ALGORITMOS EVOLUTIVOS PARALELOS: IMPLEMENTACIONES SOBRE UN MODELO UNIFICADO

Alfonso H., Fernández N., Minetti G., Salto C.
Proyecto UNLPAM-09/F015[1]
LISI (Laboratorio de Investigación de Sistemas Inteligentes)
Departamento de Informática - Facultad de Ingeniería
Universidad Nacional de La Pampa
Calle 110 esq. 9
(6360) General Pico – La Pampa – Rep. Argentina
e-mail: {alfonsoh, fernaty, minettig, saltoc }@ing.unlpam.edu.ar
Tel.: (02302)422780/422372, Int. 6302

Resumen

En la actualidad los *algoritmos evolutivos* (AE) se usan para buscar soluciones a problemas complejos para los cuales otras técnicas pueden insumir mucho tiempo y que, por lo general, proveen una única solución óptima.
Una tendencia actual consiste en disponer de la mayor cantidad de recursos computacionales para alcanzar los resultados de forma más rápida por medio de un trabajo cooperativo. La inclusión del paralelismo, distribución de tareas en varios procesadores, en el diseño de los algoritmos evolutivos ha sido muy importante dando lugar a mecanismos de búsqueda y optimización mejorados: *algoritmos evolutivos paralelos*.
Este trabajo presenta una breve revisión de los algoritmos evolutivos paralelos. Además, realiza un análisis comparativo del comportamiento de estos algoritmos con su versión secuencial, a fin de identificar cuáles son sus aciertos y debilidades. El paquete de software utilizado responde a un modelo unificado desarrollado en la Universidad de Málaga. La evaluación de los algoritmos se realiza analizando los resultados obtenidos para dos problemas de optimización bien conocidos como lo son: *OneMax* y *Mochila Binaria*.

Palabras claves: AE paralelos, AE distribuidos, AE celulares.

[1] Grupo de investigación avalado por la Universidad Nacional de La Pampa.

1. Introducción

La Computación Evolutiva (CE) toma como base la *teoría de la evolución* de Charles Darwin, según la cual las especies van evolucionando para adaptarse al medio que les rodea, de forma que aquellos individuos que mejor se adecuen tengan mayor probabilidad de sobrevivir hasta la edad adulta y procrear, consiguiendo que sus genes pasen a la siguiente generación.
Dentro de la CE se pueden distinguir distintos enfoques de simulación de la evolución: *Algoritmos Genéticos* (AG) [16], *Estrategias Evolutivas* (EE) [19, 22] y la *Programación Evolutiva* (PE) [12]. Las implementaciones actuales, descendientes de estos enfoques, se conocen como *algoritmos evolutivos* y tienen en común las siguientes operaciones:

- *Reproducción* (crossover): se realiza a través de la transferencia del programa genético de un individuo a su progenie.
- *Modificaciones aleatorias* (mutación): en un sistema positivamente entrópico, se garantiza que necesariamente ocurren errores de replicación durante la transferencia de información.
- C*ompetencia*: es una consecuencia de la expansión de poblaciones en un espacio de recursos finitos.
- *Selección*: es el resultado de la replicación competitiva a medida que la especie colma el espacio disponible.

Estos cuatro procesos son la esencia de la evolución, puesto que ésta no es más que el resultado inevitable de la interacción que se produce entre los mismos.
La forma usual de un AG fue descrito por Goldberg [14], en el cual el algoritmo comienza con un conjunto de soluciones o individuos escogidos al azar, llamado *población*. Cada individuo de la población, conocido como *cromosoma*, evoluciona a través de las distintas iteraciones, denominadas *generaciones*. Denotamos por P(t) una población de μ individuos en la generación t. Durante cada generación, el cromosoma se *evalua* usando alguna medida de *fitness*. Para la creación de una nueva población, P(t), los nuevos cromosomas o *hijos* se forman mediante la mezcla de dos cromosomas de P(t-1) utilizando el operador *crossover* y modificándolos a través del operador de *mutación*. El criterio de parada, que indica la finalización de la evolución, consiste de alguna condición como alcanzar un número dado de evaluaciones de la función, completar un número de generaciones, hallar el valor óptimo (si es conocido), o detectar un estancamiento en el algoritmo luego de un número dado de generaciones. El Algoritmo 1 presenta un bosquejo de un algoritmo evolutivo secuencial basado en un AG.

Algoritmo Evolutivo Secuencial

t = 0
inicializar P(t)
evaluar P(t)
mientras no (criterio de parada) **hacer**
    t = t+1
    Seleccionar individuos de P(t-1) y colocarlos en P(t)
    Modificar P(t) //por medio de crossover y mutación
    Evaluar P(t)
fin mientras

Algoritmo 1: algoritmo evolutivo secuencial

El hecho de que el AE trabaje con poblaciones de gran tamaño y/o cromosomas de considerable longitud supone una significativa utilización de recursos computacionales, como memoria física y tiempo de procesador. El tiempo de ejecución se puede disminuir al reducir el número de evaluaciones para alcanzar una solución o al ejecutar el algoritmo en una máquina paralela. Los modelos paralelos de estos algoritmos, llamados *algoritmos evolutivos paralelos* (AEP), son

interesantes porque consiguen ambos objetivos, ya que modifican el comportamiento típico del algoritmo secuencial equivalente mediante el uso de una población estructurada – una distribución espacial de individuos ya sea en la forma de un conjunto de islas [23] o de una grilla de difusión [17,21].

Las características interesantes que incluyen los AEP son: (a) disminución del tiempo para ubicar una solución (algoritmos más rápidos), (b) reducción en el número de evaluaciones (costo de búsqueda), (c) posibilidad de tener poblaciones de mayor tamaño y (d) aumento de la calidad en las soluciones halladas.

Las versiones de AEP son menos propensas a la convergencia prematura con lo cual se mejora el proceso de búsqueda [2]. Los AEP son interesantes porque no son simplemente "versiones más rápidas" de AE secuenciales sino que además proporcionan un mecanismo de búsqueda distinto, frecuentemente mejor.

Un AEP extiende la versión secuencial incluyendo una fase de *comunicación* con un vecindario formado por otros algoritmos evolutivos, como se muestra en el Algoritmo 2. Su comportamiento global queda determinado por la forma de realizar esa comunicación, el tipo de operaciones del resto de algoritmos en paralelo y otros detalles de funcionamiento. En el Algoritmo 2 se nota la generación de cada subalgoritmo con superíndice t^i para constatar el hecho de que distintos subalgoritmos pueden estar en distintas etapas de su evolución.

Algoritmo Evolutivo Paralelo
AEP^i

$t^i = 0$
Inicializar $P^i(t^i)$
Evaluar $P^i(t^i)$
mientras no (criterio de parada) **hacer**
    $t^i = t^i + 1$
    Seleccionar individuos de $P^i(t^i\text{-}1)$ y colocarlos en $P^i(t^i)$
    Modificar $P^i(t^i)$ //por medio de crossover y mutación
    Evaluar $P^i(t^i)$
    Comunicar $P^i(t^i) \cup \text{AEP}^j :: P^j(t^j)$; // interacción con los vecinos
    Seleccionar entorno $P^i(t^i) \cup P^i(t^i\text{-}1)$
fin mientras

Algoritmo 2: algoritmo evolutivo paralelo

La ejecución paralela se puede lograr a nivel lógico o físico (ambientes SIMD o MIMD). En el último caso, la ganancia consiste en una mayor velocidad de ejecución sin alterar el algoritmo básico; esto implica que cualquier aplicación puede usar un modelo distribuido y/o celular sin poseer una red de comunicaciones o un sistema multiprocesador. Es claro que al disponer de un sistema paralelo físico, las ventajas aumentan significativamente.

El presente trabajo trata los AEP, sus distintos tipos, centrando la atención en un enfoque de diseño y estudio unificado de modelos secuenciales y paralelos. Existen publicaciones con revisiones que pueden utilizarse para hacer un seguimiento de los AEP en el tiempo, como por ejemplo [4, 2, 9, 1, 11].

La organización de este trabajo se detalla a continuación. Primero, una clasificación de los AE según el tratamiento que se realiza sobre la población. Luego, la especificación de varios experimentos que utilizan un paquete que implementa un modelo unificado de AE paralelo. Finalmente, un delineamiento de las principales conclusiones elaboradas.

2. Clasificación

Los aspectos de un AE secuencial posibles de paralelizar son aquellos operadores que no usan panmixia, es decir que se pueden ejecutar a la vez sobre diferentes porciones de la población. Entre los operadores no-panmíticos se encuentran los unarios (mutación) y los binarios (crossover) que, junto con las operaciones de evaluación de adaptación de los descendientes, se pueden paralelizar sin provocar ningún cambio en el modelo secuencial básico. A partir de estas paralelizaciones y manteniendo una selección centralizada se genera un algoritmo más rápido en tiempo real pero que conserva las mismas características de búsqueda que si no estuviese trabajando en paralelo. A los modelos resultantes se los denomina de *paralelización global* [10].
Otra forma de lograr la paralelización es considerar un modelo de selección descentralizado donde los individuos se agrupan espacialmente dando lugar a AE *estructurados*. Entre los tipos más conocidos de AE estructurados se encuentran: los *distribuidos* (AEd) y los *celulares* (AEc) [2]. Ambos tipos resultan de particionar una única población en varias. En los primeros se generan islas de AE que se ejecutan realizando intercambios esporádicos de individuos, mientras que en los segundos, la descentralización se da en forma de vecindarios. En la Figura 1 se muestra: (a) un AE panmítico (todos los individuos pertenecen a una misma población y se pueden cruzar entre sí), (b) un AE distribuido y (c) un AE celular.

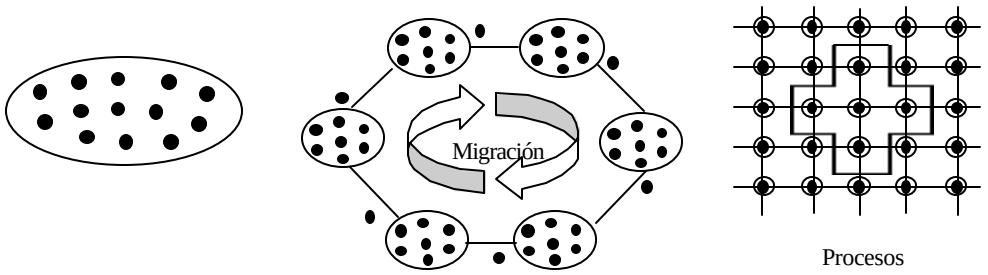


Figura 1: Algoritmos evolutivos panmíticos y estructurados.

En [4, 3] se presenta el cubo de poblaciones estructuradas, Figura 2, que establece las diferencias entre los modelos, donde cada uno admite múltiples implementaciones. El modelo celular consta de un elevado número de subalgoritmos con frecuente comunicación entre ellos, cada uno con unos pocos individuos (usualmente sólo uno). Por el contrario, un modelo distribuido está formado por pocos subalgoritmos con poblaciones de mayor tamaño y realiza intercambios esporádicos de información. Otras combinaciones dan lugar a posibles modelos, a veces de difícil clasificación.

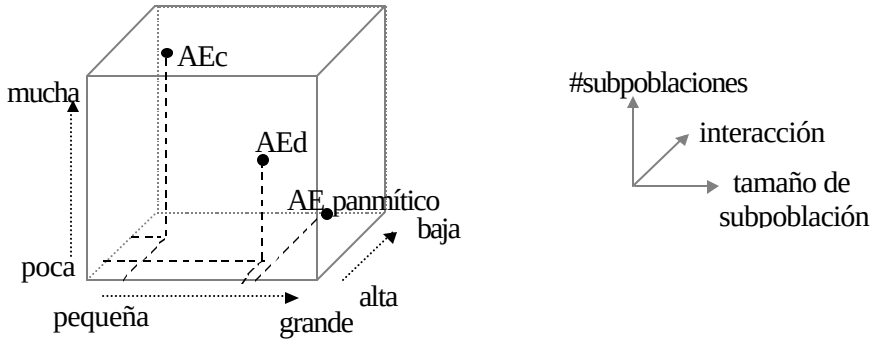


Figura 2: Cubo de poblaciones estructuradas

Los modelos celulares disponen los individuos de una manera bidimensional, como por ejemplo una malla toroidal, haciendo interactuar a cada individuo con sus vecinos en función del vecindario definido. En estos modelos, un individuo pertenece a varios vecindarios provocando el solapamiento de ellos. La existencia de pequeños vecindarios solapados ayuda en la exploración del espacio de búsqueda [8] y conduce a una migración continua y transparente en la grilla.
Hay varios tipos de vecindarios, básicamente se los divide en dos grandes clases: *lineales y compactos* [20] (Figura 3). En los vecindarios lineales, *Lr*, los vecinos de un punto dado incluyen a las *r*-1 estructuras más cercanas elegidas sobre los ejes horizontales y verticales (por ejemplo L5 y L9 de la Figura 3). En los vecindarios compactos, *Cr*, los vecinos de un punto dado incluyen a los *r*-1 individuos más cercanos. El tipo de elección puede dar lugar a vecindarios compactos cuadrados o diamantes (por ejemplo C9, C13 y C25 de la Figura 3). Uno de los vecindarios más usado es el L5, conocido con el nombre de *NEWS* (North-East-West-South), debido a su sencillez teórica y a la rapidez en la selección. Se pueden definir nuevos tipos de vecindarios según lo requiera el problema en particular.
Es claro que los vecindarios pequeños (como el L5) sobrecargarán al sistema menos que otros vecindarios más complejos (como el C9 o superior). En el caso de que se trabaje con vecindarios muy grandes y mallas chicas el efecto es similar al de usar una única población (panmixia).

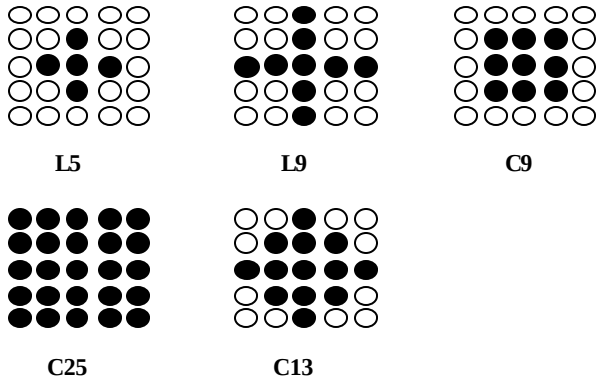


Figura 3: Tipos de vecindarios Lineal y Compacto.

Por otra parte, los modelos distribuidos pueden utilizar: (a) islas cuya evolución básica es secuencial y en panmixia, (b) islas de estado estacionario [24] ó (c) islas celulares. Debido a la amplia disponibilidad de algoritmos generacionales la opción (a) ha sido la más difundida. El tipo de AE distribuido en uso lo determina la política de migración, la cual define: la topología de las islas, los individuos a intercambiar, la sincronización entre las subpoblaciones, y la forma en que se integran los individuos en la subpoblación destino. Existe un consenso respecto a utilizar una topología anillo (o hipercubo) debido a su simplicidad, fácil implementación y ganancia en tiempo de comunicación. La ventaja de este modelo distribuido es que normalmente es más rápido que un panmítico. Además mantiene una alta diversidad y permite la formación de especies.
Los modelos distribuidos son débilmente acoplados facilitando una implementación distribuida, mientras que los celulares son fuertemente acoplados debido a las interacciones de cada individuo con su vecindario. La sincronización en los AE celulares se realiza al final de cada ciclo reproductor, mientras que en los distribuidos se hace con menor frecuencia. Comparando los AE estructurados con los secuenciales, en la mayoría de los casos, los primeros realizan un mejor muestreo del espacio de búsqueda y presentan un comportamiento más eficiente y rápido en la obtención de soluciones [5, 15].

En [4] se puede encontrar un overview de distintos modelos e implementaciones de AEP con diferentes niveles de flexibilidad, que abarcan desde un AEP simple hasta la especificación de un modelo general de AEP. Estos trabajos se enfocan parcialmente sobre modelos paralelos absolutamente propios no estandarizados y, además, resuelven problemas particulares. Si bien ofrecen resultados bastante importantes, cada uno en su campo de aplicación, son sólo particularizaciones que no representan ningún modelo en sí.
La necesidad de contar con un sistema de búsqueda flexible y eficiente que sea lo suficientemente genérico, como para derivar de él diferentes familias de AEP y que permita elegir el modelo más conveniente para un problema, ha llevado al desarrollo de varios modelos paralelos distribuidos que permiten ajustarse a los requerimientos del problema.

3. Implementaciones de AE secuenciales y estructurados

En esta sección se presenta un análisis comparativo del comportamiento de un AE secuencial versus AE estructurados, con el objetivo de identificar cuáles son los aciertos y debilidades de usar estas nuevas tendencias.
Para llevar a cabo este trabajo se utilizó un paquete de software, denominado *arg*, que implementa un AE de estado estacionario (AEs) y su correspondiente distribuido (AEd). Este paquete fue desarrollado por el grupo de investigación de Ingeniería de Software de la Universidad de Málaga [13]. El AEs contiene el código de un algoritmo genético secuencial que implementa la selección por torneo, el crossover clásico de un punto, la mutación tradicional y el nuevo individuo generado reemplaza al peor de la población. Utiliza una población inicial aleatoria formada por cromosomas binarios. El AEd adiciona las modificaciones necesarias para establecer la conexión de las subpoblaciones en el anillo.
El paquete *arg* está desarrollado con una metodología orientada a objetos, implementado en JAVA. Tiene especificado dos problemas de optimización (maximización) listos para ser usados, ellos son el *OneMax* y *MaxSat* [7]. Además, se le incorporaron en general dos modificaciones:

- Una para la resolución del problema de optimización conocido como *Mochila Binaria* (*Knapsack Problem*) [18].
- Otra para adicionar un módulo que permite ejecutar un AE celular (AEc) y una versión distribuida del mismo (AEcd).

En particular, el AEc trabaja con los mismos operadores que el AEs a nivel de vecindario. Se consideró el vecindario Lineal 5 (L5 o NEWS). La selección de los individuos se hace por torneo pero dentro de los individuos del vecindario considerado. El individuo generado se almacena en una población auxiliar sólo si es mejor al individuo original, caso contrario se mantiene el individuo original, y al finalizar el recorrido de toda la población se reemplaza la población actual por la población auxiliar para continuar con el ciclo evolutivo. Se utilizó un tamaño de malla de 64x8, determinado a partir de un estudio preliminar.
Se realizaron corridas de 30 ejecuciones en ambientes de una máquina, aún para las versiones distribuidas. Cada experimento se realizó con un máximo de 200.000 pasos y una población de 512 individuos, que para el caso de los distribuidos fue dividida en 4 islas de 128 individuos.
Para los AE distribuidos, la frecuencia de migración se calcula multiplicando el tamaño de la población de la isla por 32, lo que indica que la migración se produce con una frecuencia de 4096 pasos, migrando el mejor individuo encontrado a la próxima isla del anillo. El control de si se ha recibido inmigrantes de la isla previa se hace cada 50 pasos y el individuo recibido reemplazará al peor individuo de la población.

A continuación se describen con más detalle los experimentos y el análisis de los resultados para cada problema tratado: *OneMax* y *Mochila Binaria*. Las variables de performance utilizadas para comparar el comportamiento de los algoritmos son:

- **#Pasos Promedio:** número promedio del total de iteraciones del algoritmo evolutivo de estado estacionario.
- **#Pasos Promedio Mejor Fitness:** número promedio de iteraciones donde se encuentra el mejor valor.
- **Mejor Fitness Promedio:** Promedio de los mejores valores hallados de cada corrida
- **Fitness Promedio Poblacional:** Media de los fitness promedio de los individuos de cada población final
- **#Penalizaciones:** Cantidad promedio de mejores soluciones, detectadas por corrida, que han sido penalizadas.
- **Segundos**: tiempo promedio de cada corrida.
- **Iteraciones/segundo**: promedio de iteraciones por unidad de tiempo para cada corrida.

3.1. Problema *OneMax*

Para el problema OneMax se usó una longitud de cromosoma de 512, probabilidad de crossover 85% y probabilidad de mutación 0.19%, calculada en base a la siguiente fórmula: 1/(#genes*long_gen). El valor óptimo esperado es 512, por lo que se decidió detener el algoritmo cuando se encuentra dicho valor o cuando se alcanza un máximo de pasos.

En la Tabla 1, donde se muestran los promedios de los resultados alcanzados para cada una de las 30 corridas, se observa una mejor calidad de las soluciones para los AE celulares, pues el valor óptimo se alcanza en cada una de las corridas. Analizando el fitness promedio poblacional se desprende que también para estos algoritmos hay una alta concentración de los individuos de la población alrededor del valor óptimo. Estos resultados se pueden corroborar en los Gráficos 1 y 2.

Algoritmo	#Pasos Promedio	#Pasos Promedio Mejor Fitness	Mejor Fitness Promedio	Fitness Promedio Pobl.	Segundos	Iteraciones/ Segundo
AEs	173658,4667	143958,7000	511,3000	510,7758	50,60	6.351,67
AEd	135962,4333	135962,4333	511,9667	509,9667	136,43	48.365,07
AEc	3623,6333	3623,6333	512,0000	510,9876	587,27	6.315,20
AEdc	3378,2000	3378,2000	512,0000	510,2833	468,70	88.125,17

Tabla 1 . Promedio de resultados alcanzados para cada tipo de algoritmo.

Por otra parte, en la Tabla 1, se puede apreciar que el número de pasos para hallar el óptimo disminuye al utilizar una población estructurada, en particular, en forma notable al usar AE celulares. El costo computacional, expresado en segundos, se incrementa con los AE estructurados debido a que los experimentos se realizaron en un ambiente de máquina única. Este incremento es más notorio con los AE celulares pero esto no se refleja en la cantidad de iteraciones que se realizan por segundo, debido a que ellos generan la población completa en cada paso mientras que el AEs, al ser un algoritmo de estado estacionario, obtiene un único individuo en cada paso.

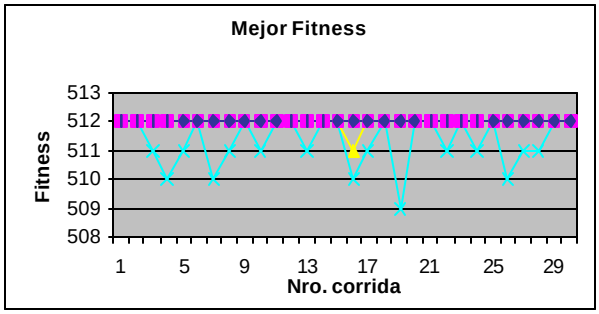


Gráfico 1. Mejor Fitness alcanzado en cada experimento.

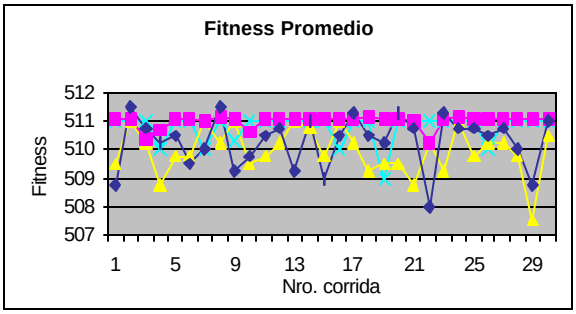
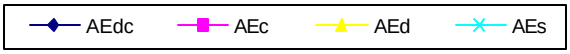


Gráfico 2. Fitness promedio de la población alcanzado en cada experimento.



3.2. Problema de la *Mochila Binaria*

El problema de la *Mochila Binaria* trabaja con capacidad promedio y penaliza los individuos que superan dicha capacidad. Aquí, se ha enfocado el problema usando *funciones de penalidad lineal* [18] ya que en experimentaciones previas es la alternativa que mejor performance ha mostrado [6]. Se usó un conjunto de 100 ítems, cuyos pesos y beneficios se definieron en forma no correlacionada.

En el paquete *arg* se modificó la definición del cromosoma para registrar si el individuo ha sido penalizado o no. Además se incorporó en las estadísticas poblacionales la contabilidad de cuántos individuos de la población final han sufrido penalización.

Se usó una longitud de cromosoma de 100, probabilidad de crossover 85% y probabilidad de mutación 1%. La capacidad promedio de la mochila es de 587 y para el cálculo de la penalidad se usa un δ=21. El valor óptimo se desconoce, por lo tanto el criterio de terminación usado es la estabilización del fitness medio por más de 2000 pasos consecutivos. Se considera estable cuando el cambio producido no supera al 0.001.

En la Tabla 2 se puede observar que el AEs ha alcanzado el mayor promedio del mejor fitness, pero para ello requiere un alto número de pasos. Inclusive se puede afirmar que el fitness promedio poblacional nunca se estacionó pues, a pesar de que el mejor fitness promedio se halla cuando media un cuarto de la cantidad total de pasos, el algoritmo continua hasta alcanzar la cota máxima. AEdc y AEc, en este orden, alcanzan valores óptimos muy cercanos a AEs, pero realizando muchos menos pasos.

Analizando el fitness promedio poblacional se desprende que los algoritmos AEs, AEc y AEdc son los que mantienen una mayor concentración de individuos en la zona próxima al mejor valor alcanzado, esto a su vez puede ser corroborado observando la distribución del fitness de los mejores individuos y de la media poblacional en cada una de las corridas en los Gráficos 3 y 4.

Es importante resaltar que los algoritmos celulares encuentran todas las soluciones válidas mientras que los AEs y AEd presentan individuos penalizados, aunque para los últimos la proporción de soluciones inválidas es mayor (columna #Penal. en Tabla 2).

Por otra parte, es claro que el costo computacional se incrementa para los AE estructurados debido a que los experimentos se realizaron en un ambiente de máquina única; siendo más notorio para el AEc aunque la cantidad de iteraciones que se realizan por segundo son similares a las del AEs.

Algoritmo	#Pasos Promedio	#Pasos Promedio Mejor Fitness	Mejor Fitness Promedio	Fitness Promedio Pobl.	#Penal.	Segundos	Iteraciones/ Segundo
AEs	200000,00	51407,30	1034,07	1033,7973	0,37	21,93	13316,57
AEd	200000,00	133714,40	1018,80	1008,1917	0,80	53,63	14795,77
AEc	9488,60	6868,10	1029,03	1028,4975	0,00	716,03	13612,73
AEdc	7333,30	2195,93	1029,87	1022,8250	0,00	276,73	314776,20

Tabla 2 . Promedio de resultados alcanzados con cada tipo de algoritmo.

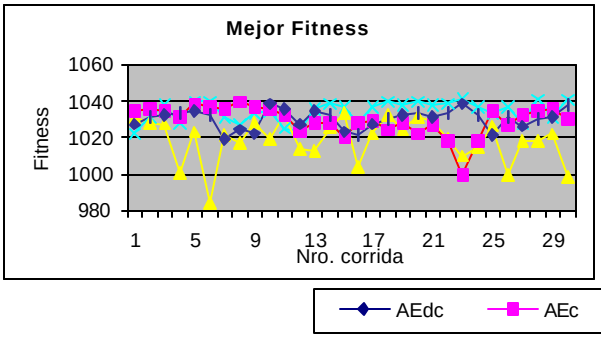


Gráfico 3. Mejor Fitness alcanzado en cada experimento.

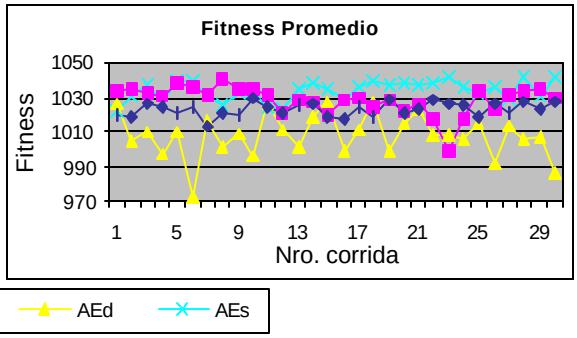


Gráfico 4. Fitness promedio de la población alcanzado en cada experimento.

4. Conclusiones

Este trabajo contiene una vista resumida de los principales aspectos de los algoritmos evolutivos paralelos. También se mostraron los resultados de varios experimentos sobre dos problemas de optimización (OneMax y Mochila Binaria) para contrastar el comportamiento de estos algoritmos con una versión secuencial.

A partir del análisis comparativo de los resultados alcanzados se concluye que al trabajar con algoritmos con poblaciones estructuradas espacialmente, en particular los celulares, se pueden alcanzar valores óptimos o de muy buena calidad en una menor cantidad de pasos.

La incorporación de la distribución en un algoritmo evolutivo secuencial de estado estacionario proporciona, en general, una mejora en cuanto a la calidad de las soluciones halladas. En tanto que para un algoritmo celular, la distribución reduce el número de pasos necesarios para encontrar la mejor solución.

Como trabajos futuros se realizarán otras experiencias tendientes a buscar alguna relación entre el tamaño de la malla y las distintas formas de vecindarios que se pueden implementar, como también determinar si se pueden obtener mayores ventajas trabajando sobre otros tipos y tamaños de mallas.

5. Agradecimientos

Agradecemos la cooperación del grupo de proyecto y de nuestro asesor, el Dr. Raúl Gallard, por proveer nuevas ideas y críticas constructivas. También a la Universidad Nacional de La Pampa de la que recibimos un aporte continuo.

6. Referencias

[1] P. Adamidis. *Review of Parallel Genetic Algorithms Bibliography*. Technical Report, Aristotle University of Thessaloniki, http://www.control.ee.auth.gr/~panos/papers/pga_review.ps.gz, Noviembre 1994.

[2] E. Alba y J. M. Troya. *A Survey of Parallel Distributed Genetic Algorithms*. En Complexity 4 (4), Pag. 31-52, 1999.

[3] E. Alba Torres. *Tesis Doctoral: Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos*, Universidad de Málaga, España, 1999.

[4] E. Alba Torres, M. Tomassini. *Parallelism and Evolutionary Algorithms*. En IEEE Transactions on Evolutionary Computation, vol 6, Pag. 443-462, Octubre 2002.

[5] E. Alba y J. M. Troya. *Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms.* En Statistical Computation, Vol. 12, Nº 2, Pag. 91–114, 2002.

[6] H. Alfonso, E. Bertone, G. Minetti, R. Gallard. *Comparación de Tres Métodos de Búsqueda Incompletos para el Problema de la Mochila*. En Anales WICC 2000 (Workshop de Investigadores en Ciencias de la Computación), Universidad Nacional de La Plata, Argentina, 2000.

[7] T. Bäck. *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, NY, 1996.

[8] S. Baluja. *Structure and Performance of Fine-Grain Parallelism in Genetic Search*. En S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms. Pag. 155–162, 1993.

[9] A. Chipperfield y P. Fleming. *Parallel Genetic Algorithms*. En A. Y. H. Zomaya, editor, Parallel and Distributed Computing Handbook, Pag. 1118-1143, MacGraw-Hill, 1996.

[10] E. Cantú-Paz. *Designing Efficient Master-Slave Parallel Genetic Algorithms*. IlliGAL report 97004, Mayo 1997.

[11] E. Cantú-Paz. *A Survey of Parallel Genetic Algorithms*. En Calculaterurs Parallèles, Réseaux et Systèmes Répartis, 10(2), Pag. 141-171, 1998.

[12] L. Fogel, A. Owens, M. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1996.

[13] GISUM: Grupo de Ingeniería de Software de la Universidad de Málaga, España. http://polaris.lcc.uma.es/~gisum/

[14] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[15] V. S. Gordon and D. Whitley. *Serial and Parallel Genetic Algorithms as Function Optimizers*. En S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms.Pag. 177–183, 1993.

[16] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[17] B. Manderick y P. Spiessens. *Fine-Grained Parallel Genetic Algorithms*. En J. D. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, Pag. 428-433. Morgan Kaufmann, 1989.

[18] Z. Michalewicz. *Genetic Algorithm + Data Structure = Evolution Programs*, 3ª ed., Springer-VerlEA, New York, 1996.

[19] I. Rechenberg. *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien derBiologischen Evolution*. Fromman-Holzboog Verlag, Sttutg, 1973.

[20] J. Sarma, K. De Jong. *An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms*. En H. M.Voigt, W. Ebeling, I. Rechengerg, H. P. Schwefel, editores,

Proceedings of the Fourth Parallel Problem Solving from Nature, Springer-Verlag , Pag. 236-244, 1996.

[21] P. Spiessens y B. Manderick. *A Massively Parallel Genetic Algorithm*. En L. B. Booker R. K. Belew, editor, Proceedings of the Fourth International Conference on Genetic Algorithms, Pag. 279-286, Morgan Kaufmann, 1991.

[22] H. Schwefel. *Numerical Optimization of Computer Model*s. Wiley, Chichester, 1981.

[23] R. Tanese. *Distributed Genetic Algorithms*. En J.D. Schaffer, editor, ICGA-3, Pag. 434-439.

[24] D. Whitley, T. Starkweather. *GENITOR II: a Distributed Genetic Algorithm*. Journal of Experimental and Theoretical Artificial Intelligence, Vol. 2, Pag. 189-214, 1990.