Algoritmos Basados en Cúmulos de Partículas para el Análisis de Microarrays de ADN

E. Alba, J. García-Nieto y G. Luque

Resumen— **En este trabajo se estudia la aplicación de los Algoritmos Basados en Cúmulos de Partículas (PSO) al problema de ordenación de genes en microarrays de ADN, un problema NP-duro con fuertes implicaciones en Biomedicina. Este problema consiste en la ordenación de un conjunto de genes, agrupando los que presenten comportamientos similares. El algoritmo PSO propuesto trabaja con representación de soluciones mediante permutaciones de enteros, y utiliza operadores adaptados a este tipo de representación. Además, se han desarrollado versiones secuenciales y paralelas del mismo integradas en la biblioteca MALLBA. La evaluación experimental sobre tres problemas reales demuestra la eficiencia y competitividad real de nuestra aproximación.**

Palabras clave— **Particle Swarm Optimization, microarrays de ADN, ordenación de genes, biblioteca MALLBA.**

I. Introducción

La investigación microbiológica a nivel molecular está viviendo un cambio espectacular en los últimos años. En apenas una década se ha pasado de trabajos basados en el estudio de uno o unos pocos genes al análisis de los genomas en su totalidad. Desde que en 1995 se publicara el primer genoma completo de un ser vivo, *Haemophilus influenzae* [1], ya se han descrito unos 145 genomas completos (incluyendo los de 19 organismos eucariotas como el hombre o el ratón) y aproximadamente 600 están siendo secuenciados en la actualidad. En este sentido, se están desarrollando nuevas tecnologías para trabajar con las grandes cantidades de datos que generalmente se obtienen en el proceso de análisis de un genoma. La técnica de *Microarrays* de ADN (MAs) [2] está atrayendo un gran interés ya que permite monitorizar la actividad de un genoma completo mediante un simple experimento. Además, cada uno de estos experimentos con MAs puede suponer el manejo desde cientos hasta decenas de miles de genes, normalmente con decenas de muestras por gen. Por este motivo, son necesarias técnicas de reducción que permitan agrupar genes con patrones de expresión relacionados ya que es probable que tales genes se regulen a sí mismos. Para este propósito se vienen utilizando técnicas de *Clustering* como K-means o métodos aglomerativos (véase por ejemplo [3] y [4]). Sin embargo, es posible mejorar más aún la calidad de las soluciones que se obtienen con estos métodos.

En este trabajo se propone la utilización de algoritmos basados en cúmulos de partículas, comúnmente llamados *Particle Swarm Optimization* (PSO) [5], para realizar una reordenación de los genes muestreados en un MA, de manera que los genes relacionados (o con una gran similitud desde el punto de vista de sus niveles de expresión) sean posicionados próximos en regiones dentro de la secuencia del MA. Esta reordenación, realizada de manera adecuada, puede proporcionar mejoras sustanciales (como se muestra en [6]) en procesos realizados a posteriori como pueden ser: el *clustering* jerárquico, el procesamiento de imágenes o el reconocimiento de reglas en minería de datos.

Por otra parte, con la intención de facilitar la utilización del algoritmo PSO propuesto, se ha seguido para su implementación la arquitectura basada en esqueletos de código de la biblioteca MALLBA [7] (disponible en `http://neo.lcc.uma.es/mallba/easy-mallba/index.html`). Mediante esta arquitectura, se proveen clases para la implementación de nuevos problemas de optimización. Además, se han desarrollado versiones de PSO implementadas en MALLBA para la ejecución en modo secuencial y en modo paralelo (LAN/WAN). En concreto, para este trabajo se ha resuelto el problema de ordenación de genes en MAs mediante el esqueleto de código PSO en sus diferentes versiones (secuencial y paralela), obteniendo soluciones de alta calidad comparables con los encontrados en la literatura y en un tiempo de ejecución razonable.

El resto de este artículo se organiza de la siguiente forma: En la Sección II, se realiza una introducción al proceso de análisis de microarrays de ADN. En la Sección III, se presenta el algoritmo PSO y se introduce una versión el mismo para permutaciones de enteros. En la Sección IV se detallan aspectos sobre la representación de las soluciones y los operadores. La Sección V describe la función de evaluación utilizada y en la Sección VI, el esquema de paralelización configurado en las versiones paralelas del mismo. En la Sección VII se presentan en primer lugar los experimentos realizados y a continuación se muestran los resultados obtenidos. Por último, en la Sección VIII se incluyen conclusiones y el trabajo futuro que tiene sentido a la luz de estos resultados.

II. Análisis de Microarrays de ADN

Un microarray o "chip" de ADN es una plantilla de vidrio (o un substrado sólido) en la cual se disponen de manera sistemática cientos o miles de muestras de moléculas de ADN. Posteriormente, sobre este MA se realiza un proceso llamado hibridación

Grupo GISUM, Dpto. de Lenguajes y Ciencias de la Computación, E.T.S.I Informática, UMA, E-mail: {eat,jnieto,gabriel}@lcc.uma.es .

que consiste en la exposición de las moléculas de ADN a moléculas de ADN complementario (cADN) obtenidas a partir de ARN mensajero (mARN). Estas moléculas de mARN se marcan con tintes de diferentes colores (normalmente rojo para genes expresados en condiciones normales y verde para genes expresados en condiciones anómalas). Las moléculas de ADN y cADN son emparejadas mediante pares de bases. En este proceso, las moléculas de cADN que no formen pareja con ningún gen serán eliminadas del MA. Por último, mediante un escáner se obtiene una imagen del MA midiendo los niveles de color rojo/verde (tonos de gris en la Fig. 1) de cada gen. Este nivel de coloración indica el nivel de expresión de cada muestra.

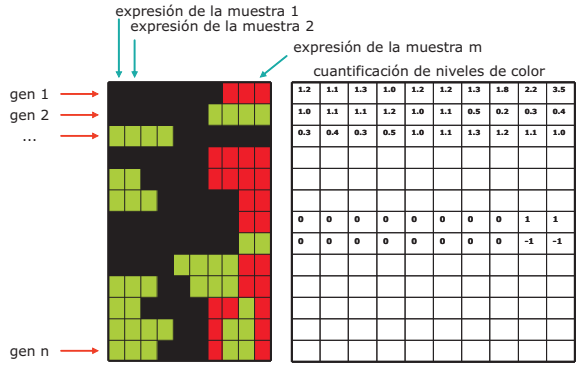


Fig. 1
Estructura de un MA y matriz de expresiones de muestras cuantificadas.

El MA resultado puede ser expresado como una matriz $G = \{g_{ij}\}_{j=1\ldots m}^{i=1\ldots n}$, donde n es el número de genes y m es el número de muestras por gen. Estas muestras corresponden al estado de los genes bajo diferentes condiciones o en diferentes instantes de tiempo. Así, el estado de cada muestra viene representado por su nivel de coloración y se cuantifica mediante un número real.

El objetivo ahora es encontrar un orden óptimo de los genes en el MA de manera que los genes con patrones de expresión similares aparezcan en posiciones cercanas en dicho orden. El grado de similitud (o desiguadad) entre dos genes se puede obtener mediante la distancia entre ambos. En la literatura podemos encontrar diferentes medidas de distancia como la correlación de Pearson (utilizada en [8]), la correlación τ de Kendall o la correlación de Spearman. En este trabajo se ha utilizado la medida de distancia propuesta en [3]. Esta métrica está basada en la distancia Euclídea a la cual se le ha añadido un mecanismo de ventana deslizante (Sección V).

III. Algoritmos Basados en Cúmulos de Partículas

Los algoritmos basados en cúmulos de partículas o Particle Swarm Optimization (PSO), es una técnica metaheurística basada en población e inspirada en el comportamiento social del vuelo de las bandadas de aves o el movimiento de los bancos de peces. PSO fue originalmente desarrollado por J. Kennedy y por R. Eberhart en 1995, basándose en un concepto conocido como la *"metáfora social"* [5].

En la búsqueda de una solución óptima o cuasi-óptima, PSO actualiza el cúmulo actual de partículas (cada partícula es un candidato a solución de un problema) utilizando información acerca de la mejor solución obtenida por cada partícula y la mejor solución obtenida en el cúmulo entero. Cada partícula tiene los siguientes atributos: la velocidad actual, la posición actual, la mejor posición obtenida por la partícula hasta el momento y la mejor posición encontrada por los vecinos de la partícula hasta el momento. El vecindario de una partícula puede ser *global*, en el cual todas las partículas del cúmulo son consideradas vecinas entre sí, o *local*, en el que sólo son vecinas las partículas inmediatamente cercanas. En la primera fase del algoritmo, se inicializa aleatoriamente la velocidad y la posición de cada partícula del cúmulo. En la segunda fase, para cada partícula i del cúmulo se actualizan la velocidad (v_i) y la posición (x_i) mediante las siguientes ecuaciones:

$$v_i = \omega \cdot v_i + \varphi_1 \cdot (pBest_i - x_i) + \varphi_2 \cdot (g_i - x_i) \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

donde ω es el factor de inercia [9], φ_1 y φ_2 son números aleatorios, $pBest_i$ es la mejor posición encontrada por la partícula i hasta el momento y g_i es la mejor posición encontrada hasta el momento en el vecindario de dicha partícula.

PSO ha sido aplicado con éxito en diferentes campos de investigación para la resolución de problemas de optimización. Algunos ejemplos son: optimización de funciones numéricas [10], entrenamiento de redes neuronales [11], aprendizaje de sistemas difusos [12] y registrado de imágenes [13]. La mayoría de estos problemas requieren codificación continua y por tanto, aún no existe un gran número de propuestas de PSO para trabajar con otro tipo de codificación (como la binaria o para permutaciones de enteros). En este trabajo se presenta una versión de PSO para la resolución de problemas que requieran codificación para permutaciones de enteros basada en la propuesta realizada por Clerc en [14]. En concreto, esta versión de PSO se ha utilizado para resolver el problema de reordenación de genes en MAs de ADN.

En el Algoritmo 1, se muestra el pseudocódigo del algoritmo PSO para permutaciones de enteros utilizado. Como puede observarse, en primer lugar se realiza una inicialización de todas las partículas que forman el cúmulo. En una segunda fase se actualizan las mejores posiciones encontradas hasta el momento por cada partícula ($pBest_i$), así como las mejores posiciones del vecindario (g_i). Además, se actualizan los correspondientes valores de aptitud (*fitness*). Por último, se realizan las actualizaciones de velocidad y

Algoritmo 1 PSO para permutaciones de enteros

```
S ← InicializarCumulo()
while no se alcance la condición de parada do
  for i = 1 to size(S) do
    evaluar cada partícula x_i del cúmulo S
    if fitness(x_i) es mejor que fitness(pBest_i) then
      pBest_i ← x_i; fitness(pBest_i) ← fitness(x_i)
    end if
    if fitness(pBest_i) es mejor que fitness(g_i) then
      g_i ← pBest_i; fitness(g_i) ← fitness(pBest_i)
    end if
  end for
  for i = 1 to size(S) do
    v_i ← v_i ∘ φ1 ⊗ (pBest_i ⊖ x_i) ∘ φ2 ⊗ (g_i ⊖ x_i)
    x_i ← x_i ⊕ v_i
  end for
end while
Salida: la mejor solución encontrada
```

de posición para cada partícula. Las ecuaciones empleadas en esta última fase se diferencian de las ecuaciones 1 y 2 en los operadores de suma ($\oplus$ y $\circ$), diferencia ($\ominus$) y producto ($\otimes$), adaptados para que trabajen con permutaciones de enteros. Pero antes de describir estos nuevos operadores es necesario aclarar cómo se representan las partículas en PSO para permutaciones de enteros.

IV. Representación de Partículas y Operadores

La posición x de una partícula en el espacio de soluciones se representa mediante un vector de n valores enteros sin que existan repeticiones ni omisiones. Cada entero del vector posición representa un gen en el MA con sus correspondientes muestras. De este modo, la disposición de los enteros del vector posición determina la ordenación de los genes en el MA que representa. Por lo tanto, la posición de una partícula contiene una solución al problema, siendo n la longitud de dicha solución. Un sencillo ejemplo de longitud 6 puede ser:

$$x = (2, 4, 3, 6, 1, 5)$$

Lo que significa que el primer gen en el MA sería el 2, a continuación estaría el gen número 4, después el 3 y así hasta el último gen del MA que sería el 5.

La velocidad v de una partícula se representa mediante una lista de pares de enteros $(i \rightarrow j)$. Cada uno de estos pares representa un intercambio a realizar sobre los elementos de la posición x. Por ejemplo, si aplicamos el intercambio del par $(4 \rightarrow 1)$ a la posición anterior obtendremos la nueva posición $x = (2, 1, 3, 6, 4, 5)$. Así, el movimiento de una partícula viene dado por la sucesiva aplicación de pares de la lista velocidad a la lista posición. Un ejemplo de lista velocidad es el siguiente:

$$v = [(3 \rightarrow 1), (6 \rightarrow 6), (1 \rightarrow 5)]$$

Aplicando esta lista v a la posición anterior se obtiene: en primer lugar $x' = (2, 4, 1, 6, 3, 5)$ al aplicar $(3 \rightarrow 1)$ y en segundo lugar $x'' = (2, 4, 5, 6, 3, 1)$ al aplicar sobre x' el par $(1 \rightarrow 5)$. El par $(6 \rightarrow 6)$ no realiza ninguna modificación sobre la posición. Por este motivo, se toma como el tamaño de la velocidad $|v|$ el número de pares que realizan intercambios, excluyendo las operaciones identidad, es decir, los pares $(i \rightarrow i)$. Para el ejemplo anterior $|v| = 2$.

A continuación, se describen los operadores utilizados en la actualización de la posición y la velocidad de una partícula dada.

A. Operador Diferencia de Posiciones ($\ominus$)

Al restar dos posiciones el resultado es una lista velocidad, es decir, una lista de pares. Un par i está formado por el i-ésimo valor del segundo operando y el i-ésimo valor de primer operando de la resta. Por ejemplo, si restamos las siguientes posiciones:

$$(2, 4, 3, 6, 1, 5) \ominus (5, 1, 3, 2, 4, 6),$$

el resultado será la velocidad:

$$[(5 \rightarrow 2), (1 \rightarrow 4), (3 \rightarrow 3), (2 \rightarrow 6), (4 \rightarrow 1), (6 \rightarrow 5)]$$

B. Operador Suma de Velocidades ($\circ$)

La suma de dos velocidades consiste en ir "solapando" los pares de las listas de dichas velocidades, de manera que se obtiene una nueva lista de pares. Para que se produzca un nuevo par, deben coincidir los valores final e inicial de los pares involucrados, es decir, los pares $(i \rightarrow j)$ y $(j \rightarrow k)$ se solapan formando el par $(i \rightarrow k)$. Si dos pares correspondientes no se solapan, el resultado será el primer par. A continuación se muestra un ejemplo completo de suma de velocidades:

$$[(1 \rightarrow 1), (2 \rightarrow 5), (5 \rightarrow 4), (1 \rightarrow 3)]$$
$$\circ$$
$$[(1 \rightarrow 1), (5 \rightarrow 1), (4 \rightarrow 3), (2 \rightarrow 4)]$$

dando como resultado

$$[(1 \rightarrow 1), (2 \rightarrow 1), (5 \rightarrow 3), (1 \rightarrow 3)]$$

C. Operador Producto Coeficiente Velocidad ($\otimes$)

Mediante el producto coeficiente velocidad se realiza la multiplicación de un coeficiente real φ y una lista velocidad. Se dan varios casos:

- si $\varphi \epsilon [0, 1]$, se obtiene $\varphi' = rand(0, 1)$ para
$$\begin{cases} \varphi' \leq \varphi \Rightarrow (i \rightarrow j) \rightarrow (i \rightarrow i) \\ \varphi' > \varphi \Rightarrow (i \rightarrow j) \rightarrow (i \rightarrow j) \end{cases}$$
- si $\varphi > 1$, tal que $\varphi = k + \varphi'$, con k entero y $\varphi' < 1$, se realiza *velocidad más velocidad k* veces y *coeficiente φ' por velocidad* una vez.

Por ejemplo, si multiplicamos:

$$0.5 \otimes [(1 \rightarrow 3), (2 \rightarrow 5), (4 \rightarrow 4), (3 \rightarrow 2)],$$

para la secuencia de valores aleatorios (φ') 0.2, 0.8, 0.5 y 0.3 se obtiene como resultado:

$$[(1 \rightarrow 1), (2 \rightarrow 5), (4 \rightarrow 4), (3 \rightarrow 3)]$$

D. Operador Suma de Posición con Velocidad ($\oplus$)

Mediante este operador se realiza el proceso de intercambiar los valores de la posición de una partícula para generar una nueva posición. Para cada par

$(i \rightarrow j)$ de la lista de velocidad, se lleva a cabo en el vector posición un intercambio de los elementos i y j. El resultado es la nueva posición a la que se mueve la partícula tras la realización de sucesivos intercambios. Por ejemplo, sean v una lista velocidad y x la posición de la partícula, al sumarlas:

$$x = (3, 5, 2, 6, 4, 1)$$
$$\oplus$$
$$v = [(2 \rightarrow 1), (4 \rightarrow 2), (3 \rightarrow 3)]$$

se obtiene como resultado

$$x = (3, 5, 1, 6, 2, 4)$$

E. Búsqueda Local

Uno de los principales inconvenientes de PSO es que debido al rápido incremento de la velocidad de las partículas, el algoritmo podría caer fácilmente en óptimos locales perjudicando a la exploración en el espacio de búsqueda. Para evitar esto se incorporan mecanismos para controlar el crecimiento de la velocidad como la inercia o el factor de constricción [9]. En el PSO desarrollado se ha incorporado además un mecanismo de búsqueda local para forzar cambios de direcciones en la evolución del cúmulo. Se trata de una búsqueda de un nivel realizada sobre la mejor partícula del cúmulo en cada iteración. Esta búsqueda proporciona nuevas soluciones, provocando que todas las demás cambien su dirección (velocidad) hacia ella. Metafóricamente, la nueva partícula toma el testigo de lider y dirige al resto del cúmulo hacia nuevas regiones del espacio de búsqueda. De este modo se consigue una mejor exploración y se evita la caída prematura en óptimos locales.

V. Función de Evaluación

Como se ha mencionado en la Sección II, una solución deseable en el problema de ordenación de genes en MAs (es decir, una buena permutación) tendrá los genes similares agrupados juntos en regiones (o *clusters*). Por lo tanto, es necesario definir una función de distancia para medir el grado de similitud entre los genes. Se puede considerar en primer lugar la distancia Euclídea entre cada par de genes adyacentes en el MA. La distancia total viene dada por la suma de todas las distancias Euclídeas parciales, de manera similar a como se hace en el problema del Viajante de Comercio.

El inconveniente de este mecanismo es que sólo utiliza información de los genes inmediatamente adyacentes. Esta función tiene una visión bastante "miope" de una solución, provocando que algunas soluciones con genes agrupados en conjuntos disjuntos reducidos sean evaluadas como buenas. Para realizar un buen agrupamiento de genes es necesario tener una visión "panorámica". En este tipo de situaciones, se suelen utilizar "ventanas flotantes". La función distancia total está formada en este caso por dos términos: el primer término suma las distancias entre el gen situado en el centro de la ventana y los genes situados dentro de la longitud de la ventana. El segundo término suma las distancias parciales según se va moviendo la ventana a lo largo de la secuencia. De este modo, siendo $\pi = \langle \pi_1, \pi_2, ...\pi_n \rangle$ el orden de los n genes de un vector solución, se representa la función de distancia total (DT) con ventana flotante como sigue:

$$DT(\pi) = \sum_{l=1}^{n} \sum_{i=max(l-s_w,1)}^{min(l+s_w,n)} w(i,l) D[\pi_l, \pi_i] \quad (3)$$

$$w(i,l) = s_w - |l - i| + 1 \quad (4)$$

$$D[\pi_i, \pi_j] = \sqrt{\sum_{k=1}^{m} (\pi_{ik} - \pi_{jk})^2} \quad (5)$$

donde $2s_w + 1$ es el tamaño de la ventana, y $w(i,l)$ es una función de peso que mide la influencia del gen situado en la posición l sobre el gen situado en la posición i. Considerando los pesos proporcionales a $s_w - |l - i| + 1$ (es decir, lineal en la distancia entre genes) y normalizado para que la suma de todos los pesos implicados en cada aplicación de la Ecuación 3 sea 1.

Respecto al tamaño de la ventana, experimentos realizados en [3] indican que una elección del parámetro s_w entre el 5 % y el 10 % del tamaño de la instancia proporciona un buen compromiso entre la calidad de la solución y el coste computacional.

VI. Esquema de Paralelización

Por lo general, el trabajo con MAs de ADN supone el manejo de una gran cantidad de datos obtenidos en experimentos previos. Por este motivo, tareas como la evaluación de las soluciones y la búsqueda local pueden requerir un gran esfuerzo computacional. Mediante la paralelización del algoritmo PSO se consigue reducir el tiempo de ejecución y agilizar la obtención de resultados finales. En este sentido, siguiendo la arquitectura basada en esqueletos de código de la biblioteca MALLBA, se ha implementado en este trabajo una serie de versiones paralelas (LAN/WAN) de PSO siguiendo un modelo descentralizado de grano grueso [7] [15]. En estas versiones se ha empleado una topología de anillo en la que cada proceso del algoritmo se comunica con el proceso adyacente según un orden configurado previamente. En la Fig. 2 se puede observar un esquema de la topología empleada en la paralelización de PSO.

En esta topología se distinguen dos tipos de procesos: los que participan en la exploración del algoritmo (desde P1 hasta P5 en Fig. 2) y el proceso maestro (P0). Este proceso maestro se centra en realizar únicamente tareas de recolección de información acerca del estado local del resto de los procesos y a partir de ella mantener un estado global del algoritmo paralelizado. Este proceso no consume CPU debido a que no realiza espera activa y está bloqueado la mayor parte del tiempo, desbloqueándose solamente cuando se detecta la entrada de un mensaje (flecha discontinua).

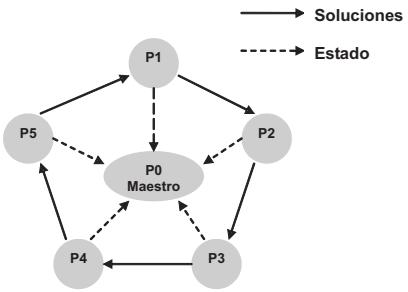


Fig. 2
Topología empleada en la paralelización de PSO.

El intercambio de soluciones (flecha continua) entre procesos se puede realizar de manera síncrona o asíncrona dependiendo de un parámetro configurable. Este intercambio de soluciones se realizará con frecuencia de migración configurada inicialmente. Cada subalgoritmo envía la mejor solución encontrada hasta el momento en el subcúmulo sobre el que trabaja y reemplaza la solución recibida por la peor solución del mismo subcúmulo. El comportamiento del PSO paralelo es diferente al del PSO secuencial. La principal razón es que la influencia del vecindario de cada partícula se reduce al subcúmulo al que pertenece, al contrario que en PSO secuencial donde el vecindario puede abarcar la totalidad de las partículas en proceso. Por este mismo motivo, la mejor partícula global o local en cada momento puede variar dependiendo de la frecuencia de migración y el tamaño del vecindario de cada partícula.

VII. Experimentos y Resultados Computacionales

En esta sección se describen una serie de experimentos realizados para evaluar el algoritmo PSO para permutaciones de enteros tanto en su versión secuencial como paralela. Además, se evalúa la calidad de las soluciones finales obtenidas en la resolución del problema de ordenación de genes. El algoritmo PSO secuencial fue ejecutado en un PC con sistema operativo Linux (distribución Suse 9.0 con kernel 2.4.19) y equipado con un procesador Pentium IV a 2.8GHz, 512MB de RAM, y 60GB de disco duro. Para la versión paralela de PSO se utilizó hasta 6 PCs del mismo tipo que para la versión secuencial interconectados mediante una red de comunicaciones (LAN) FastEthernet 10/100 Mbps. El software de comunicaciones empleado consta de la biblioteca *Netstream* [16], mediante la que se ofrece una interfaz orientada a objetos sobre la biblioteca estándar de paso de mensajes MPI.

Las instancias de MAs de ADN utilizadas fueron obtenidas de secuencias de genes reales disponibles en la literatura. Estas instancias incluyen diferentes cantidades de genes y diferentes cantidades de muestras por gen, permitiendo así la evaluación del algoritmo bajo diferentes escenarios. A continuación se describen estas instancias:

- HERPES: estos datos fueron obtenidos de [17]. Se trata de muestras obtenidas en el proceso de inducción del sarcoma asociado al virus *herpes de Kaposi* durante el periodo de latencia y después de la inducción en la replicación lítica. Comprenden 106 genes (21 muestra por gen).

- FIBROBLAST: estos datos fueron obtenidos de [18]. Corresponden a la respuesta del *fibroblasto* humano a la aplicación de suero en la cicatrización. Comprenden 517 genes (19 muestras por gen).

- DIAUXIC: estos datos fueron obtenidos de [19]. Corresponden a un experimento hecho por un grupo de Stanford sobre el *salto diauxico*, es decir, el paso de condiciones aerobias a condiciones anaerobias en el *Saccharomyces cerevisiae* (levadura de cerveza). Comprenden 210 genes (7 muestras por gen).

En cuanto a la configuración del algoritmo, en la Tabla I se muestran los parámetros principales introducidos en la versión secuencial y paralela de PSO.

TABLA I
Configuración de parámetros en PSO secuencial y paralelo.

Parámetro	Secuencial	Paralelo
Tamaño del Cúmulo	100	20
Tamaño del Vecindario	8	8
Límites de Inercia	(-10,10)	(-10,10)
Velocidad (Mín,Máx)	(0.4,1.4)	(0.4,1.4)
Frecuencia de Migración	-	8
Número de Procesos	1	(5+1 Maestro)

Para cada una de estas instancias se han realizado 30 ejecuciones independientes de PSO en las versiones secuencial, paralelo en modo síncrono y paralelo en modo asíncrono. Cada ejecución realiza 500 iteraciones más una operación de búsqueda local en cada iteración.

En la Tabla II se disponen los resultados obtenidos según la siguiente nomenclatura:

- **M** es la distancia mínima global obtenida (según la Ecuación 3),
- **MV** es la media de las distancias mínimas encontradas en cada ejecución independiente,
- **PE** es el porcentaje de error calculado como (mejor conocido - mejor encontrado) / mejor conocido,
- **E** es la evaluación en la que se encontró la menor distancia,
- **T** es el tiempo (en segundos) en que se encontró la menor distancia,

Además, se ha incluido en esta tabla una entrada correspondiente a otros resultados disponibles de la literatura actual [3]. En concreto se trata de resultados (media de diez ejecuciones independientes)

TABLA II

Resultados obtenidos por el algoritmo PSO secuencial (PSO SEQ), PSO paralelo síncrono (PSO LAN SYNC), paralelo asíncrono (PSO LAN ASYNC) y Algoritmo Memético.

Instancia	Algoritmo	M	MV	PE	E	T
HERPES	PSO SEQ	500.478	**554.202**	0.00000 %	39800	707.54 *s*
	PSO LAN SYNC	497.459	**559.080**	0.00000 %	50000	1586.71 *s*
	PSO LAN ASYNC	496.263	**552.345**	0.00000 %	41900	375.03 *s*
	Memético	-	598.798	-	-	-
FIBROBLAST	PSO SEQ	1359.680	1391.060	0.01035 %	50000	21436.40 *s*
	PSO LAN SYNC	1359.680	**1362.670**	0.00000 %	47000	16652.60 *s*
	PSO LAN ASYNC	1362.980	1386.700	0.00718 %	50000	26972.40 *s*
	Memético	-	1376.804	-	-	-
DIAUXIC	PSO SEQ	355.047	**388.323**	0.00000 %	49700	797.55 *s*
	PSO LAN SYNC	349.232	**381.630**	0.00000 %	49900	909.14 *s*
	PSO LAN ASYNC	345.796	**374.734**	0.00000 %	50000	2206.16 *s*
	Memético	-	-	-	-	-

obtenidos por un Algoritmo Memético con diferentes niveles de búsqueda local.

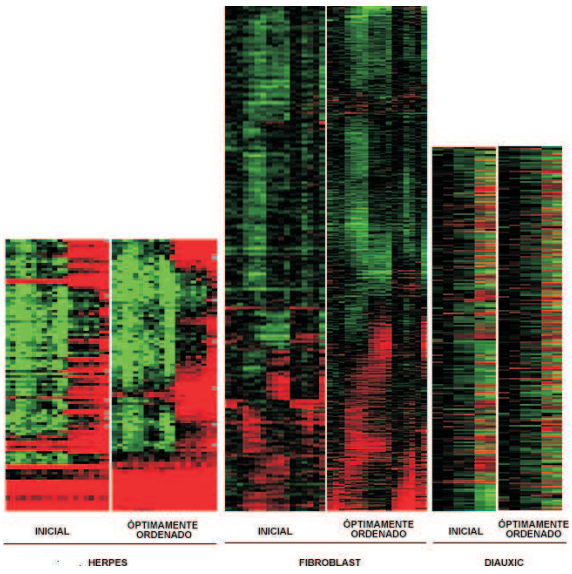
A partir de los resultados mostrados en la Tabla II se extraen varias conclusiones. Examinando la calidad de las soluciones encontradas se puede ver que el algoritmo PSO desarrollado en este trabajo resuelve adecuadamente el problema de la ordenación de genes en MAs, encontrando resultados similares e incluso mejores que los obtenidos por el Algoritmo Memético referenciado anteriormente. En cuanto a las diferentes versiones del PSO, se observa que el PSO paralelo se comporta mejor que el secuencial, ya que el modelo en islas permite en este caso conservar una mayor diversidad evitando la convergencia prematura. En concreto, la versión asíncrona es la que proporciona mejores resultados (en media). Al ser la ordenación de genes en MAs un problema que trabaja con grandes instancias, la evaluación de una solución comprende gran parte de la carga de proceso. Por este motivo, la paralelización del algoritmo en islas que trabajan con subcúmulos más pequeños agiliza de manera considerable la ejecución de una iteración del algoritmo. Añadido a que la comunicación se realiza de manera asíncrona, las islas disponen rápidamente de nuevas partículas migradas, lo que contribuye, como se mencionó anteriormente, a mantener la diversidad dentro de cada subcúmulo.

Una de las mejores formas de comprobar el efecto de la ordenación de genes en un MA es comparando la disposición de éstos antes y después de la ordenación. En la Fig. 3, se muestran las imágenes de los MAs de las instancias procesadas con el MA inicial a la izquierda y el MA óptimamente ordenado mediante PSO a la derecha. Como se puede observar, los MAs ordenados presentan mayores áreas homogéneas en la distribución de los niveles de color, facilitando de este modo la posterior clusterización.

Otro aspecto interesante consiste en observar la evolución que experimenta el algoritmo a lo largo de una ejecución. En la Fig. 4, se muestra una gráfica de la evolución en la que se comprueba cada 50 iteraciones el mejor resultado que obtiene. Los valores presentados corresponden a las ejecuciones del PSO secuencial, el PSO paralelo en modo síncrono y

Fig. 3

Microarrays inicial y óptimamente ordenado de las tres instancias objeto de estudio.



el PSO paralelo en modo asíncrono. Para el problema se ha utilizado la instancia HERPES, aunque el perfil de la gráfica resultado es similar en las demás instancias.

Como se ilustra en esta gráfica, el algoritmo PSO, tanto en la versión secuencial como en las versiones paralelas, realiza una rápida mejora sustancial de las soluciones en las primeras iteraciones. Alrededor del punto medio de la ejecución empieza a converger hasta que deja de incorporar nuevas soluciones. Entre las iteraciones 50 y 300 se observa una etapa de refinamiento de las soluciones producida probablemente gracias al factor de inercia y a la búsqueda local que colaboran retrasando una convergencia prematura.

Por último, hemos realizado pruebas estableciendo como final de la ejecución un cierto valor de resultado para medir el *speedup* (S). Para esto se ha utilizado la fórmula estándar del *speedup* (Ecuación 6) en la que p es el número de procesadores, $\overline{T_1}$ es el tiempo medio de ejecución del algoritmo paralelo en un procesador y $\overline{T_p}$ es el tiempo medio de ejecución

TABLA III

Speedup y *Eficiencia* calculados respecto a las ejecuciones secuenciales y paralelas en modo síncrono (s/ls) y respecto a las ejecuciones secuenciales y paralelas en modo asíncrono (s/las).

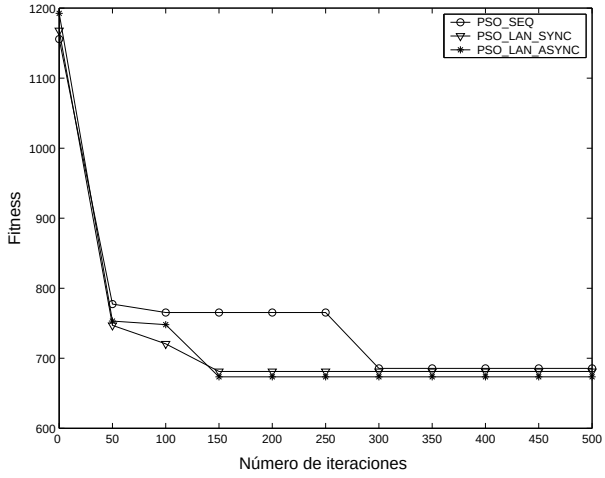
Instancia	$Speedup_{s/ls}$	$Speedup_{s/las}$	$Eficiencia_{s/ls}$	$Eficiencia_{s/las}$	**Final**
HERPES	2.525	3.569	50 %	71 %	630
FIBROBLAST	2.786	2.834	55 %	56 %	1600
DIAUXIC	2.529	2.116	50 %	42 %	500

TABLA IV

Speedup y *Eficiencia* calculados respecto a las ejecuciones paralelas en modo síncrono utilizando uno y cinco procesadores (s *1,5*) y respecto a las ejecuciones paralelas en modo asíncrono utilizando uno y cinco procesadores (a *1,5*).

Instancia	$Speedup_{s1,5}$	$Speedup_{a1,5}$	$Eficiencia_{s1,5}$	$Eficiencia_{a1,5}$	**Final**
HERPES	2.299	3.952	46 %	70 %	630
FIBROBLAST	1.444	1.992	28 %	40 %	1600
DIAUXIC	2.856	2.621	57 %	52 %	500

Fig. 4
Mejor resultado obtenido por el PSO secuencial (PSO_SEQ), el PSO paralelo en modo síncrono (PSO_LAN_SYNC) y el PSO paralelo en modo asíncrono (PSO_LAN_ASYNC) para la instancia HERPES.



del algoritmo paralelo con p procesadores.

$$S = \frac{\overline{T_1}}{\overline{T_p}} \tag{6}$$

De esta forma, se obtendría un *speeup ideal* cuando $S = p$ lo cual se considera un buen factor de escalabilidad. En nuestro caso, el *speeup ideal* sería 5 pues es el número de procesadores que utilizamos en las versiones paralelas.

Para obtener una estimación de cómo de buena es la paralelización realizada, hemos utilizado la Ecuación 7 mediante la que se calcula el tanto por ciento de la eficiencia (E) lograda. Así, podemos hacernos una idea del esfuerzo computacional empleado en tareas como la sincronización y la comunicación entre procesos.

$$E = \frac{S}{p} \times 100 \tag{7}$$

En la Tabla III se presentan los valores de *speedup* y *eficiencia* calculados respecto a las ejecuciones secuenciales y paralelas (síncronas/asíncronas). En la Tabla IV se presentan los valores de *speedup* y *eficiencia* calculados respecto a las ejecuciones paralelas (síncronas/asíncronas) con uno y cinco procesadores. En estas ejecuciones se ha empleado como condición de parada un valor de resultado predefinido (columna **Final** en las tablas). Según se observa en ambas tablas, las versiones paralelas ejecutadas en cinco procesadores consiguen agilizar el proceso de cálculo del algoritmo aportando *eficiencias* alrededor del 50 %. Especialmente, la versión paralela asíncrona aporta los mejores valores de *speedup* y *eficiencia* bebido a que no realiza sincronización de los procesos a la hora de migrar soluciones.

VIII. Conclusiones

Este trabajo se presenta un algoritmo PSO para realizar una eficiente ordenación de genes en MAs de ADN. Las principales características de este algoritmo son la representación de partículas para la codificación de soluciones mediante permutaciones de enteros y la utilización de operadores *ad-hoc* para trabajar con esta representación. Para su desarrollo se ha seguido la arquitectura de esqueletos de código propuesta en la biblioteca MALLBA, facilitando su utilización a futuros usuarios. Además, se proporcionan versiones secuenciales y paralelas del mismo.

Se han realizado experimentos sobre el PSO desarrollado en sus distintas versiones, obteniendo soluciones de alta calidad y en tiempos de ejecución razonables. Para ello, se han utilizando instancias reales de MA encontradas en la literatura.

Posiblemente, parte del buen funcionamiento del esqueleto PSO para permutaciones de enteros reside en la naturaleza de los operadores. Casualmente, para este problema trabajan de manera adecuada, aunque es necesario estudiar si este comportamiento

se mantiene para otros problemas que utilizan representaciones con permutaciones de enteros. De esta forma, por ejemplo, el operador diferencia de posiciones ($\ominus$ en la Subsección IV-A) introduce un gran número de pares de intercambios en el vector velocidad. Con esto se consigue una gran variabilidad en las permutaciones obtenidas como solución tras el movimiento de las partículas, y de este modo, diversificar el conjunto de soluciones en proceso. Unido a la búsqueda local de un nivel realizada sobre la mejor partícula en cada iteración se consigue cierto grado de intensificación, lo que ayuda a la obtención de óptimos que desvían la dirección general del cúmulo.

Como futuro trabajo, se estudiarán nuevas versiones de PSO para permutaciones de enteros que utilizen operadores más avanzados. Por ejemplo, la representación *Fuzzy* en PSO está teniendo buenos resultados en problemas de permutaciones [14], [20].

Respecto al problema de ordenación de genes, podría evaluarse el esqueleto con muchas más instancias reales de MAs y realizar un estudio más extenso de los resultados. Además, la incorporación de una nueva clase (o método) para la realización de *clustering* jerárquico a partir de la ordenación de genes resultado proporcionaría una funcionalidad añadida a la implementación de este problema.

Agradecimientos

Este trabajo está parcialmente financiado por el Ministerio de Educación y Ciencia y FEDER con número de proyecto TIN2005-08818-C04-01 (proyecto OPLINK `http://oplink.lcc.uma.es`).

Referencias

[1] R. Fleischmann, O. Adams, R. White, E. Clayton, A. Kirkness, C. Kerlavage, and et al., "Whole-Genome Random Sequencing and Assembly of *Haemophylus Influenzae*," *Science*, vol. 269, pp. 496–5 1 2, 1995.

[2] P. Brown and D. Botstein, "Exploring the New World of the Genome with DNA Microarrays," *Nature Genetics*, vol. 21, pp. 33–37, 1999.

[3] C. Cotta, A. Mendes, V. Garcia, P. França, and P. Moscato, "Applying Memetic Algorithms to the Analysis of Microarray Data," in *Proceedings of the Applications of Evolutionary Computing*, Berlin, 2003, vol. 2611 of *Lecture Notes in Computer Science*, pp. 22–32, Springer-Verlag.

[4] M. Eisen, P. Spellman P. Brown, and D. Botstein, "Cluster Analysis and Display of Genome-Wide Expression Patterns," in *Proceedings of the National Academy of Sciences of the USA*, 1998, vol. 95, pp. 14863–14868.

[5] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia, Nov 1995, vol. 4, pp. 1942–1948.

[6] C. Cotta and P. Moscato, "A Memetic-Aided Approach to Hierarchical Clustering from Distance Matrices: Application to Gene Expression Clustering and Phylogeny," *Byosystems*, vol. 72(1-2), pp. 75–97, 2003.

[7] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa, "Efficient Parallel LAN/WAN Algorithms for Optimization. The MALLBA Project," *Parallel Computing*, vol. 32, no. 5-6, pp. 415–440, June 2006.

[8] H. Tsai, J. Yang, and C. Kao, "Applying Genetic Algorithms to Finding the Optimal Gene Order in Displaying the Microarray Data," in *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, USA, 2002, pp. 610–617, Morgan Kaufmann Publishers.

[9] R. Eberhart and Y. Shi, "Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization," in *Proceedings of the International Congress on Evolutionary Computation*, July 2000, vol. 1, pp. 84–88.

[10] X. Xie, W. Zhang Z., and Yang, "Solving Numerical Optimization Problems by Simulating Particulates in Potential Field with Cooperative Agents," in *International Conference on Artificial Intelligence*, Las Vegas, NV, USA, 2002.

[11] G. Gudise and G. Venayagamoorthy, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks," in *Proceedings of the IEEE Swarm Intelligence Symposium 2003*, Indianapolis, Indiana, USA, 2003, pp. 110–117.

[12] K. Parsopoulos, E. Papageorgiou, P. Groumpos, and M. Vrahatis, "A First Study of Fuzzy Cognitive Maps Learning Using Particle Swarm Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation 2003*, Canbella, Australia, 2003, pp. 1440–1447.

[13] M. Omran, A. Salman, and A. Engelbrecht, "Image Classification Using Particle Swarm Optimization," in *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning 2002*, Singapore, 2002, pp. 370–374.

[14] M. Clerc, *Discrete Particle Swarm Optimization. Illustrated by the Traveling Salesman Problem*, In URL `http://clerc.maurice.free.fr/pso/pso_tsp/Discrete_PSO_TSP.htm`, Apr 2003.

[15] E. Alba, Ed., *Parallel Metaheuristics: A New Class of Algorithms*, Wiley, 2005.

[16] E. Alba, "Netstream: A Flexible and Simple Message Passing Service for LAN/WAN Utilization," *E.T.S.I.I. Málaga. Departamento de Lenguajes y Ciencias de la Computación*, 2001.

[17] R. Jenner, M. Alba, C. Boshoff, and P. Kellam, "Kaposi's Sarcoma-Associated Herpesvirus Latent and Lytic Gene Expression as Revealed by DNA Arrays," *Journal of Virology*, vol. 2, pp. 891–902, 2001.

[18] Iyer et al., "The Transcriptional Program in the Response of Human Fibroblasts to Serum," *Science*, vol. 283, pp. 83–87, 1999.

[19] J. DeRisi, V. Iyer, and P. Brown, "Exploring the Metabolic and Genetic Control of Gene Expression on a Genomic Scale," *Science*, vol. 278, no. 5338, pp. 680–686, 1994.

[20] Wei Pang, Kang ping Wang, Chun guang Zhou, and Long jiang Dong, "Fuzzy Discrete Particle Swarm Optimization for Solving Traveling Salesman Problem," *cit*, vol. 00, pp. 796–800, 2004.