamente pequeño (M < 10), hace que el número de evaluaciones que cada subpoblación utiliza durante el proceso de optimización sea bastante elevado, lo que implica la ejecución de un gran número de evaluaciones y la consiguiente subida en el tiempo de ejecución. En el caso contrario, un M grande (M > 100), implicaría una reducción del tiempo total. No obstante, para asegurar la convergencia, el presupuesto por subpoblación para la optimización no puede ser muy pequeño, por lo que en este caso deberemos escoger un N mayor. Además, un M elevado aumenta la probabilidad de encontrar el óptimo global, ya que se permite la existencia de varias subpoblaciones repartidas por el espacio de búsqueda.

El parámetro R_L, que indica el tamaño del radio para el último nivel, no ha de ser muy grande. Así, se permite la existencia de subpoblaciones diferentes próximas entre sí y que evolucionan hacia óptimos distintos. Además, la probabilidad de encontrar óptimos cercanos aumenta, ya que el proceso de fusión no las fundirá y las englobará en una sóla subpoblación.

Como conclusión podría decirse que un conjunto de parámetros robustos consiste en un número de niveles (l) elevado, un radio mínimo pequeño (R_L), un máximo número de especies suficiente (M) y un gran valor de N con el fin de obtener el mínimo presupuesto por subpoblación, el cual es su ciente en el proceso de optimización.

Es interesante remarcar que pequeñas modificacions en los diferentes parámetros de entrada no afectan en demasía al comportamiento de los algoritmos. No obstante, la elección de los mismos depende en gran medida de la aplicación que se estudie y de las restricciones en tiempo o en precisión en la solución que se consideren.

IV. ALGORITMO GA DE MATLAB

Global Optimization Toolbox para Matlab R2008a provee métodos de búsqueda de soluciones globales para problemas que contienen múltiples máximos o mínimos, entre ellos un algoritmo genético [33].

Para resolver problemas con varios picos (óptimos locales) obteniendo una única solución global el manual [33] recomienda tres de sus herramientas:

- *simulannealbnd:* el algoritmo de enfriamiento simulado, no se basa en poblaciones por lo que fácilmente se queda atrapado en óptimos locales. A menudo es menos eficiente que el algoritmo genético.
- *patternsearch:*requiere definir el patrón de búsqueda, condición no válida para nuestro objetivo de tratamiento de los problemas como caja negra, cuyo modelo interno nos es desconocido.
- *ga*: algoritmo genético, es válido para problemas de tipo radio niche y es el más eficiente de los tres.

La herramienta GA *(Genetic Algorithm*, Algoritmo Genético en español) está pensada especialmente para la optimización global de problemas no lineales con o sin restricciones. Aporta una gran flexibilidad, ya que además de los múltiples parámetros de los que dispone, es posible personalizar las funciones principales de la herramienta, tales

como la función de selección, de cruce, de mutación y de creación de la población inicial, permitiendo proporcionar funciones hechas a medida para el problema de que se trate. Asimismo permite aplicar una función de búsqueda local, tales como *fmincon* y *patternsearch*, al resultado final obtenido para mejorarlo.

La herramienta GA se puede utilizar mediante la interfaz gráfica del tool de optimización (*optimtool*), o bien mediante la ventana de comandos de Matlab con una llamada a la función *ga()* introduciendo los parámetros de entrada y salida necesarios; esta segunda opción será la empleada en este trabajo. La herramienta GA sirve resolver problemas de la forma $\min f(x)$ sujeto a:

$A \cdot x \le B$, $Aeq \cdot x \le Beq$ (restricciones lineales)

$x \cdot C(x) \le 0$, $Ceq(x) = 0$ (restricciones no lineales)

$A \cdot x \le B$

La función GA requiere un amplio conjunto de parámetros de entrada que incluye la definición del problema y la configuración deseada para el algoritmo genético. Estos parámetros se pueden introducir de forma individual en la llamada al algoritmo o bien incluir un sólo parámetro correspondiente a una variable estructura que englobe al conjunto, esta última opción es más cómoda de manejar si se seleccionan los valores desde la interfaz gráfica y se exporta la variable estructura del problema. A continuación se describen los campos de la estructura problema:

- *objective*: Función objetivo. Es el manejador o *handle* de la función a evaluar, por ejemplo @funcion1 que se corresponde con un fichero funcion1.m con una función predefinida de Matlab o bien definida por el usuario y almacenado en el directorio de trabajo actual o en un directorio añadido a la lista de *path* de Matlab.
- *nvars*: Número de dimensiones de la función objetivo.
- A: Matriz A para restricciones lineales de desigualdad
- B: Vector B para restricciones lineales de desigualdad
- *Aeq*: Matriz A para restricciones lineales de igualdad
- *Beq*: Vector B para restricciones lineales de igualdad
- *lb*: Límite inferior de x
- *ub*: Límite superior de x
- *nonlcon*: Función de restricción no lineal.
- *randstate*: Campo opcional para resetear el estado de rand
- *randnstate*: Campo opcional para resetear el estado de randn
- *solver*: Algoritmo de optimización deseado, en este caso con valor 'ga'.
- *options*: Estructura de opciones concretas para el algoritmo genético creada utilizando la función *gaoptimset()*.

La estructura de opciones propia del algoritmo genético se compone de 33 parámetros, los cuales podríamos clasificar en cuatro subconjuntos: parámetros simples, criterios de parada, funciones de los operadores genéticos, y opciones de visualización. A continuación se describen los parámetros que se han configurado en este trabajo:

- *PopulationSize*: número máximo de individuos en cada generación. [positive scalar]
- *EliteCount*: número de mejores individuos que sobreviven a la siguiente generación sin ningún cambio. [positive scalar | {2}]
- *CrossoverFraction*: fracción de genes intercambiados entre individuos [positive scalar | {0.8}]
- *Generations*: cantidad máxima de generaciones permitidas. [positive scalar | {100}]
- *TimeLimit*: tiempo máximo permitido (en segundos) [positive scalar | {Inf}]
- *StallGenLimit*: máximo número de generaciones seguidas cuyo cambio en el valor de la función objetivo no ha superado TolFun [positive scalar | {50}]
- *StallTimeLimit*: máximo tiempo durante el cual los cambios en el valor de la función objetivo no ha superado TolFun [positive scalar | {20}]
- *TolFun*: tolerancia de finalización en el valor de la función objetivo. [positive scalar | {1e-6}]
- CreationFcn: función usada para generar la población inicial [@gacreationlinearfeasible | @gacreationuniform]
- *SelectionFcn*: función usada para seleccionar padres de la siguiente generación [@selectionremainder | @selectionuniform| @selectionroulette | @selectiontournament | @selectionstochunif]
- *CrossoverFcn*: función usada para realizar la reproducción [@crossoverheuristic | @crossoverintermediate | @crossoversinglepoint | @crossovertwopoint | @crossoverarithmetic | {@crossoverscattered}]
- *MutationFcn*: función usada en la mutación de genes [@mutationuniform | @mutationadaptfeasible |@mutationgaussian]

V. ESTUDIO COMPUTACIONAL

V.A. Descripción del hardware

Todos los resultados computacionales mostrados en este trabajo han sido obtenidos bajo un computador con procesador Intel Core i5 con frecuencia de reloj de 3.33GHz y 4GB de RAM. El sistema operativo es Windows 7 Enterprisa de 32 bits, y el entorno de trabajo es Matlab 7.6 (R2008a).

V.B. Descripción de funciones test

Para medir computacionalmente la eficiencia y eficacia de los algoritmos de optimización estudiados en este trabajo, es necesario ejecutarlos para un conjunto de problemas lo suficientemente amplio, donde se varíe tanto la complejidad como el tamaño de los mismos. A continuación se describen las funciones test utilizadas en este estudio computacional.

1) *unc_n1_sin1*, maximiza:

$$f(x) = 2^{-2\left(\frac{x-0.1}{0.9}\right)^2} \sin^6 5\pi x\ ,\ \forall x \in [0,1]$$

$x^* = 0.1$; $f(x^*)=1$

2) *unc_n1_02*, minimiza*:*

$f(x) = x \cdot \sin(10\pi x) + 2$; ∀x ϵ [-1, 2]

x^*=1.9505; $f(x^*)$= 0.04974;

3) *unc_n1_03*, minimiza:

x≤20 →$f(x)$= -exp(-(x/20)^2);

x>20 →$f(x)$= -exp(-1)+(x-20)*(x-22); ∀x ϵ [-10 , 25]

x^*=21; $f(x^*)$= -1.3679;

4) *unc_n2_rastrigin*, minimiza:

$f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x2)$ ∀x ϵ [-1, 1]2;

x*=(0,0) ; f(x*)= -2;

5) *unc_n2_shubert3*, minimiza:

$$f(x) = \left(\sum_1^5 i\cos[(i+1)x_1 + i]\right) * \left(\sum_1^5 i\cos[(i+1)x_2 + i]\right) + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2$$

∀x ϵ [-10 , 10];

x*=(-1.42513;-0.80032) ; f(x*)=-186.7309;

6) *unc_n2_shubert2*, minimiza:

$$f(x) = \left(\sum_1^5 i\cos[(i+1)x_1 + i]\right) * \left(\sum_1^5 i\cos[(i+1)x_2 + i]\right) + [(x_1 + 1.42513)^2 + (x_2 + 0.80032)^2]/2$$

∀x ϵ [-10, 10];

x*=(-1.42513;-0.80032) ; f(x*)=-186.7309;

7) *unc_nn_Schwefel*

$$f(x) = \sum_{i=1}^n -x_i \sin\left(\sqrt{|x_i|}\right)$$ ∀x ϵ [-500, 500];

x*=(420.9687, ..., 420.9687); f(x*)=-n*418.9829;

8) *unc_nn_step*, maximiza:

$$f(x) = 6n + \sum_{i=1}^n [x_i]$$ ∀x ϵ [1, 10.1];

x*=(9.5~10.1, ..., 9.5~10.1); f(x*)=16*n;

9) *unc_n4_Shekel*, minimiza:

$$f(x) = -\sum\left(\sum_{j=1}^4 (x_i - a_{i,j})^2 + c_i\right)^{-1}$$ ∀x ϵ [0, 10];

x*=(10,0,10,0); f(x*)=-705.5000;

10) *unc_n2_camel3*, minimiza:

$$f(x) = 2x_1^2 - 1.05x_1^4) + \frac{x_1^6}{6} - x_1x_2 + x_2^2;$$ ∀x ϵ [-3, 3];

x*=(0,0); f(x*)= 0;

11) *unc_n2_camel6*, minimiza:

$$f(x) = 4x_1^2 - 2.1x_1^4 + x_1^6/3 + x_1x_2 - 4x_2^2 + 4x_2^4$$

∀x ϵ [-3, 3];

x*=(0.0898, -0.7127) ó (-0.0898, 0.7127);

f(x*)=-1.0316;

12) *unc_n2_treccani*, minimiza:

$f(x) = x_1^4 + 4x_1^3 + 4x_1^2 + x_2^2$ ∀x ϵ [-3, 3];

x*=(0,0) or x*=(-2,0) ; f(x*)= 0;

13) *unc_n2_goldstein*, minimiza:

$$g(x) = 1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)$$

h=30+(2x1-3x2)^2*(18-32x1+12x1^2+48x2-36x1x2+27x2^2);

f=g*h; ∀x ϵ [-3, 3];

x*=(0,-1) ; f(x*)= 3;

14) *unc_n2_branin*, minimiza:

f(x)=(x2-1.275x1^2/ π ^2+5x1/ π -6)^2+10(1-0.125/π)*cos(x1)+10;

∀x ϵ [-5<=x1<=10,0<=x2<=15];

x*= (3.1416 , 2.2750) ,(-3.1416 , 12.2750) or (9.4248 , 2.4750)and ; f(x*)=0.3979;

15) *unc_n2_sin*, minimiza:

$f(x) = (1 - 2x_2 + c\sin 4\pi x_2 - x_1)^2 + (x_2 - 0.5\sin 2\pi x_1)^2$

or (1-2x2+csin(4 π x2)-x1)^2+(x2-0.5sin(2 π x1))^2;

16) *unc_n2_Ackley*, minimiza:

F(x)=-20*exp(-0.2(0.5(x1^2+x2^2))^2)-exp(0.5(cos(2π x1)+cos(2π x2))) +22.71282; ∀x ϵ [-5, 5];

x*=??; f(x*)=???; hay más de 100 óptimos globales.

17) *unc_nn_dejong1*, minimiza:

$$f = \sum_{i=1}^n i * x_i^2$$, ∀x ϵ [-5.12, 5.12];

x*=zeros(n,1); f(x*)= 0;

18) *unc_nn_dejong2*, minimiza:

$$f(x) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j\right)^2$$ ∀x ϵ [-5.12, 5.12];

x*=zeros(n,1); f(x*)= 0;

19) *unc_nn_Griewangk*

$$f(x) = \sum_{i=1}^n \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$,∀x ϵ [-600, 600];

x*=zeros(n,1); f(x*)=0;

20) *unc_nn_dpower*

$$f(x) = \sum_{i=1}^n |x_i|^{i+1}$$, ∀x ϵ [-1, 1];

x*=zeros(n,1); f(x*)=0;

21) unc_nn_sin

$$f(x)=\frac{\pi}{n}\left\{\begin{array}{l}10\sin^2\pi x_1+\sum_{i=1}^{n-1}(x_i-1)^2\left(1+10\sin^2\pi x_{i+1}\right)\\+(x_n-1)^2\end{array}\right\}$$

$\forall x \in [-10, 10]$;

x*=ones(n,1); f(x*)=0;

22) *unc_nn_sin1*, maximiza::

$$f(x)=\frac{\sin\left(\sum_{i=1}^{n}|x_i-5|\right)}{\sum_{i=1}^{n}|x_i-5|} \quad \forall x \in [1, 10];$$

x*=5*ones(n,1), ; f(x*)=1;

El resto de funciones test aparecen definidas en [44]

V.C. Estudio de los parámetros de los algoritmos

UEGO

Se ha realizado un estudio preliminar para determinar un conjunto robusto. El propósito de este estudio es encontrar un conjunto de parámetros del algoritmo UEGO que permita encontrar el valor óptimo de todas las funciones test, o la mayoría, sin incluir ningún análisis sobre los tiempos de ejecución. En concreto se ha empleado una colección de funciones test lo suficientemente heterogénea. Dada la naturaleza estocástica del algoritmo UEGO, cada función test se ha ejecutado un total de 5 veces, con cada configuración de parámetros. Los datos de este estudio preliminar aparecen detallados en el Anexo II.

Haciendo uso de las guías, y de las funciones test seleccionadas se ha llegado a dos buenas configuraciones para la comparación del algoritmo UEGO (véase TABLA II). El primer conjunto robusto es A1 que resultaba exitoso para una mayoría de funciones test en el estudio preliminar, y el conjunto C5 que ha devuelto un éxito casi absoluto para todo el conjunto de entrenamiento.

GA

En el estudio preliminar de este algoritmo se utilizará el mismo conjunto de entrenamiento de funciones test usadas en UEGO con objeto de buscar un conjunto robusto de parámetros. El éxito de la función 8 también sigue los mismos criterios que en UEGO.

Se empieza a analizar los resultados utilizando tres configuraciones de prueba: GA1 es la configuración por defecto del algoritmo; GA2 se parece a la configuración A1 de UEGO, utiliza 10000 generaciones en lugar de evaluaciones y 20 como tamaño de la población en lugar de especies máximas en cada nivel; GA3 sin embargo trata de ser una configuración homóloga a A1 en cuanto a número de evaluaciones. La media de evaluaciones en A1 era inferior a 3000, le damos ventaja al genético dándole 5000 evaluaciones

TABLA II
CONFIGURACIONES DE PARÁMETROS ROBUSTOS PARA UEGO

Config.	N	M	L	r	E(%)
A1	10000	20	10	0,03	69
C5	10000000	300	50	0,003	92

TABLA III
ESTUDIO PRELIMINAR DE PARÁMETROS DE GA (PARTE 1)

Problema	GA1	GA2	GA3
1	40%	60%	20%
2	60%	60%	40%
3	0%	0%	0%
4	40%	40%	0%
5	20%	20%	20%
6	20%	0%	20%
7	0%	0%	0%
8	40%	60%	80%
Éxito (%)	28%	30%	23%

que se dividirían en 250 generaciones de una población de 20 individuos por generación. Los resultados aparecen en la TABLA III.

La configuración GA1, por defecto, no realiza todas las evaluaciones y/o generaciones ofrecidas como parámetros sino que la ejecución termina cuando la media de cambios en el valor de la función objetivo a lo largo de 50 generaciones no supera la tolerancia (criterio de parada StalGenLimit). Las demás configuraciones están configuradas con criterios de parada infinitos por lo que están forzadas a evaluar todas las generaciones indicadas por parámetro. Para ello se utilizó como límite de tiempo (TimeLimit) su valor por defecto, infinito, y como límite de tiempo sin cambios significativos en la solución (StallTimeLimite) una cifra muy superior a la ejecución más lenta que se obtuvo anteriormente en UEGO, en concreto 10000 segundos. El límite de generaciones sin cambios significativos (StallGenLimit) se determinjó con el máximo número de generaciones de su correspondiente configuración. Como factor de tolerancia se utilizó su valor por defecto, una millonésima.

Dado que GA2 es la mejor configuración de las tres iniciales propuestas, se van a intentar ajustar los parámetros utilizando como base ésta. Se han realizado pruebas cambiando los valores de todos los parámetros descritos en el apartado anterior, a continuación se listan los valores probados:

- PopulationSize: 20, 30, 50, 100, 300, 10000.
- Generations: 1000, 10000, 100000.
- EliteCount: 2, 10, 50, 100, 250.
- CrossoverFraction: 0.6, 0.7, 0.8, 0.9.
- CreationFcn, SelectionFcn, CrossoverFcn, y

MutationFcn se han probado todas las opciones y combinaciones entre sí.

En la TABLA IV se muestran los resultados de algunas configuraciones con mayor tasa de éxito. La configuración GB1 viene determinada por una población de 300 individuos y 10000 generaciones. GB2 mantiene la población de 300 y emplea 1000 generaciones. GB3 se basa en los valores de GB2 excepto por el nuevo valor de CrossoverFraction de 0.6. GB4 se basa en GB3 pero con la función de creación lineal. Y por último GB5 coincide con los datos de GB3 pero con una función de Croosver de dos puntos. Esta última configuración será considerada como el conjunto robusto del algoritmo genético para realizar el estudio computacional posterior y la comparación con UEGO.

Las variaciones sobre parámetros como EliteCount, SelectionFcn y MutationFcn no han alterado las tasas de éxito por lo que para el conjunto robusto se mantendrán los valores por defecto.

V.A. Resultados

Dada la natuarelza heurística de los algoritmos analizados, cada una de las funciones test descritas en la subsección V.B se han ejecutado un total de 100 veces por cada configuración, con objeto de poder calcular valores medios representativos y poder inferir conclusiones.

TABLA V
RESUMEN COMPARACIÓN UEGO Y GA

Alg/ Config	Av_evals	Av_t	Éxito (%)	tasa éxito/ Evals (%)	tasa éxito/ tiempo
A1	2627,95	0,68	91,28	3,47	133,54
C5	287618,42	95,57	100,00	0,03	1,05
Ga1	1054,43	0,92	58,78	5,57	63,48
Ga2	200020,00	22,62	72,22	0,04	3,19
Ga3	5020,00	0,66	69,67	1,39	104,66
Gb5	300300,00	28,88	82,39	0,03	2,85

TABLA IV
ESTUDIO PRELIMINAR DE PARÁMETROS DE GA (PARTE 2)

Problema	GB1	GB2	GB3	GB4	GB5
2	100	100	100	100	100
3	0	0	0	0	0
4	100	100	100	100	100
5	40	60	100	20	100
6	80	60	60	60	100
7	0	0	0	0	0
8	100	100	100	100	100
Éxito (%)	52,50	52,50	57,50	47,50	62,50

Para los resultados de una función test con una configuración se guardan los datos en un fichero binario de Matlab, tanto en UEGO como en GA. De cada ejecución se ha guardado un registro con el número de evaluaciones realizadas (*evals*), la lista final de soluciones (*sp*), el tiempo consumido en segundos (*t*), el mejor valor de la función objetivo encontrado (*bestO*), el número de óptimos globales encontrados (*opt*), y la mejor solución dada en coordenadas en el espacio de búsqueda (*bestSol*). Además, en un fichero de texto se guarda la lista de especies finales (coordenadas de la solución y valor de la función objetivo asociado). A modo de resumen, por cada configuración se guarda un archivo de medias, en cada línea o registro aparecen los valores medios de las 100 ejecuciones de una función test. Este fichero guarda el número de la función test (*P*), la media de evaluaciones (*Av_evals*), la media de soluciones de la lista final (*Av_sp*), la media de tiempo (*Av_t*), la media de los valores objetivos (*Av_bestO*), el mínimo valor objetivo encontrado en las 100 ejecuciones (*min_bestO*), el máximo valor objetivo encontrado en las 100 ejecuciones (*max_bestO*), el mejor valor objetivo encontrado en las 100 ejecuciones (*best_bestO*; este valor coincidirá con *max_bestO* en UEGO, y con *min_bestO* en GA), y las coordenadas de la mejor solución o de una de ellas en caso de que existan varios óptimos globales (*best_bestSol*). También se guarda la desviación típica de los valores objetivo de las 100 ejecuciones (*desvT_bestO*) y la media del número de óptimos globales encontrados (*av_opt*). Finalmente, también se calcula el porcentaje de éxito en las 100 ejecuciones (*éxito*). Se computará un éxito si en una ejecución, el algoritmo ha sido capaz de encontrar, al menos, un ópitmo global.

En el Anexo I se muestran los resultados medios de las dos configuraciones de UEGO propuestas, y las cuatro de GA. Para simplificar las tablas de datos se han omitido las columnas de mínimo y máximo valor óptimo, y las coordenadas de la mejor solución encontrada.

A continuación se muestra una tabla de resumen (TABLA V) de los resultados obtenidos. *Av_evals* muestra la media de evaluaciones de todas las funciones test evaluadas. *Av_t* muestra la media de tiempos, *Éxito* sería la media de éxito . La *tasa de éxito/evals* es la proporción de éxito por cada evaluación realizada, dada en porcentaje. Y la tasa *éxito/tiempo* es la proporción de éxito obtenida por unidad de tiempo consumido en la ejecución. La columna *Alg/Config* indica el algoritmo y configuración que representa, A1 y C5 son las dos configuraciones robustas de UEGO y Ga1, Ga2, Ga3 y Gb5 son las configuraciones de GA vistas en el apartado anterior.

Si se busca un algoritmo lo más fiable posible, nos fijaremos en el parámetro *Éxito* únicamente. Según este parámetro Uego con su configuración C5 es fiable 100%, seguido de Uego con su configuración A1, y después la configuración Cb5 del algoritmo genético con el mayor

número de evaluaciones entre las configuraciones vistas.

A menudo se permite sacrificar un poco de fiabilidad para alcanzar un valor aceptable de la proporción *éxito/tiempo*. En este caso el resultado sigue siendo favorable a UEGO, esta vez a su configuración sencilla A1. En segundo lugar se posiciona el algoritmo genético con su configuración Ga3, y después Ga1. Esto se debe a que la configuración C5 de UEGO alcanza la máxima fiabilidad sacrificando un gran número de evaluaciones y tiempo.

Por otro lado si nos fijamos en el número de evaluaciones vemos que Ga1 tiene la máxima tasa de éxito por cada evaluación, seguido de UEGO A1. La configuración C5 de UEGO tiene un número de evaluaciones intermedio entre Ga2 y Gb5 del genético, y sin embargo ninguno de estos dos últimos se acerca a su fiabilidad. Y el número de evaluaciones de A1 es casi la mitad de Ga3 y sin embargo tiene un 20% más de fiabilidad y casi el mismo tiempo consumido de media.

VI. Conclusiones

La implementación del algoritmo UEGO para el entorno Matlab amplía la utilidad de su toolbox ya que este algoritmo ha resultado eficaz para funciones test que no son satisfactoriamente resueltas con los algoritmos actuales del toolbox. Esto es porque el algoritmo genético, el más eficaz del toolbox, a menudo no alcanza el óptimo global al quedar atrapado en óptimos locales. Además ningún algoritmo del toolbox permite encontrar varios óptimos globales, si el problema evaluado los tuviera, mientras que UEGO sí tiene dicha capacidad (es de tipo multimodal).

En el futuro se desea alcanzar la integración total de UEGO en el toolbox de Matlab y aplicar técnicas de aceleración de código como librerías precompiladas para Matlab con objeto de reducir los tiempos de ejecución.

VII. Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación (TIN2008-01117).

VIII. Bibliografía

[1] E. Aarts and P. Laarhoven. Simulated annealing: an introduction. Statistica Neerlandia, 43(1):31{52, 1989.

[2] E. Aarts and J. Lenstra, editors. Local Search in Combinatorial Optimization. John Wiley & Sons, Inc., 1997.

[3] E. Alba. Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience, 2005.

[4] R.S. Andersen. Global optimization. In R.S. Andersen, L.S. Jennings, and D.M. Ryan, editors, Optimization, pages 1{15. University of Queensland, 1972.

[5] D. Beasley, D. Bull, and R. Martin. A sequential niche technique for multimodal function optimization. Evolutionary Computation, 1(2):101{125, 1993.

[6] M. Birattari, L. Paquete, T. Stützle, and K. Varrentrapp. Classi_cation of metaheuristics and design of experiments for the analysis of components. Technical Report AIDA-01-05, Darmstadt University of Technology, Darmstadt, Germany, 2001.

[7] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 35(3):268{308, 2003.

[8] S.H. Brooks. A discussion of random methods for seeking maxima. Operation Research, 6:244{251, 1958.

[9] D. Connolly. An improved annealing scheme for the QAP. European Journal of Operational Research, 46:93{100, 1990.

[10] C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. AI Communications, 11(3-4):223{224, 1998.

[11] P. Darwen and X. Yao. Speciation as automatic categorical modularization. IEEE Transactions on Evolutionary Computation, 1(2):100{108, 1997.

[12] Y. Davidor. A naturally occurring niche and species phenomenon: The model and _rst results. In R. Belew and L. Booker, editors, Proceedings of the 4th International Conference on Genetic Algorithms, pages 257{263. Morgan Kaufmann, 1991.

[13] R. Dawkins. The Sel_sh Gene. Clarendon Press, Oxford, 1976.

[14] E. Dijkstra. A note on two problems in connection with graphs. In Numerishe Math, pages 269{271, 1959.

[15] M. Dorigo. Ant Colony Optimization. Scholarpedia, 2007.

[16] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization, pages 11{32. McGraw-Hill, London, 1999.

[17] G. Dueck. New optimization heuristics. Journal of Computational Physics, 104:86{92, 1993.

[18] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Jounal of Computational Physics, 90:161{175, 1990.

[19] M. Fleischer. Simulated annealing: past, present, and future. In WSC '95: Proceedings of the 27th conference on Winter simulation, pages 155{161. IEEE Computer Society, 1995.

[20] S. Forrest, B. Javornik, R. Smith, and A. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. Journal of Evolutionary Computation, 1(3):191{211, 1993.

[21] F. Glover. Tabu search{part I. ORSA J. on Computing, 1(1):190{206, 1989.

[22] F. Glover. Tabu search{part II. ORSA J. on Computing, 1(2):4{32, 1990.

[23] D. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, New York, 1989.

[24] J. Hu and E. Goodman. Robust and efficient genetic algorithms with hierarchical niching and a sustainable evolutionary computation model. In K. D. et al., editor, Genetic and Evolutionary Computation{GECCO 2004.

Proceedings of the Genetic and Evolutionary Computation Conference. Part I, volume 3102 of Lecture Notes in Computer Science, pages 1220{1232. Springer-Verlag, 2004.

[25] L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. Control and Cybernetics, 25(1):33{54, 1996.

[26] M. Jelasity. The shape of evolutionary search: Discovering and representing search space structure. PhD thesis, Leiden University, January 2001.

[27] A. R. Kan and G. Timmer. Stochastic global optimization methods; part I: clustering methods. Mathematical Programming, 39(1):27{56, 1987.

[28] A. R. Kan and G. Timmer. Stochastic global optimization methods; part II: multi level methods. Mathematical Programming, 39(1):57{78, 1987.

[29] P. V. Laarhoven, E. Aarts, and J. Lenstra. Job shop scheduling by simulated annealing. Operational Research, 40:113{125, 1992. [68] J.-P. Li, M. Balazs, G. Parks, and P. Clarkson. A species conserving genetic algorithm for multimodal function optimization. Evolutionary Computation, 10(3):207{234, 2002.

[30] S. Louis, X. Yin, and Z. Yuan. Multiple vehicle routing with time windows using genetic algorithms. In P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, Proceedings of the Congress on Evolutionary Computation, volume 3, pages 1804{1808. IEEE Press, 6-9 1999.

[31] J.R. McDonnell and D. Waagen. Evolving recurrent perceptrons for time-series modeling. IEEE Trans. on Neural Network, 5(1):24{38, January 1994.

[32] S. Mahfoud. Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[33] MathWorks, Inc. *Global Optimization Toolbox User's Guide*. *EE.UU., 2012.*

[34] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. Journal of Chemical Physics, 21:1087{1092, 1953.

[35] Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 1994.

[36] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, California Institute of Technology, Pasadena, CA, 1989.

[37] P. Moscato. On genetic crossover operators for relative order preservation. Technical Report C3P-778, Caltech Concurrent Computation Program, 1989.

[38] P. Moscato and C. Cotta. Memetic algorithms. http://www.lcc.uma.es/ ccottap/papers/memetic_HAAM.pdf.

[39] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, pages 105{144. Kluwer Academic Publishers, Boston MA, 2003.

[40] P. Moscato and C. Cotta. An introduction to memetic algorithms. Inteligencia Arti_cial, Revista Iberoamericana de Inteligencia Arti_cial., 19:131{148, 2003.

[41] P. Moscato, C. Cotta, and A. Mendes. Memetic algorithms. In G. Onwubolu and B. Babu, editors, New Optimization Techniques in Engineering, volume 141 of Studies in Fuzziness and Soft Computing, chapter 3, pages 53{86. Springer, 2004.

[42] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. In Acta Numerica 2004, pages 271{369. Cambridge University Press, 2004.

[43] K. Nurmela. Constructing combinatorial designs by local search. Series A 27, Helsinki University of Technology, FINLAND, November 1993.

[44] P.M. Ortigosa. Métodos Estocásticos de Optimización Global. Procesamiento Paralelo. PhD thesis, Universidad de Málaga, 1999.

[45] P. Ortigosa, I. Garc__a, and M. Jelasity. Reliability and performance of UEGO, a clustering-based global optimizer. Journal of Global Optimization, 19(3):265{289, 2001.

[46] R. Prim. Shortest connection networks and some generalisations. In Bell System Technical Journal, pages 1389{1401, 1957.

[47] H. Ratschek and J. Rokne. New computer methods for global optimization. Halsted Press, New York, NY, USA, 1988.

[48] J. Redondo. Solving competitive location problems via memetic algorithms. High performance computing approaches. Universidad de Almería, 2009.

[49] B. Sareni and L. Kr□ahenb□uhl. Fitness sharing and niching methods revisited. IEEE Transaction on Evolutionary Computation, 2(3):97{106, 1998.

[50] F.J. Solis and B. Wets. Minimization by random search techniques. Mathematics of Operations Research, 6(1):19{50, 1981.

[51] P. Surry and N. Radcli_e. Inoculation to initialise evolutionary search. In T. Fogarty, editor, Evolutionary Computing, Proceedings of the AISB96 Workshop, pages 260{276, 1996.

[52] A. T_orn and A. Zilinskas. Global optimization. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[53] L. Yu, K. Liu, and K. Li. Ant colony optimization in continuous problem. Frontiers of Mechanical Engineering in China, 2(4):459{462, 2007.

IX. ANEXO 1

En este anexto se muestran de manera detallada los resultados obtenidos por los algoritmos analizados. En concreto, la Tabla VI muestra los resultados medios obtenidos por UEGO con la configuración de parámetros A1; la Tabla VII, …. La notación usada en tales tablas se corresponde con la especificada en la Sección ¿? (pon el número de sección).

TABLA VI
RESULTADOS MEDIOS DE 100 EJECUCIONES CON UEGO, CONFIGURACIÓN A1

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	2269,50	5,00	0,54	1,00	1,00	1,95E-09	1,00	100,00
2	2907,40	8,60	0,72	-0,05	-0,05	2,00E-02	0,99	99,00
3	1810,56	3,00	0,45	1,37	1,37	3,90E-09	1,00	100,00
4	3895,41	12,20	1,02	1,99	2,00	3,48E-02	0,91	91,00
5	4296,81	13,60	1,20	185,22	186,73	6,73E+00	0,43	43,00
6	4679,59	15,20	1,35	184,69	186,73	5,38E+00	0,40	40,00
7	5203,44	18,40	1,65	1129,34	1256,95	1,03E+02	0,34	31,00
8	2503,60	4,60	0,70	48,00	48,00	0,00E+00	3,20	100,00
9	1084,80	2,00	0,31	8,11	10,15	2,79E+00	0,60	60,00
10	1084,00	2,00	0,26	0,00	0,00	6,87E-10	1,00	100,00
11	1190,20	2,20	0,29	1,03	1,03	2,54E-09	1,40	100,00
12	1317,24	2,00	0,31	0,00	0,00	1,36E-09	2,00	100,00
13	2524,80	4,20	0,64	-3,00	-3,00	5,99E-07	1,00	100,00
14	2927,38	4,40	0,77	-0,40	-0,40	2,41E-09	3,00	100,00
15	2298,00	5,80	0,60	0,00	0,00	1,87E-08	1,80	100,00
16	3904,60	13,40	1,06	0,00	0,01	8,59E-02	0,99	99,00
17	785,20	1,00	0,19	0,00	0,00	2,12E-08	1,00	100,00
18	913,60	1,00	0,24	0,00	0,00	4,82E-07	1,00	100,00
19	973,00	1,00	0,26	0,00	0,00	2,41E-06	1,00	100,00
20	8290,77	20,00	2,51	0,00	0,00	5,43E-09	9,34	100,00
21	3225,94	2,20	0,72	1,00	1,00	7,96E-13	2,00	100,00
22	2124,09	6,00	0,48	3,32	3,32	1,33E-05	0,84	65,00
23	2650,94	3,00	0,60	4,60	4,60	1,04E-10	1,00	100,00
24	3381,24	3,60	0,80	0,82	0,82	3,64E-11	1,00	100,00
25	2386,05	7,00	0,57	14,59	14,59	1,56E-07	1,00	100,00
26	3123,11	11,20	0,79	12,03	12,03	1,09E-08	2,92	100,00
27	2595,43	4,00	0,61	7,92	7,92	7,00E-11	2,00	100,00
28	743,40	1,20	0,16	-0,01	-0,01	5,47E-08	1,07	100,00
29	973,00	1,00	0,22	0,00	0,00	0,00E+00	1,00	100,00
30	2203,91	2,20	0,54	0,00	0,00	5,85E-09	2,00	100,00
31	1654,40	3,20	0,37	1,03	1,03	2,61E-09	1,80	100,00
32	881,80	1,40	0,19	0,00	0,00	1,45E-08	1,00	100,00
33	5256,64	17,40	1,49	176,54	176,54	3,66E-06	2,52	94,00
34	4534,71	10,40	1,08	0,00	0,00	8,38E-06	0,91	91,00
35	3219,06	5,20	0,74	0,00	0,00	2,44E-03	0,73	73,00
36	792,60	1,00	0,18	0,00	0,00	4,54E-08	1,00	100,00
Media	2627,95	6,10	0,68			3,29E+00	1,53	91,28

TABLA VII
RESULTADOS MEDIOS DE 100 EJECUCIONES CON UEGO, CONFIGURACIÓN C5

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	111894,42	5,00	26,17	1,00	1,00	2,73E-13	1,00	100
2	148969,39	16,00	38,01	-0,05	-0,05	9,06E-12	1,00	100
3	348604,38	9,00	82,58	1,37	1,37	9,46E-15	1,00	100
4	300811,76	47,40	82,72	2,00	2,00	3,87E-10	1,00	100
5	249348,72	156,00	141,89	186,73	186,73	3,00E-06	1,00	100
6	257483,66	157,60	155,38	186,73	186,73	3,14E-06	1,00	100
7	997347,77	161,00	397,12	1256,95	1256,95	1,51E-06	1,16	100
8	877705,47	27,40	245,54	48,00	48,00	0,00E+00	22,36	100
9	68760,96	5,00	17,27	10,15	10,15	2,23E-07	1,00	100
10	138778,54	3,00	30,88	0,00	0,00	4,71E-13	1,00	100
11	93940,15	5,80	20,35	1,03	1,03	1,68E-12	2,00	100
12	155628,79	2,00	35,30	0,00	0,00	2,14E-13	2,00	100
13	242533,55	4,00	56,21	-3,00	-3,00	1,00E-10	1,00	100
14	553720,77	8,80	135,60	-0,40	-0,40	1,98E-13	3,00	100
15	129148,75	27,40	32,59	0,00	0,00	1,91E-11	3,81	100
16	403130,78	93,20	137,31	0,01	0,01	4,16E-09	1,00	100
17	44163,60	1,00	8,02	0,00	0,00	3,33E-09	1,00	100
18	44301,60	1,00	8,52	0,00	0,00	1,72E-08	1,00	100
19	46662,20	1,00	8,61	0,00	0,00	3,00E-08	1,00	100
20	495224,11	300,00	576,53	0,00	0,00	1,46E-11	300,00	100
21	433539,82	3,00	94,64	1,00	1,00	0,00E+00	2,00	100
22	288187,23	9,00	63,52	3,32	3,32	3,28E-08	2,00	100
23	195501,55	3,00	41,40	4,60	4,60	4,61E-15	1,00	100
24	780884,54	4,00	172,63	0,82	0,82	2,07E-16	1,00	100
25	200783,50	8,00	45,58	14,59	14,59	1,06E-12	1,00	100
26	307837,53	20,00	75,65	12,03	12,03	1,09E-13	3,00	100
27	376685,85	4,00	82,59	7,92	7,92	1,08E-14	2,00	100
28	52449,72	2,00	8,69	-0,01	-0,01	2,38E-11	2,00	100
29	75787,00	1,00	14,11	0,00	0,00	0,00E+00	1,00	100
30	486319,03	2,00	115,09	0,00	0,00	8,35E-14	2,00	100
31	107142,12	6,00	21,54	1,03	1,03	1,45E-12	2,00	100
32	88759,47	3,00	17,26	0,00	0,00	1,35E-12	1,00	100
33	260259,55	157,60	166,26	176,54	176,54	2,08E-07	8,89	100
34	361394,55	151,80	150,18	0,00	0,00	1,20E-08	1,00	100
35	586391,04	20,20	127,98	0,00	0,00	7,83E-10	1,01	100
36	44181,20	1,00	6,94	0,00	0,00	2,40E-10	1,00	100
Media	287618,42	39,64	95,57			2,27E-07	10,53	100

TABLA VIII
RESULTADOS MEDIOS DE 100 EJECUCIONES CON GA, CONFIGURACIÓN GA1

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	1040	20	3,73	-0,96	-1,00	6,57E-02	0,54	54
2	1040	20	2,13	0,29	0,05	3,33E-01	0,49	49
3	1040	20	2,25	-1,00	-1,00	2,68E-12	0	0
4	1040	20	0,52	-1,84	-2,00	1,26E-01	0,19	19
5	1040	20	0,52	-103,00	-186,73	5,22E+01	0,07	7
6	1040	20	0,50	-108,53	-186,73	4,96E+01	0,06	6
7	1041	20	0,36	-11,84	-11,84	1,67E-04	0	0
8	1040	20	0,32	-47,17	-48,00	1,01E+00	0,49	49
9	1060	20	0,34	-5,05	-5,06	1,61E-03	0	0
10	1040	20	0,31	0,00	0,00	7,37E-07	1	100
11	1040	20	0,31	-1,03	-1,03	2,57E-06	1	100
12	1040	20	0,31	0,00	0,00	1,20E-06	0,99	99
13	1165	20	0,32	17,80	3,00	2,76E+01	0,53	53
14	1040	20	0,74	0,40	0,40	3,24E-06	0,99	99
15	1040	20	1,62	0,00	0,00	2,81E-06	0,96	96
16	1040	20	4,66	0,03	-0,01	1,69E-01	0,95	95
17	1042	20	2,45	0,00	0,00	1,45E-04	0,82	82
18	1228	20	0,52	0,00	0,00	1,67E-03	0	0
19	1051	20	0,51	0,00	0,00	1,29E-03	0,49	49
20	1060	20	0,50	0,00	0,00	3,10E-03	0,4	40
21	1040	20	0,53	-1,00	-1,00	6,13E-10	1	100
22	1040	20	0,56	-3,25	-3,32	2,51E-01	0,04	4
23	1040	20	0,50	-4,59	-4,60	5,60E-02	0,97	97
24	1040	20	0,52	-0,82	-0,82	2,08E-09	1	100
25	1040	20	0,50	-14,59	-14,59	5,80E-07	1	100
26	1040	20	0,66	-11,48	-12,03	2,03E+00	0,92	92
27	1040	20	0,80	-1,82	-1,82	1,30E-09	0	0
28	1040	20	0,87	0,01	0,01	5,74E-08	1	100
29	1040	20	1,01	0,00	0,00	3,75E-09	1	100
30	1059	20	0,55	0,00	0,00	1,15E-04	0,68	68
31	1040	20	0,54	-1,03	-1,03	5,06E-07	1	100
32	1040	20	0,78	0,00	0,00	2,58E-08	1	100
33	1041	20	0,63	-92,98	-176,54	5,39E+01	0,04	4
34	1041	20	0,62	0,03	0,00	5,06E-02	0,64	64
35	1153	20	0,84	0,03	0,00	1,99E-01	0,12	12
36	1057	20	0,49	0,00	0,00	2,30E-04	0,78	78
Media	1054,43	20	0,93			5,21	0,59	58,78

TABLA IX
RESULTADOS MEDIOS DE 100 EJECUCIONES CON GA, CONFIGURACIÓN GA2

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	200020	20	20,76	-0,97	-1,00	5,26E-02	0,58	58
2	200020	20	20,60	0,16	0,05	1,71E-01	0,63	63
3	200020	20	18,51	-1,00	-1,00	2,86E-12	0	0
4	200020	20	20,51	-1,89	-2,00	1,11E-01	0,32	32
5	200020	20	25,96	-142,03	-186,73	3,82E+01	0,11	11
6	200020	20	22,69	-143,92	-186,73	4,14E+01	0,19	19
7	200020	20	24,70	-11,84	-11,84	8,88E-14	0	0
8	200020	20	24,33	-46,89	-48,00	1,19E+00	0,42	42
9	200020	20	32,03	-5,06	-5,06	1,33E-10	0	0
10	200020	20	23,13	0,00	0,00	1,39E-14	1	100
11	200020	20	23,48	-1,03	-1,03	3,28E-14	1	100
12	200020	20	22,85	0,00	0,00	2,24E-14	1	100
13	200020	20	22,11	9,21	3,00	1,87E+01	0,87	87
14	200020	20	25,84	0,40	0,40	1,07E-14	1	100
15	200020	20	23,60	0,00	0,00	2,05E-12	1	100
16	200020	20	22,70	0,00	-0,01	8,59E-02	0,99	99
17	200020	20	21,62	0,00	0,00	1,11E-12	1	100
18	200020	20	23,84	0,00	0,00	2,32E-11	1	100
19	200020	20	23,17	0,00	0,00	6,15E-15	1	100
20	200020	20	21,80	0,00	0,00	2,93E-11	1	100
21	200020	20	22,17	-1,00	-1,00	5,37E-10	1	100
22	200020	20	18,94	-3,32	-3,32	5,29E-05	0,04	4
23	200020	20	22,23	-4,59	-4,60	4,59E-02	0,98	98
24	200020	20	19,87	-0,82	-0,82	2,01E-09	1	100
25	200020	20	22,95	-14,59	-14,59	5,72E-07	1	100
26	200020	20	23,21	-12,03	-12,03	1,71E-07	1	100
27	200020	20	22,00	-1,82	-1,82	1,48E-09	0	0
28	200020	20	18,41	0,01	0,01	5,00E-08	1	100
29	200020	20	19,54	0,00	0,00	5,36E-17	1	100
30	200020	20	20,36	0,00	0,00	7,64E-15	1	100
31	200020	20	25,95	-1,03	-1,03	5,75E-14	1	100
32	200020	20	23,46	0,00	0,00	1,89E-14	1	100
33	200020	20	25,54	-127,78	-176,54	4,48E+01	0,13	13
34	200020	20	20,38	0,03	0,00	4,84E-02	0,74	74
35	200020	20	20,46	0,00	0,00	6,26E-12	1	100
36	200020	20	24,64	0,00	0,00	1,47E-13	1	100
Media	200020	20	22,62	-14,22	-18,21	4,02E+00	0,722	72,22

TABLA X
RESULTADOS MEDIOS DE 100 EJECUCIONES CON GA, CONFIGURACIÓN GA3

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	5020	20	0,65	-0,94	-1,00	8,48E-02	0,46	46
2	5020	20	0,63	0,26	0,05	2,97E-01	0,51	51
3	5020	20	0,63	-1,00	-1,00	2,83E-12	0	0
4	5020	20	0,67	-1,86	-2,00	1,07E-01	0,21	21
5	5020	20	0,71	-113,76	-186,73	5,22E+01	0,08	8
6	5020	20	0,71	-117,09	-186,73	4,84E+01	0,1	10
7	5020	20	0,69	-11,84	-11,84	6,70E-11	0	0
8	5020	20	0,70	-47,15	-48,00	1,10E+00	0,5	50
9	5020	20	0,82	-5,06	-5,06	2,22E-08	0	0
10	5020	20	0,68	0,00	0,00	2,45E-10	1	100
11	5020	20	0,68	-1,03	-1,03	6,54E-10	1	100
12	5020	20	0,68	0,00	0,00	1,83E-10	1	100
13	5020	20	0,67	21,12	3,00	3,27E+01	0,73	73
14	5020	20	0,69	0,40	0,40	2,87E-10	1	100
15	5020	20	0,68	0,00	0,00	2,72E-02	0,98	98
16	5020	20	0,67	0,00	-0,01	8,59E-02	0,99	99
17	5020	20	0,70	0,00	0,00	6,68E-10	1	100
18	5020	20	0,71	0,00	0,00	1,12E-08	1	100
19	5020	20	0,71	0,00	0,00	3,60E-08	1	100
20	5020	20	0,68	0,00	0,00	3,71E-04	0,92	92
21	5020	20	0,62	-1,00	-1,00	5,16E-10	1	100
22	5020	20	0,62	-3,28	-3,32	1,92E-01	0,1	10
23	5020	20	0,62	-4,59	-4,60	7,15E-02	0,95	95
24	5020	20	0,63	-0,82	-0,82	1,82E-09	1	100
25	5020	20	0,65	-14,58	-14,59	1,68E-01	0,99	99
26	5020	20	0,65	-11,81	-12,03	1,23E+00	0,96	96
27	5020	20	0,62	-1,82	-1,82	1,51E-09	0	0
28	5020	20	0,62	0,01	0,01	5,42E-08	1	100
29	5020	20	0,63	0,00	0,00	1,58E-09	1	100
30	5020	20	0,65	0,00	0,00	3,79E-10	1	100
31	5020	20	0,66	-1,03	-1,03	5,90E-10	1	100
32	5020	20	0,66	0,00	0,00	1,71E-10	1	100
33	5020	20	0,67	-93,06	-176,54	5,56E+01	0,03	3
34	5020	20	0,64	0,04	0,00	5,23E-02	0,66	66
35	5020	20	0,64	0,00	0,00	5,50E-04	0,91	91
36	5020	20	0,64	0,00	0,00	6,52E-10	1	100
Media	5020	20	0,67	-11,39	-18,21	5,34E+00	0,70	69,67

TABLA XI
RESULTADOS MEDIOS DE 100 EJECUCIONES CON GA, CONFIGURACIÓN GB5

P	Av_evals	Av_sp	Av_t	Av_bestO	Best_bestO	DesvT_bestO	Av_opt	Éxito %
1	300300	300	27,68	-1,00	-1,00	7,56E-07	1	100
2	300300	300	25,27	0,05	0,05	1,03E-06	1	100
3	300300	300	26,50	-1,00	-1,00	2,38E-12	0	0
4	300300	300	29,55	-2,00	-2,00	3,67E-13	1	100
5	300300	300	29,47	-186,33	-186,73	3,92E-01	0,48	48
6	300300	300	30,58	-186,53	-186,73	1,96E-01	0,49	49
7	300300	300	31,21	-11,84	-11,84	2,93E-14	0	0
8	300300	300	28,94	-48,00	-48,00	0,00E+00	1	100
9	300300	300	38,13	-5,06	-5,06	2,21E-11	0	0
10	300300	300	28,51	0,00	0,00	6,94E-15	1	100
11	300300	300	30,28	-1,03	-1,03	1,58E-14	1	100
12	300300	300	30,80	0,00	0,00	7,65E-15	1	100
13	300300	300	29,66	3,00	3,00	1,22E-12	1	100
14	300300	300	28,83	0,40	0,40	5,30E-15	1	100
15	300300	300	30,00	0,00	0,00	2,03E-12	1	100
16	300300	300	29,25	-0,01	-0,01	5,86E-14	1	100
17	300300	300	28,94	0,00	0,00	4,38E-13	1	100
18	300300	300	32,35	0,00	0,00	1,01E-11	1	100
19	300300	300	29,93	0,00	0,00	2,42E-15	1	100
20	300300	300	31,06	0,00	0,00	1,29E-11	1	100
21	300300	300	24,77	-1,00	-1,00	4,43E-10	1	100
22	300300	300	27,03	-3,32	-3,32	4,97E-05	0,09	9
23	300300	300	24,99	-4,60	-4,60	5,76E-09	1	100
24	300300	300	25,11	-0,82	-0,82	1,59E-09	1	100
25	300300	300	28,69	-14,59	-14,59	4,27E-07	1	100
26	300300	300	28,53	-12,03	-12,03	1,32E-07	1	100
27	300300	300	26,50	-1,82	-1,82	1,19E-09	0	0
28	300300	300	28,10	0,01	0,01	4,71E-08	1	100
29	300300	300	27,69	0,00	0,00	5,61E-17	1	100
30	300300	300	28,87	0,00	0,00	1,78E-15	1	100
31	300300	300	28,29	-1,03	-1,03	1,30E-14	1	100
32	300300	300	28,80	0,00	0,00	4,19E-15	1	100
33	300300	300	30,51	-164,12	-176,54	1,53E+01	0,6	60
34	300300	300	29,25	0,00	0,00	1,47E-13	1	100
35	300300	300	27,48	0,00	0,00	3,18E-12	1	100
36	300300	300	28,42	0,00	0,00	6,11E-14	1	100
Media	300300	300	28,89	-17,85	-18,21	4,41E-01	0,82	82,39

X. ANEXO 2

En la primera ronda de experimentación se evaluaron las 11 funciones test propuestas con tres configuraciones distintas de parámetros, basadas en el conjunto robusto propuesto en [44]. En concreto se eligieron los valores: L=10 niveles, *M*=20 especies como máximo en la lista, min_r=0.03 de radio asociado al último nivel, y el número máximo de evaluaciones variable *N*={10000, 50000, 2000} correspondiente a las tres configuraciones A1, A2 y A3 respectivamente. En la TABLA XII aparecen los resultados. El número de evaluaciones máximo, *N*, es el primer parámetro en analizar pues es el más común entre los algoritmos de búsqueda, desde el Pure Random Search [4][8]. Puede apreciarse que al aumentar únicamente el valor de *N* en A2, sin modificar el resto de parámetros, empeora la media de éxito. Del mismo modo, al disminuir N en la configuración A3 se reduce la media de éxito debido a la reducción del número de especies que pueden crearse en cada nivel para explorar el espacio del búqueda.

TABLA XII
ESTUDIO PRELIMINAR DE PARÁMETROS DE UEGO (PARTE 1)

Problema	A1	A2	A3
1	100%	100%	100%
2	100%	100%	40%
3	100%	100%	100%
4	80%	80%	100%
5	80%	60%	20%
6	80%	80%	40%
7	20%	0%	0%
8	100%	100%	100%
Éxito (%)	76%	74%	60%

Con los datos anteriores se puede deducir que el número de evaluaciones no puede ser considerado un índice representativo de la cantidad de exploración que se realiza y de la probabilidad de encontrar el óptimo (como en PRS[3j][8]), sino que para mejorar la calidad de la solución se requiere la combinación del resto de parámetros. Por ello, en la segunda ronda de experimentación preliminar, se probarán 7 configuraciones distintas de los 4 parámetros de UEGO. Se va a evaluar las tres funciones test complejas, problemas P={7, 7^4}, donde 7 es la función de Schwefel con 3 dimensiones, y 7^4 la misma con 4. Las siete configuraciones empiezan con una variación B1 del conjunto A1, y en las sucesivas configuraciones B_i va aumentando la capacidad de uno de los parámetros. B1 es A1 pero aumentando a 50 el número de especies, M. B2 se basa en B1 aumentando el número de evaluaciones N a 100 mil. B3 se basa en B2 aumentando el número de niveles a 20. B4 vuelve a aumentar M, 150. B5 incrementa N=1 millón. B6 L=50. Y por último B7 que aumenta r=0.05 sobre la configuración B5, no sobre su antecesor B6. Los resultados mostrados en la TABLA XIII revelan como mejor configuración de las probadas la B6 correspondiente a los valores M=150 especies, N= 10^6 millón máximo de evaluaciones, N=50 niveles de ejecución, R=0.03 de radio mínimo.

TABLA XIII
ESTUDIO PRELIMINAR DE PARÁMETROS DE UEGO (PARTE 2)

P	B1	B2	B3	B4	B5	B6	B7
7	20	20	40	80	80	80	80
7^4	0	0	0	0	40	60	40
E(%)	10	10	20	40	60	70	60

Los escalón de mejora más pronunciado en la sucesión de incrementos de las configuraciones Bi se da en B6 al multiplicar por 2.5 la cantidad de niveles L. Un aumento del radio mínimo en la configuración 7, correspondiente al área de búsqueda del último nivel del algoritmo, ha provocado una disminución de la media de éxito. Esto denota que la calidad de la solución se relaciona de forma inversamente proporcional al radio mínimo, al contrario que el resto de parámetros. Un radio mínimo mayor implica espacios de búsqueda mayores en los procesos de optimización de las especies, posibilidad de optimizar con saltos mayores en el espacio si se encuentra punto mejor pero también disminuye la probabilidad de encontrar ese punto mejor. Hay funciones con picos de la función objetivo muy agudos que se ven beneficiadas por saltos mayores de la optimización (radio mínimo mayor), sin embargo hay funciones como la 7 con colinas/valles muy suaves que necesitan un radio mínimo pequeño para optimizar en cada ocasión aunque eso suponga saltos más pequeños.

La configuración B6 alcanza un éxito mayor al 50% en las dos funciones test complejas evaluadas, por lo que se considera una buena candidata como configuración robusta B6. En la tercera ronda de experimentación se probarán cinco configuraciones distintas derivadas del conjunto B6, tanto aumentando las capacidades de los parámetros para mejorar la tasa de éxito, como disminuyéndolas para ver hasta qué punto se puede reducir la capacidad manteniendo el mismo éxito. Las configuraciones C1, C2, y C3 derivan de B6 disminuyendo los parámetros, en concreto {M=100 y L=40}, {N=500}, y {R=0.003} respectivamente. C4 deriva de C3 con N=5 millones, y C5 también deriva de C3 con N=10 millones, M=300 especies.

En la Tabla TABLA XIV aparecen los resultados de esta ronda de experimentación. C1 empeoraba el número de especies y los niveles y ha reducido un 10% el éxito con respecto B6. C2 empeora el número de evaluaciones y mantiene el

TABLA XIV
ESTUDIO PRELIMINAR DE PARÁMETROS DE UEGO
(PARTE 3)

P	C1	C2	C3	C4	C5
7	80	100	100	100	80
7^4	40	40	40	60	100
E(%)	60	70	70	80	90

mismo éxito que su antecesor B6. En la Tabla TABLA XIV aparecen los resultados de esta ronda de experimentación. C1 empeoraba el número de especies y los niveles y ha reducido un 10% el éxito con respecto B6. C2 empeora el número de evaluaciones y mantiene el mismo éxito que su antecesor B6. C3 mejoraba el radio mínimo con respecto B6 y ha mejorado un 8% la tasa media de éxito. C4 mejora el número de evaluaciones con respecto C3 y aumenta el éxito. Y por último C5 aumenta el número de evaluaciones y el número de especies con respecto C3 y alcanza una tasa de éxito del 92%, convirtiéndose así en una configuración casi perfecta para este conjunto de entrenamiento.

En la cuarta y última ronda de experimentación preliminar se tiene por objeto refinar el conjunto de parámetros C5, hasta el momento el más robusto, de manera que reduciéndose las capacidades de los parámetros se mantenga la tasa de éxito. Los conjuntos de Di para i entre 1 y 5, derivan de C5 disminuyendo N=5 millones, M=200 especies, L=40 niveles, radio mínimo = 0.004, y radio mínimo = 0.0003 respectivamente.

TABLA XV
ESTUDIO PRELIMINAR DE PARÁMETROS DE UEGO
(PARTE 3)

P	D1	D2	D3	D4	D5
7	100	100	100	100	100
7^4	80	60	0	80	40
E(%)	89,33	84,00	61,33	88,00	80,00

Todas las derivaciones de C5 han reducido la tasa de éxito por lo que se concluye que la configuración C5 será nuestro conjunto robusto para la posterior etapa de evaluación. Esta configuración ha devuelto una tasa media del 100% de éxito para el resto de funciones test del conjunto de entrenamiento.