Implementación de un algoritmo genético distribuido para optimización de problemas reales

Ignacio Robles†, José Manuel Benítez⋆, Manuel Lozano⋆ y Francisco Herrera⋆

Resumen— **En este trabajo describimos una implementación distribuida eficiente de un algoritmo genético para optimización en problemas reales. El algoritmo original fue propuesto en [9], donde se ofrecían resultados sobre una simulación realizada en un ordenador secuencial. Hemos realizado su implementación sobre un cluster de ordenadores equipados con Linux y empleando la biblioteca MPI. Comprobamos la eficacia real del algoritmo sobre un cojunto de problemas propuestos en una competición internacional (CEC'2005) y evaluamos la calidad de las soluciones y las ganancias de nuestra implementación. Los resultados experimentales muestran que el método es competitivo y que el rendimiento en la versión paralela aprovecha eficazmente los recursos hardware permitiendo su aplicación en problemas de mayor tamaño que en el caso de una implementación secuencial. Además ofrecemos resultados preliminares sobre la adaptación del método a problemas con dimensionalidad elevada.**

Palabras clave— **Algoritmo genético, computación distribuida, optimización contínua**

I. Introducción

Los algoritmos genéticos con codificación real (RCGAs)[1] han demostrado ser, tanto desde un punto de vista teórico como práctico, una herramienta potente para proporcionar una búsqueda robusta en espacios complejos y se han aplicado con éxito en multitud de problemas reales de búsqueda y optimización. Gran parte de este éxito se debe a su *capacidad de adaptación*, especialmente en grandes espacios de búsqueda donde la información disponible es escasa y en los que las técnicas clásicas de búsqueda no ofrecen buenos resultados.

El comportamiento de este tipo de algoritmos está fuertemente influenciado por el equilibrio entre explotación de buenas soluciones y exploración del espacio de búsqueda en busca de regiones más prometedoras [2]. Cuando este equilibrio no está adecuadamente proporcionado, la búsqueda suele estancarse y no ofrecer mejores soluciones que las ya encontradas. Este problema se conoce como *convergencia prematura*. Se han propuesto multitud de métodos para evitar la convergencia prematura y uno de los más imporantes son los *algoritmos genéticos con codificación real distribuidos* (DRCGAs) [3].

Los DRCGAs se basan en dividir la población en varias subpoblaciones las cuales evolucionan independientemente unas de otras [4][5]. Este nuevo enfoque basado en *separación espacial* [5][6] permite mantener una alta diversidad en las subpoblaciones junto con una adecuada explotación de la información, lo que favorece la conservación de alelos críticos para la búsqueda. Además, existe un mecanismo de migraciones que permite la cooperación entre subpoblaciones, lo que permite una mayor eficacia en la búsqueda a nivel global.

Una de las grandes ventajas de este tipo de algoritmos distribuidos es que son fácilmente implementables en hardware paralelo [7]. El hecho de que hoy en día el hardware paralelo (*clusters*, procesadores multicore, ...) sea muy asequible y tenga una gran difusión hace de los DRCGAs una herramienta ideal para abordar problemas de búsqueda complejos. Además, estos algoritmos no sólamente obtienen mejores tiempos de ejecución debidos al paralelismo intrínseco de la búsqueda con subpoblaciones, sino que pueden presentar soluciones de mejor calidad que sus versiones no distribuidas.

Uno de los aspectos más relevantes de los DRCGAs es la necesidad de una cooperación efectiva entre subpoblaciones [8]. Para ello, se proponen multitud de topologías para modelar el comportamiento distribuido de los algoritmos. Una buena topología junto con una política de migraciones equilibrada son los puntos clave para obtener una cooperación efectiva y que la búsqueda se desarrolle de forma adecuada.

II. Modelo distribuido en hipercubo

Un modelo especialmente interesante de DRCGA, llamado *algoritmo genético gradualmente distribuido con codificación real*(GDRCGA), se propone en [9]. Este modelo plantea un GA distribuido que usa ocho subpoblaciones ubicadas en ocho nodos conectados entre sí según una topología en hipercubo. En el hipercubo se distinguen dos caras claramente diferenciadas por su cometido:

- **Cara delantera**: estas subpoblaciones se dedican a la **exploración** del espacio de búsqueda para alcanzar regiones más prometedoras.
- **Cara trasera**: las subpoblaciones de esta cara se dedican a la **explotación** de la información disponible encontrada por el proceso de búsqueda.

La diferencia esencial entre las subpoblaciones radica en el operador de cruce para codificación real empleado y en la presión selectiva asociada al operador de selección.

†E-mail: ignaciorobles@gmail.com
⋆E-mail: {J.M.Benitez; lozano; herrera}@decsai.ugr.es

Como puede observarse en la Figura 1, las subpoblaciones de cada cara tienen sus operadores genéticos ajustados para favorecer la diversidad (en el caso de la cara delantera) ó el refinamiento (en el caso de la cara trasera). Además, estos operadores genéticos están ajustados de forma gradual de manera que actúan con diferentes intensidades.

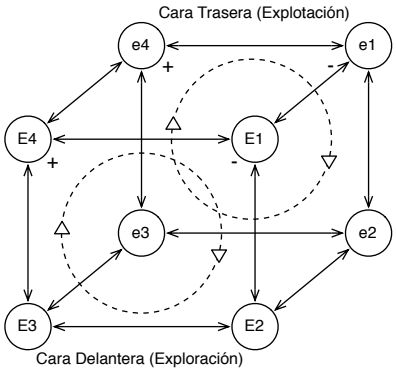

Fig. 1. Topología en hipercubo

También se propone una política de migraciones consistente en tres movimientos de **emigración** [13] diferentes, donde el mejor individuo de cada subpoblación abandona ésta para dirigirse a una única subpoblación destino. Cada cierto número de generaciones, se lleva a cabo la siguiente secuencia de movimientos de emigración:

1. **Migraciones de refinamiento**: los individuos emigran a favor de la gradualidad en la cara trasera y en contra en la cara delantera.
2. **Migraciones exp/ref**: los individuos emigran a la subpoblación de la misma posición pero en la cara opuesta.
3. **Migraciones de expansión**: los individuos emigran a favor de la gradualidad en la cara delantera y en contra en la cara trasera.

Una vez que se han aplicado los tres tipos de movimientos, la secuencia comienza de nuevo. De forma gráfica, los tres movimientos de emigración pueden observarse en la Figura 2.

Se considera que la búsqueda se ha estancado a nivel global si durante las últimas 50 generaciones no se ha actualizado el mejor individuo en ninguna subpoblación y si la mejora en el último reemplazo de los individuos mejor adaptados de las subpoblaciones ha sido muy baja. Si se dan ambas condiciones se aplica un *operador de reinicio* consistente en volver a crear las subpoblaciones de forma aleatoria.

Los operadores genéticos propuestos para este modelo son:

- **Operador de selección**: *Linear Ranking Selection* con muestreo estocástico universal [10].
- **Operador de mutación**: mutación no uniforme [11].
- **Operador de cruce**: BLX$-\alpha$ [12].

Dado que el esquema evolutivo propuesto es generacional, en cada generación se crean nuevos descendientes que reemplazarán directamente a sus padres.

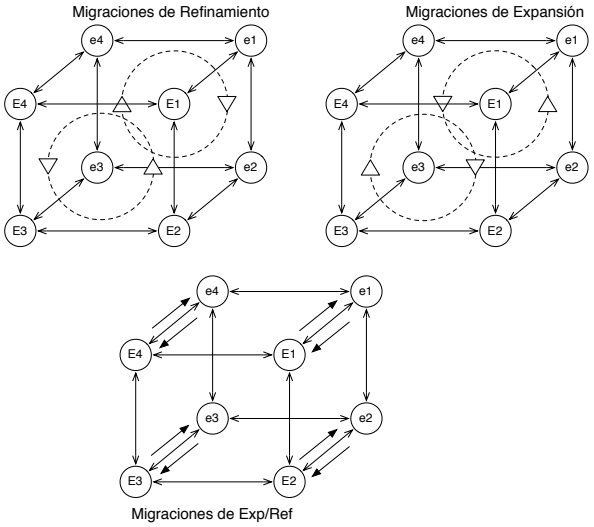

Fig. 2. Movimientos de emigración en el hipercubo

La gradualidad del operador de cruce BLX$-\alpha$ se puede regular mediante el parámetro $\alpha \in [0,1]$ que indica el grado de diversidad que produce este operador genético. A mayores valores de α, mayor exploración y a menores valores mayor explotación. La gradualidad del mecanismo de selección empleado se puede ajustar mediante el parámetro $\eta_{min} \in [0,1]$, el cual especifica el número de copias esperado para el peor individuo. Si η_{min} toma valores bajos, se alcanza una *presión selectiva* alta, en cambio si toma valores altos la presión es baja.

Tal y como se comentó anteriormente, los operadores genéticos de cruce y selección se aplican de forma gradual dependiendo del propósito de la subpoblación. Así, en la Tabla I se proponen los valores de los parámetros para la aplicación de estos dos operadores:

TABLA I
PARÁMETROS DE LOS OPERADORES DE CRUCE Y SELECCIÓN

	e_1	e_2	e_3	e_4	E_1	E_2	E_3	E_4
α	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8
η_{min}	0,5	0,6	0,7	0,8	0,3	0,2	0,1	0,0

III. Plataforma hardware y software

Un **cluster** es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador. Usualmente se emplean clusters para mejorar el rendimiento y/o la disponibilidad de la que es provista un solo ordenador.

La construcción de un cluster es económica y flexible: pueden tener todos la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y siste-

TABLA II
ERROR MEDIO RESPECTO AL ÓPTIMO GLOBAL DE LAS FUNCIONES DEL CEC'2005

	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
$D = 10$	5,92E+00	1,27E+03	2,02E+01	0,00E+00	1,49E+01	5,31E+00	5,62E+00
$D = 30$	4,04E+01	4,86E+03	2,05E+01	9,95E-02	1,11E+02	3,26E+01	4,23E+04
$D = 50$	6,03E+01	6,54E+03	2,04E+01	2,98E+00	1,75E+02	5,61E+01	2,37E+05
	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}
$D = 10$	4,41E-01	3,68E+00	5,16E+02	1,55E+02	1,47E+02	4,70E+02	4,66E+02
$D = 30$	1,40E+00	1,34E+01	5,01E+02	5,00E+02	5,41E+02	9,10E+02	9,08E+02
$D = 50$	2,80E+00	2,30E+01	3,00E+02	2,20E+02	2,18E+02	1,00E+03	9,92E+02
	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}	
$D = 10$	4,66E+02	9,08E+02	9,22E+02	1,03E+03	4,64E+02	1,41E+03	
$D = 30$	9,08E+02	1,10E+03	9,80E+02	1,10E+03	9,72E+02	1,38E+03	
$D = 50$	9,92E+02	1,01E+03	9,16E+02	1,02E+03	1,69E+03	1,38E+03	

mas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).

Para que un cluster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de gestión y explotación del cluster, el cual se encargue de interactuar con el usuario y los procesos que se ejecutan en él para optimizar el funcionamiento.

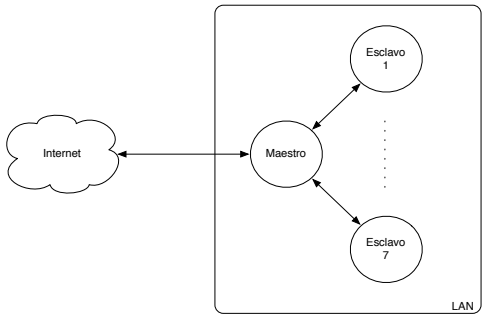

Fig. 3. Estructura del cluster utilizado

La figura 3 muestra gráficamente la estructura del cluster empleado en nuestro estudio. En particular, sus características son:

- 1 nodo maestro con conexión a internet
- 7 nodos esclavos
- Cada nodo, con un microprocesador Intel Xeon, 3,20 GHz y 1GB de RAM.
- Red de comunicación: Fast Ethernet
- Sistema operativo: Linux Fedora Core 8 (kernel 2.6.25.6-27)

El cluster cuenta con todo el software necesario para facilitar la comunicación de procesos distribuidos y paralelos. Concretamente se ha optado por implementar el modelo en Java utilizando MPJ [14], un envoltorio java sobre MPI, como biblioteca para gestionar procesos distribuidos.

IV. Evaluación experimental

De acuerdo al marco experimental establecido para la sesión especial en *Metaheurísticas, Algoritmos Evolutivos y Bioinspirados para Problemas de Optimización Contínua* del congreso, se han considerado las funciones propuestas en la sesión especial sobre optimización contínua del *IEEE Congress on Evolutionary Computation 2005* (CEC'2005) [15]. En esta sesión especial se proponían 25 funciones de diferente complejidad para encontrar el óptimo global. Se han considerado solamente las 20 últimas funciones, ya que las 5 primeras son especialmente fáciles de optimizar al ser unimodales. A su vez, se proponen tres dimensiones diferentes para cada función: 10, 30 y 50 variables. Evidentemente, el espacio de búsqueda crece conforme crece la dimensionalidad y por lo tanto aumenta la dificultad de encontrar el óptimo global buscado.

El criterio de parada está definido en función de la dimensión, D, del problema: el algoritmo finalizará su proceso de búsqueda cuando el número de evaluaciones realizadas sea mayor o igual a $D \cdot 10^4$ ó cuando el error respecto al óptimo global es inferior a 10^{-8}.

El algoritmo se ejecutará 25 veces con semillas diferentes para el generador de números pseudoaleatorios. Concretamente, cada subpoblación se inicializará con una semilla aleatoria diferente dentro de la misma ejecución de cara a favorecer la diversidad en la búsqueda.

En la Tabla II se presentan la media de los errores respecto al óptimo global para cada dimensión, D, propuesta: 10, 30 y 50.

Uno de los objetivos al utilizar GDRCGAs es, aparte de obtener soluciones de mayor calidad, obtener una ganancia en tiempo en las ejecuciones. En el trabajo original [9] se experimentó utilizando sólamente un único procesador, puesto que ahora el trabajo que hacía un solo procesador se divide entre ocho procesadores de forma distribuida, nos interesa saber en qué grado se está mejorando el tiempo de

TABLA III
SPEED-UP DE LAS FUNCIONES DEL CEC'2005

	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
$D = 10$	6,37E+00	4,33E+00	2,52E+00	1,98E+00	2,40E+00	7,88E+00	5,70E+00
$D = 30$	3,12E+00	5,19E+00	3,37E+00	2,38E+00	3,25E+00	7,79E+00	7,51E+01
	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}
$D = 10$	2,21E+00	2,55E+00	8,14E+00	7,84E+00	7,99E+00	7,91E+00	7,90E+00
$D = 30$	2,76E+00	3,38E+00	7,94E+00	7,91E+00	7,91E+00	7,81E+00	7,92E+00
	F_{20}	F_{21}	F_{22}	F_{23}	F_{24}	F_{25}	
$D = 10$	7,91E+00	7,87E+00	7,95E+00	7,84E+00	8,20E+00	5,85E+00	
$D = 30$	7,94E+00	7,91E+00	7,98E+00	7,98E+00	7,85E+00	7,05E+00	

ejecución.

La versión distribuida de nuestro algoritmo se ejecuta en ocho procesadores distintos, uno por cada nodo del hipercubo. En cada nodo *reside* una subpoblación que evoluciona paralelamente a las subpoblaciones de los otros procesadores. En este caso, cabría plantearse que el algoritmo distribuido fuese ocho veces más rápido que su versión secuencial. Este es el límite teórico ideal, dificilmente alcanzable porque habitualmente, no todo el algoritmo es paralelizable. Además, la versión distribuida tiene un sobrecoste en tiempo que no tenía la versión secuencial y es el tiempo invertido en las comunicaciones y sincronizaciones entre los procesos en ejecución paralela. En el caso del algoritmo que nos ocupa, esta comunicación se debe fundamentalmente a las migraciones del mejor individuo a otras poblaciones residentes en procesadores distintos, la cual se lleva a cabo mediante el paso de mensajes por la red del cluster. Esto implica la aparición de código no paralelizable que será el de envío/recepción de mensajes aparte de la necesidad de establecer métodos de sincronización que nos permitan ejecutar el algoritmo distribuido correctamente.

Una medida clásica en la computación distribuida y paralela es la *ganancia en tiempo (speed-up)*. El speed-up se refiere a cuánto más rápido es un algoritmo distribuido con respecto a su versión secuencial. El speed-up se define como:

$$S = \frac{T_1}{T_p} \qquad (1)$$

donde T_1 es el tiempo de ejecución de la versión secuencial equivalente del algoritmo donde se utiliza una sola población en un único procesador y T_p el tiempo de ejecución de nuestro algoritmo distribuido siendo p el número de procesadores. En nuestro caso $p = 8$, ya que tenemos 8 subpoblaciones, cada una asignada a un procesador diferente.

El speed-up lineal o speed-up ideal se obtiene cuando $S = p$. Cuando se ejecuta un algoritmo con speed-up lineal, duplicar el número de procesadores duplica la velocidad. Como es ideal, se toma como punto de referencia para estimar que los valores cercanos tienen una escalabilidad muy buena.

En algunas ocasiones se puede obtener un speed-up superior a N cuando se utilizan N procesadores. Este fenómeno es conocido como *speed-up superlineal*. El speed-up superlineal hace que parezca que la experimentación no ha ido bien ya que podría pensarse que existe un límite superior para el speed-up el cual debería ser N cuando se utilizan N procesadores. Una posible causa del speed-up superlineal es el efecto caché como resultado de las diferentes jerarquías de memoria en los ordenadores modernos.

En la Tabla III se presentan las medidas speed-up para cada dimensión, D, estudiada.

Como se observa en la Tabla III, el speed-up alcanzado en todas las funciones es excelente. Como se ha comentado anteriormente, no vamos a alcanzar el límite teórico de speed-up lineal, pero estamos cerca de él, lo que sugiere que nuestro algoritmo tiene una buena escalabilidad. El speed-up es prácticamente idéntico en las diferentes funciones estudiadas, en todas las ejecuciones, de lo que se desprende que el buen speed-up no es fruto del azar (por ejemplo debido a una convergencia prematura debido a la semilla de números aleatoria utilizada) sino de un buen diseño e implementación.

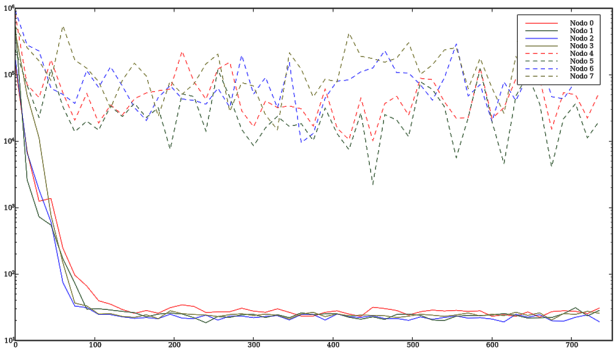

Fig. 4. Evolución de las subpoblaciones

Resulta muy interesante observar una gráfica de convergencia de una ejecución de nuestro algoritmo para comprender mejor su funcionamiento. La Figura 4 muestra la evolución del mejor individuo de cada subpoblación del hipercubo. El eje horizontal representa la generación actual y el vertical el *fit-*

TABLA IV
ERROR MEDIO RESPECTO AL ÓPTIMO GLOBAL DE LAS FUNCIONES DEL CEC'2008

	F_1	F_2	F_3	F_4	F_5	F_6
$D = 100$	5,11E-13	1,21E+01	2,39E+01	4,47E+01	2,84E-13	1,40E-07
$D = 500$	6,20E-07	2,76E+01	1,02E+03	2,05E+02	3,65E-07	1,13E-02
$D = 1000$	2,15E-04	3,53E+01	2,65E+03	3,69E+02	7,84E-06	4,11E+00

ness del mejor individuo de cada subpoblación. En la gráfica se pueden observar dos tendencias:
1. Las cuatro **líneas discontínuas** se corresponden con los **nodos de exploración** del hipercubo. Como se observa, presentan perturbaciones muy frecuentemente. Esto es debido a que en esos nodos se está generando diversidad, y lo que interesa es precisamente eso: generar soluciones candidatas muy diferentes entre sí con el propósito de intentar obtener soluciones mejores.
2. Las cuatro **líneas con trazo continuo** se corresponden con los **nodos de explotación** del hipercubo. En este caso, presentan mucha menos perturbación que en los nodos de exploración. Esto se debe a que estos nodos son los responsables de optimizar las soluciones, y generalmente tras la optimización el valor de adaptación (*fitness*) se queda en niveles próximos.

Evidentemente, la solución que devuelve el algoritmo se corresponde con la mejor solución de entre todas las mejores soluciones encontradas en los nodos.

A modo ilustrativo, algunos tiempos reales de ejecución del algoritmo se muestran en la Tabla V.

TABLA V
TIEMPOS REALES DE EJECUCIÓN

	Dimensión	Tiempo (s)
F_6	10	4,60E-01
F_7	30	5,17E+00
F_8	30	3,39E+00
F_9	30	2,95E+00
F_{10}	50	7,73E+00
F_{11}	50	9,37E+00

V. Escalabilidad del algoritmo con la dimensión

Considerando que una de las principales ventajas de los algoritmos paralelos es su capacidad para resolver problemas de mayor tamaño que las vesiones secuenciales, existe un experimento sumamente ilustrativo. Se trata de comprobar cómo se comporta nuestro DRCGA frente a funciones de dimensión creciente y dimensiones mucho más grandes de las que habitualmente se consideran en problemas de optimización. En particular, hemos considerado las funciones de la competición CEC'2008 [16]. Al igual que su predecesora del 2005, el objetivo en la competición del CEC'2008 es tratar de encontrar el óptimo global en una serie de funciones.

Como avanzábamos, la competición CEC'2008 tiene una especial dificultad: se trata de **optimización de gran escala** donde las funciones presentan hasta **mil dimensiones** cada una. Esta peculiaridad hace que el **espacio de búsqueda sea especialmente grande**. Por lo tanto, esta competición es mucho más difícil de abordar que la del 2005 y presenta un reto para nuestro algoritmo. Algunas de las funciones propuestas presentan la misma dificultad que anteriormente comentábamos: la presencia de muchos óptimos locales. Las características de las funciones utilizadas son:

- F_1: **Shifted Sphere Function**, unimodal y separable.
- F_2: **Shifted Schwefel's Problem 2.21**, unimodal y no separable.
- F_3: **Shifted Rosenbrock's Function**, multimodal y no separable.
- F_4: **Shifted Rastrigin's Function**, multimodal y separable.
- F_5: **Shifted Griewank's Function**, multimodal y no separable.
- F_6: **Shifted Ackley's Function**, multimodal y separable.

Como en el CEC'2008 las funciones presentadas son escalables entre dimensiones 2 y 1000, se han seleccionado tres dimensiones a abordar: 100, 500 y 1000. La Tabla IV muestra el error (diferencia entre la adaptación de la solución encontrada y la adaptación del óptimo global) de las soluciones obtenidas para las funciones del CEC'2008. Se puede comprobar que el algoritmo mantiene un nivel de calidad aceptable frente a incrementos en el tamaño del problema. A la luz de estos resultados podemos indicar que son muy prometedores. No obstante, es preciso realizar un estudio experimental más detallado para establecer resultados con suficiente fundamento. Ese trabajo está actualmente en curso.

VI. Concluciones y comentarios finales

En este trabajo se ha realizado una implementación eficaz de un Algoritmo Gradualmente Distribuido de Codificación Real. La calidad de la implementación se ve avalada por los buenos resultados de ganancia en tiempo respecto a su versión secuencial. La implementación permite acercarse a ganancias casi lineales.

También hemos comprobado la validez del método en la resolución de problemas complejos de optimización en espacios reales. Los resultados que ofrece son de buen nivel en relación con los algoritmos participantes en el CEC'2005 [15], si bien se hace necesario un refinamiento para alcanzar al mejor. Estos resultados avalan la ventaja de conseguir una adecuada separación espacial: el uso de ocho subpoblaciones diferentes en lugar de una única gran población beneficia al proceso de búsqueda.

Finalmente, hemos ilustrado la adaptabilidad del método en problemas de elevada dimensionalidad. Aunque se precisa una experimentación más extensiva para establecer conclusiones definitivas (experimentación en curso) los resultados preliminares permiten comprobar que el algoritmo se comporta bien frente a problemas de dimensión que crece hasta el millar de variables.

Referencias

[1] F. Herrera, M. Lozano, and J.L. Verdegay. Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. Artificial Intelligence Review, 12:265–319, 1998.

[2] S. Wright. (1934). Proceedings of the Sixth International Congress on Genetics, pages 356-366.

[3] V. S. Gordon and D. Whitley. (1992). Serial and parallel genetic algorithms as function optimizers. Proc. 5th Int. Conf. Genetic Algorithms, pages 177-183.

[4] M. Dorigo and V. Maniezzo. (1993). Parallel Genetic Algorithms: Theory and Applications, pages 5-42.

[5] P. Adamidis. (1994). Review of Parallel Genetic Algorithms Bibliography. Aristotle University of The ssaloniki.

[6] S. C. Lin, W. F. Punch III and E.D. Goodman. (1993). Proceedings of the Sixth IEEE Parallel and Distributed Processing. IEEE Press, pages 28-37.

[7] E. Alba, F. Luna, A. Nebro, J.M Troya. (2004). Parallel Heterogeneous Genetic Algorithms for Continuous Optimization, Parallel Computing, 30(5-6):699-719.

[8] E. Alba and M. Tomassini. (2002). Parallelism and Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation, IEEE Press, 6(5):443-462.

[9] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. IEEE Transactions on Evolutionary Computation, 4(1):43–63, April 2000.

[10] –. (1987). Reducing bias and inefficiency in the selection algorithm. Proc. 2nd Int. Conf. Genetic Algorithms Appl, pages 14-21.

[11] A. Neubauer. (1997). A theoretical analysis of the non-uniform mutation operator for themodified genetic algorithm. IEEE International Conference on Evolutionary Computation, pages 93-96.

[12] H. J. Bremermann, M. Rogson and S. Salaff. (1966) Global properties of evolution processes. Natural Automata and Useful Simulations, pages 3-41.

[13] B. Kröger, P. Schwenderling, and O. Vornberger. Parallel Genetic Algorithms: Theory and Applications, pages 151–186, 1993.

[14] M.A. Baker, D.B Carpenter, et.al., mpiJava: An Object-Oriented Java interface to MPI, the 1st Java Workshop at the 13 th IPPS & 10th SPDP Conference, Puerto Rico, April 1999, LNCS, Springer Verlag, Heidelberg, Germany, ISBN 3-540-65831-9.

[15] Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.P., Auger, A., Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC'2005 Special Session on Real Parameter Optimization. Technical Report. Nanyang Technological University.

[16] K. Tang, X. Yao, P. Suganthan et al. (2007). Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical Report. Nature Inspired Computation and Applications Laboratory, USTC, China.