XIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (XIII MAEB)

MAEB 8.2: Sesión Especial: Algoritmos Paralelos

Organizadores:

Enrique Alba y Gabriel Luque

Explotación de Paralelismo Multinivel e Híbrido en Metaheurísticas Híbridas

José M. Cecilia, Baldomero Imbernón
Bioinformatics and High Performance Computing Research Group (BIO-HPC)
Polytechnic School, University Católica San Antonio of Murcia (UCAM)
Murcia, Spain
{jmcecilia,bimbernon}@ucam.edu

Javier Cuenca
Department of Engineering and Technology of Computers
University of Murcia
Murcia, Spain
jcuenca@um.es

José-Matías Cutillas-Lozano, Domingo Giménez
Department of Computing and Systems
University of Murcia
Murcia, Spain
{josematias.cutillas,domingo}@um.es

***Resumen*—Los sistemas computacionales actuales están formados por nodos que comprenden CPUs *multicore* junto con uno o varios coprocesadores (normalmente GPUs pero ocasionalmente también MICs), de forma que se dispone de un sistema híbrido y heterogéneo. Por otro lado, el desarrollo de metaheurísticas híbridas y de hiperheurísticas que trabajan sobre ellas también sigue una estructura híbrida, en la que se puede explotar paralelismo multinivel para llevar a cabo computaciones con distinto volumen de cómputo (heterogéneas). En este trabajo analizamos la combinación de paralelismo híbrido y heterogéneo a los dos niveles, de software y de hardware, en la aplicación de metaheurísticas híbridas. Se realizan experimentos con la aplicación de metaheurísticas a dos problemas, uno de *docking* de moléculas y otro de modelos de autoregresión vectorial, sobre nodos multicore+multiGPU.**

***Index Terms*—metaheurísticas, hiperheurísticas, paralelismo híbrido, paralelismo heterogéneo, docking de moléculas, modelos de autoregresión vectorial**

I. Introducción

Las metaheurísticas se utilizan para la aproximación de soluciones de problemas de gran dificultad computacional [1]–[3]. Hay variedad de métodos metaheurísticos, básicamente agrupados en dos clases: distribuidos o basados en poblaciones [4], [5], y de búsqueda local, y también métodos híbridos [6], [7] que combinan las características de diferentes metaheurísticas para una mejor adaptación al problema con el que se trabaja. Así, el *software* es híbrido al combinar distintas técnicas, y heterogéneo en el sentido de que cada uno de sus componentes tiene un coste computacional distinto.

Con la aparición de la computación paralela se han desarrollado versiones paralelas de las metaheurísticas, o nuevas metaheurísticas paralelas [8]. Los nodos de la mayoría de los sistemas computacionales actuales están formados por sistemas multinúcleo junto con coprocesadores, que son principalmente GPUs (Graphics Processing Unit)) y MICs (Many Integrated Core), por lo que se está trabajando en la adaptación y optimización de software para este tipo de sistemas híbridos, y, en particular, hay trabajos sobre metaheurísticas para GPU [9], [10]. Además, el *hardware* sobre el que se trabaja es híbrido y heterogéneo, con elementos computacionales con arquitecturas distintas, en distinto número y distinta velocidad, y está organizado en muchos casos de forma jerárquica (un *cluster* con nodos híbridos multicore+GPU, GPUs organizadas en *grids* y bloques...).

Esta situación propicia la línea de trabajo que se presenta: análisis de la combinación de las características híbridas, heterogéneas y jerárquicas de *software* y *hardware* para explotar el paralelismo y obtener beneficios en cuanto a reducir el tiempo de ejecución o mejorar las soluciones obtenidas.

Trabajamos con metaheurísticas híbridas desarrolladas a partir de un esquema parametrizado [11], que combina características de metaheurísticas distribuidas y de búsqueda local, y se ha aplicado a diversidad de problemas: p-hub, asignación de tareas a procesos, y modelado de ecuaciones simultáneas [11]; consumo de electricidad en explotación de pozos de agua [12]; determinación de constantes en ecuaciones cinéticas [13], etc. Además, al estar el esquema parametrizado, se puede utilizar para diseñar hiperheurísticas que buscan en el espacio de los parámetros que determinan las metaheurísticas [14].

El esquema parametrizado se puede ampliar a esquemas paralelos que incluyen parámetros de paralelismo dependiendo del sistema computacional destino (memoria compartida [15] o paso de mensajes [16]), con una paralelización unificada para las distintas metaheurísticas híbridas del esquema.

Analizamos la adaptación del esquema metaheurístico parametrizado a sistemas más complejos: nodos multicore+multiGPU, posiblemente con GPUs de distintas características. Y los problemas a los que se aplican las metaheurísticas tienen una característica que facilita el uso eficiente de la alta capacidad computacional de los coprocesadores: alto coste de cálculo del *fitness*, que puede delegarse a

Subvencionado por MINECO y fondos FEDER de la Comisión Europea, TIN2015-66972-C5-3-R y TIN2016-78799-P (AEI/FEDER, UE).

las GPUs.

Se utilizan dos problemas: *docking* de moléculas y modelado de autoregresión vectorial; y un tercer problema es la búsqueda por medio de hiperheurísticas (programadas como metaheurísticas) de metaheurísticas apropiadas para estos dos problemas básicos. El mayor coste computacional reside en el cálculo del *fitness*, y en las hiperheurísticas el cálculo del *fitness* se hace a su vez con la aplicación de metaheurísticas al problema básico.

El resto del trabajo está estructurado de la siguiente manera. La Sección II muestra el esquema metaheurístico parametrizado que se utiliza, y se comentan las características de las hiperheurísticas sobre este esquema. Las posibilidades de paralelización del esquema para multicore+multiGPU se comentan en la Sección III. Los dos problemas básicos usados como casos de prueba se describen en la Sección IV, y la Sección V muestra algunos resultados experimentales en la aplicación a los dos problemas básicos y a hiperheurísticas para el problema del *docking*. Las conclusiones y posibles trabajos futuros se resumen en la Sección VI.

II. Esquema Parametrizado de Metaheurísticas e Hiperheurísticas

Resumimos las ideas generales de los esquemas metaheurísticos parametrizados usados en este trabajo. Se pueden encontrar más detalles del esquema, de su aplicación a varios problemas de optimización y un estudio estadístico de la influencia de los parámetros en [11]. El esquema se muestra en el Algoritmo 1. Incluye Parámetros Metaheurísticos en cada una de las funciones básicas. No es un esquema general que incluya todas las posibles metaheurísticas (incontables) sino que depende de la implementación de las funciones básicas y de los parámetros que se incluyan en ellas. Instancias concretas de los parámetros pueden dar lugar a metaheurísticas básicas o a hibridaciones suyas. Por ejemplo, en [11] se implementan versiones de Algoritmos Genéticos, Búsqueda Dispersa y GRASP (Greedy Randomized Adaptive Search Procedure), y otras de las referencias mencionadas incluyen también Búsqueda Tabú. La mayoría de los parámetros son generales para el esquema, pero puede haber algunos propios del problema al que se aplican las metaheurísticas. Una descripción detallada se encuentra en las referencias previas.

Las hiperheurísticas que consideramos son a la vez metaheurísticas con el mismo esquema parametrizado, pero el significado de los parámetros es distinto, ya que depende del problema al que se aplican, que en este caso es el de búsqueda de metaheurísticas híbridas satisfactorias para el problema básico considerado. El esquema metaheurístico para implementar hiperheurísticas se muestra en el Algoritmo 2. Es igual al del Algoritmo 1 con la única diferencia los elementos con los que se trabaja (ahora son vectores de parámetros metaheurísticos) y el *fitness* asociado a cada elemento (metaheurística), que se obtiene con la aplicación de la metaheurística representada por el vector de parámetros metaheurísticos a uno o varios problemas de entrenamiento. Algunas formas de calcular el *fitness* se discuten en [14].

III. Esquemas Paralelos para multicore+multiGPU

Hemos desarrollado esquemas paralelos para memoria compartida [15] y paso de mensajes [16], y hemos analizado la utilización de esquemas parametrizados en la resolución de problemas particulares en nodos con coprocesadores. Por ejemplo, en [17] se analiza la aplicación de hiperheurísticas al problema de *docking* en un MIC. El esquema paralelo es el de memoria compartida, pero los MIC pueden ser considerados *manycore* más que multicore al ser el número de procesos mucho más elevado, y esto hace que la explotación del paralelismo a varios niveles (en la hiperheurística y las metaheurísticas donde ella busca) sea más adecuado para estos sistemas.

En el caso de la explotación de paralelismo de GPUs el esquema paralelo es distinto de los dos anteriores, pues la programación en GPUs sigue un modelo SIMD. El desarrollo de metaheurísticas para GPUs es un campo de investigación actual [9], [10], y hemos utilizado el esquema parametrizado para el problema del *docking* en nodos con GPUs, ya sean locales o virtualizadas [18], lo que acelera la computación y facilita encontrar soluciones mejores en menor tiempo.

En este trabajo analizamos por primera vez el diseño de metaheurísticas parametrizadas de forma general para explotar toda la capacidad computacional que proporcionan los nodos multicore+multiGPU. Para analizar el problema de forma general consideramos dos problemas, que además tienen alto coste computacional, lo que es necesario para que la explotación eficiente de los coprocesadores muestre todo su potencial.

Las metaheurísticas y las hiperheurísticas que trabajan sobre ellas siguen el mismo esquema parametrizado (algoritmos 1 y 2), con diferencias en la implementación de las funciones y los parámetros, dependiendo del problema subyacente a que se aplican. Así, la versión de memoria compartida es común a metaheurísticas e hyperheurísticas (Algoritmo 3), y consiste en la paralelización independiente de las funciones básicas del esquema, con inclusión de parámetros de paralelismo que determinan el número de hilos a usar en cada función. Hay funciones donde se tratan varios elementos dentro de un bucle, y se paraleliza el bucle estableciendo un determinado número de hilos. En las funciones de mejora se tratan varios elementos, en un bucle que se paraleliza con un cierto número de hilos, y para cada elemento se analiza su vecindad, lo que se hace con otro bucle con un número de hilos diferente en un segundo nivel de paralelismo. Como las hiperheurísticas llaman a metaheurísticas para el cálculo del *fitness*, llegamos a tener en algunos casos hasta cuatro niveles de paralelismo, y se podría determinar en número de hilos en cada nivel para la mayor reducción del tiempo de ejecución teniendo en cuenta las características del sistema computacional [17].

En sistemas con paralelismo híbrido multicore+multiGPU el código correrá en la CPU, que delegará a GPU partes con alto coste computacional. El paralelismo de GPU puede explotarse a distintos niveles de entre los que aparecen en el esquema de memoria compartida. Los datos con los que trabajar en GPU se

Algorithm 1 Esquema metaheurístico parametrizado

```
Initialize(S,ParamIni) //Generación del conjunto inicial, posiblemente con mejora de elementos
while (not EndCondition(S,ParamEnd)) do
  SS=Select(S,ParamSel) //Selección de elementos para combinación
  SS1=Combine(SS,ParamCom) //Combinación de pares de elementos
  SS2=Improve(S,SS1,ParamImp) //Mejora de algunos elementos, posiblemente con diversificación
  S=Include(S,SS1,SS2,ParamInc) //Actualización del conjunto de referencia
end while
```

Algorithm 2 Esquema metaheurístico parametrizado para la implementación de hiperheurísticas

```
Initialize(S,ParamIni) //Generación de un conjunto inicial de metaheurísticas
while (not EndCondition(S,ParamEnd)) do
  SS=Select(S,ParamSel) //Selección de metaheurísticas para combinación
  SS1=Combine(SS,ParamCom) //Combinación de pares de metaheurísticas para obtener nuevas metaheurísticas
  SS2=Improve(S,SS1,ParamImp) //Mejora y diversificación de metaheurísticas para obtener otras más apropiadas
  para el problema con que se trabaja
  S=Include(S,SS1,SS2,ParamInc) //Actualización del conjunto de metaheurísticas
end while
```

Algorithm 3 Versión de Memoria Compartida del esquema metaheurístico parametrizado

```
Initialize(S,ParamIni,HilosIni) //Número de hilos en la generación, y número de hilos sobre los elementos a
mejorar y un segundo nivel para hilos que analizan la vecindad
while (not EndCondition(S,ParamEnd)) do
  SS=Select(S,ParamSel,HilosSel) //Número de hilos en bucle de selección
  SS1=Combine(SS,ParamCom,HilosCom) //Número de hilos en bucle de combinación
  SS2=Improve(S,SS1,ParamImp,HilosImp) //Número de hilos sobre los elementos a mejorar o diversificar, y un
  segundo nivel para hilos que analizan la vecindad
  S=Include(S,SS1,SS2,ParamInc,HilosInc) //Número de hilos en bucles para tratar los elementos a incluir
end while
```

transfieren de CPU a GPU, y los resultados a la inversa, y las transferencias CPU-GPU pueden ser costosas, por lo que habrá que realizar implementaciones donde se solapen computación y transferencias, y además el volumen de trabajo a realizar en GPU en cada llamada debe ser grande. Algunos niveles a los que pueden trabajar las GPUs en el esquema del Algoritmo 3 son:

- El mayor nivel corresponde al esquema de islas, con una isla por GPU en un sistema multicore+multiGPU, dividiendo los conjuntos en subconjuntos y asignando cada subconjunto a una GPU. El trabajo que se delega a las GPU es de propsito general y no sigue el modelo SIMD, que es el más adecuado para obtener buenas prestaciones en GPU.
- Si bajamos a nivel de paralelización dentro de cada función básica, el esquema se ejecuta en CPU y las funciones trabajan sobre elementos por medio de bucles, con cada iteración del bucle realizada por un hilo. Cada hilo puede tener asociada una GPU a la que delega el tratamiento del elemento, que se transfiere a la GPU, que devuelve a CPU el resultado. Las funciones básicas tienen costes distintos, por lo que habrá que determinar para cada una, dependiendo de cómo esté implementada, si es conveniente delegar el trabajo a GPU o realizarlo en CPU. Además, algunas funciones conllevan comparaciones y bifurcaciones en el análisis de la vecindad, por lo que, de nuevo, no tenemos un modelo SIMD.
- En un siguiente nivel las GPUs se utilizarían para el cálculo del *fitness*. Como en algunas funciones hay que calcular el *fitness* de los objetos obtenidos (en la generación inicial, al combinar, al analizar vecindades en las funciones de mejora), se pueden agrupar los cálculos de los *fitness* de todos los elementos, que se realizarían de nuevo en un bucle, con cada hilo delegando el cálculo de un elemento a la GPU que tiene asociada. De esta forma el trabajo que realizan las GPUs puede seguir el modelo SIMD si el cálculo del *fitness* se realiza a través de funciones matriciales, como es el caso de los problemas con los que experimentamos en este trabajo.
- En un último nivel serían todas las GPUs las que colaboraran en el cálculo del *fitness* de cada individuo. Podría ser una buena opción cuando este cálculo tenga un alto coste computacional, lo que puede ocurrir con las hiperheurísticas, donde el cálculo conlleva la aplicación completa de una metaheurística a una instancia del problema básico.

La mejor opción puede ser una combinación de las dos versiones intermedias, determinando el nivel al que trabajan las

GPU dependiendo del coste de la rutina básica y del número de elementos que se trate en cada nivel. El Algoritmo 4 muestra los puntos en los que se pueden usar las GPUs. Para cada rutina se indica con CPU o GPU el componente donde se ejecuta. Las rutinas sobre CPU usan paralelismo OpenMP. En las rutinas de mejora hay que determinar en cuál de los dos niveles de paralelismo es preferible utilizar las GPUs.

Algorithm 4 Asignación de trabajo a CPU y GPU en un esquema parametrizado de metaheurísticas para multicore+multiGPU

```
InitializeCPU(Sini,ParamIni)
ComputefitnessGPU(Sini,ParamIni)
ImproveGPU(Sini,Sref,ParamImpIni) //2 posibles niveles
while not EndConditionCPU(Sref,ParamEndCon) do
  SelectCPU(Sref,Ssel,ParamSel)
  CombineCPU(Ssel,Scom,ParamCom)
  ComputefitnessGPU(Scom,ParamCom)
  DiversifyCPU(Sref,Scom,Sdiv,ParamDiv)
  ComputefitnessGPU(Sdiv,ParamDiv)
  ImproveGPU(Scom,Sdiv,ParamImp) //2 posibles niveles
  IncludeCPU(Scom,Sdiv,Sref,ParamInc)
end while
```

IV. Problemas Básicos

Se comentan las características generales de los dos problemas de optimización que se han utilizado en los experimentos.

IV-A. Docking de Moléculas

Se dispone de una molécula de gran dimensión (receptor) y otra menor (ligando), y se quiere encontrar la mejor forma en que el ligando se acopla con el receptor, de acuerdo con una función de *scoring*. En la bibliografía se encuentran varias funciones que se pueden usar [19], y en ellas se mide el acoplamiento considerando cada átomo en el receptor y el ligando, con lo que son funciones de alto coste y con un esquema de cómputo regular, que es apropiado para ser utilizado en GPU. Cada posición del ligando viene determinada por tres coordenadas para indicar la posición en el espacio de un átomo de referencia, y otras tres para indicar los giros con respecto a esta referencia; además, el ligando puede tener puntos donde se puede flexionar. Así, cada elemento viene determinado por las variables que indican la posición del ligando. Además, hay varias regiones independientes (*spots*) en la superficie del receptor donde el ligando puede acoplarse, por lo que la búsqueda en los *spots* se realiza de forma idependiente, lo que significa que las metaheurísticas realizan la búsqueda con poblaciones independientes para cada *spot*. Teniendo en cuenta que el número de *spots* puede ser de unos cientos, y el número de átomos de receptor y ligando de unos miles y unos cientos, el coste computacional de la aplicación de metaheurísticas a este problema es alto, y es conveniente la utilización de sistemas de altas prestaciones del tipo multicore+multiGPU, posiblemente con GPUs virtualizadas para disponer de mayor potencia computacional [18].

IV-B. Modelos de Autoregresión Vectoriales

Consideramos d parámetros de los que se han tomado valores en t instantes de tiempo. Los valores para el instante k, $1 \leq k \leq t$, se almacenan en un vector $y^{(k)} = \left(y_1^{(k)}, \ldots, y_d^{(k)}\right) \in R^{1\times d}$. Y los t vectores de datos se pueden organizar en una matriz $Y \in R^{t\times d}$:

$$\begin{pmatrix} y_1^{(1)} & \cdots & y_d^{(1)} \\ \vdots & \ddots & \vdots \\ y_1^{(t)} & \cdots & y_d^{(t)} \end{pmatrix} \quad (1)$$

Consideramos también dependencias temporales con i instantes de tiempo anteriores. Puede haber variables externas al modelo, vector z, con e variables. La dependencia de vectores de datos de un instante j en función de los vectores de datos de instantes anteriores (hasta i para las variables internas y k para las externas) se puede representar como

$$y^{(j)} \approx y^{(j-1)}A_1 + y^{(j-2)}A_2 + \ldots + y^{(j-i)}A_i + z^{(j-1)}B_1 + z^{(j-2)}B_2 + \ldots + z^{(j-k)}B_k + a \quad (2)$$

donde A_l, $1 \leq l \leq i$, son matrices de dimensión $d \times d$ que representan la dependencia de los datos con los valores de l instantes anteriores, B_l, $1 \leq l \leq k$, son matrices de tamaño $e \times d$, y a es un vector de términos independientes.

Cada elemento del espacio de búsqueda de una metaheurística viene dado por los valores de las matrices A, B y a, y el *fitness* se obtiene con multiplicaciones matriciales y la norma de la diferencia entre los valores de la serie temporal (ecuación 1) y los que se obtienen en cada instante de tiempo utilizando los instantes anteriores y dichas matrices (ecuación 2) [20]. Así, también ahora las operaciones son apropiadas para el modelo SIMD, pero las dimensiones (d, e, i, k) son bastante menores que en el problema anterior.

V. Resultados Experimentales

Se resumen los resultados de los experimentos realizados en nodos computacionales con CPUs y GPUs de distintos tipos y uno de ellos con varias GPUS:

- **marte**: contiene un hexa-core CPU AMD Phenom II X6 1075T a 3.00 GHz, con arquitectura x86-64, 16 GB de RAM, cachés privadas L1 y L2 de 64 KB y 512 KB, y una caché L3 de 6 MB compartida por todos los núcleos. Incluye una GPU NVidia GeForce GTX 480 (Fermi) con 1536 MB de memoria global y 480 núcleos CUDA (15 Streaming Multiprocessors y 32 Streaming Processors por SM).
- **saturno**: tiene 4 CPU Intel Xeon E7530 (hexa-core) a 1.87 GHz, con arquitectura x86-64, 32 GB de RAM, cachés privadas L1 y L2 de 32 KB y 256 KB por núcleo, y una caché L3 de 12 MB compartida por los núcleos de cada socket. Incluye una GPU NVidia Tesla K20c (Kepler) con 4800 MB de memoria global y 2496 núcleos CUDA (13 Streaming Multiprocessors y 192 Streaming Processors por SM).

- **jupiter**: con 2 CPU Intel Xeon E5-2620 (hexa-core) a 2.00 GHz, con arquitectura x86-64, 32 GB de memoria RAM, cachés privadas L1 y L2 de 32 KB y 256 KB por núcleo, y una caché L3 de 15 MB compartida por los núcleos en cada socket. Incluye 6 GPUs: 2 NVidia Tesla C2075 (Fermi) con 5375 MB de memoria global y 448 núcleos CUDA (14 Streaming Multiprocessors y 32 Streaming Processors per SM), y 4 GPUs en dos placas, cada una es una con 2 NVidia GeForce GTX 590 (Fermi) con 1536 MB de memoria global y 512 núcleos CUDA (16 Streaming Multiprocessors y 32 Streaming Processors por SM).
- **venus**: con 2 CPU Intel Xeon E5-2620 (hexa-core) a 2.40 GHz, arquitectura x86-64, 64 GB de memoria RAM, cachés privadas L1 y L2 de 32 KB y 256 KB por núcleo, y una caché L3 de 15 MB compartida por todos los núcleos de un socket. Incluye una GPU NVidia GeForce GT 640 (Kepler) con 1024 MB de Global Memory y 384 núcleos CUDA (2 Streaming Multiprocessors y 192 Streaming Processors por SM).

La Tabla I compara los tiempos obtenidos en **jupiter** con paralelismo GPU (sólo se usa una Tesla C2075) y con CPU con un núcleo, para seis metaheurísticas híbridas aplicadas al problema de *docking* de moléculas con el par receptor-ligando COMT (se pueden consultar los detalles en [21]). Se alcanza un speed-up de GPU con respecto a CPU de alrededor de 43. La ventaja de usar GPUs es clara, incluso en este caso en que la GPU se usa sólo en el cálculo de la función de *scoring*.

Tabla I
Tiempo de ejecución (en segundos) de versiones CPU y GPU, y speed-up de GPU respecto a CPU, para diferentes metaheurísticas aplicadas al par receptor-ligando COMT, en **jupiter**

Metaheuristic	CPU	GPU	CPU/GPU
M1	140.48	10.42	39.35
M2	193.67	13.81	43.55
M3	1,911.52	9.62	54.16
M4	209.56	9.59	34.43
M5	262.65	8.55	35.51
M6	1,379.93	10.39	53.89

El par con el que se ha experimentado para los resultados de la Tabla I es pequeño pero los tiempos de ejecución pueden ser grandes dependiendo de la metaheurística. En el caso de las hiperheurísticas, el tiempo crece enormemente dado que se busca en un espacio de metaheurísticas. En una hiperheurística se puede tener hasta cuatro niveles de paralelismo (dos en la hiperheurística y hasta dos en la metaheurística), por lo que es necesario experimentar con las distintas combinaciones del número de hilos en los cuatro niveles para obtener la mejor combinación. Para simplificar la experimentación dado que el tiempo de cada experimento es bastante largo, hemos considerado los tres niveles de paralelismo de mayor nivel, con lo que las metaheurísticas usan sólo paralelismo de un nivel. La Figura 1 compara el tiempo de ejecución (segundos) con la versión de memoria compartida para una hiperheurística particular, con diferentes combinaciones de número de hilos

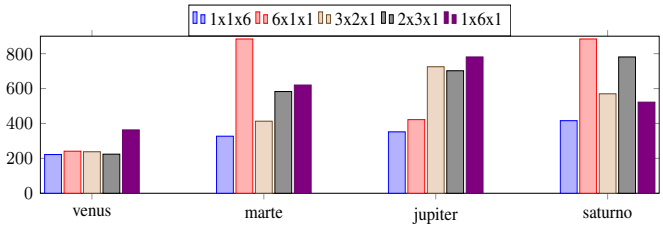


Figura 1. Tiempo de ejecución (segundos) con la versión de memoria compartida para una hiperheurística, con diferentes combinaciones de número de hilos en tres niveles de paralelismo, con un total de seis hilos y en los cuatro nodos considerados.

Tabla II
Comparación del tiempo de ejecución (en segundos) de tres hiperheurísticas con paralelismo de memoria compartida en CPU y con paralelismo GPU, en los cuatro nodos considerados.

	H1	H2	H3
	marte		
multicore	13035	4541	23957
multicore+GPU	903	335	1348
CPU/GPU	14.44	13.56	17.77
	venus		
multicore	4021	1423	6346
multicore+GPU	3197	1857	5065
CPU/GPU	1.26	0.77	9.67
	jupiter		
multicore	8045	3509	14115
multicore+GPU	990	514	1459
CPU/GPU	8.13	6.83	9.67
	saturno		
multicore	6150	1595	8582
multicore+GPU	563	200	823
CPU/GPU	10.92	7.98	10.43

en tres niveles de paralelismo, con un total de seis hilos y en los cuatro nodos considerados. Se obtienen resultados similares con otras hiperheurísticas. La mejor combinación de hilos es $1 \times 1 \times 6$ en todos los nodos, con algunas diferencias en la comparación de las distintas combinaciones dependiendo del nodo, lo que es normal dada la distinta capacidad computacional de los nodos y que hay aleatoriedad en las ejecuciones. Para evitar la aleatoriedad se deberían realizar varias ejecuciones por cada combinación, pero, dado el alto coste computacional, esto conllevaría unos tiempos de experimentación excesivos.

La Tabla II compara el tiempo de ejecución de tres hiperheurísticas cuando se explota paralelismo de memoria compartida con un número de hilos igual al número de cores en el nodo y cuando se usa GPU para el cálculo de las funciones de *scoring*. Se ha usado de nuevo el par COMT. Se comprueba que el tiempo de ejecución es mucho mayor que el de una única metaheurística, y el tiempo de ejecución con la explotación del paralelismo híbrido multicore+GPU es en general mucho menor que con paralelismo de memoria compartida. Sólo en el caso de **venus**, que tiene la CPU más rápida y la GPU más lenta, son los tiempos comparables.

El uso de GPU también puede dar resultados satisfactorios para el problema de modelos de autoregresión vectorial, pero

en este caso el coste computacional es mucho menor que en el *docking* de moléculas, por lo que el uso de GPU resulta ventajoso sólo para problemas muy grandes que seguramente no se corresponden con casos de aplicaciones reales. Por ejemplo, en **jupiter**, cuando se utiliza una única GPU en el nivel más bajo (cálculo del *fitness*, que en este caso es una multiplicación de matrices y el cálculo de la norma), con valores de los parámetros $t = 200$, $d = 4$, $e = 2$, $i = 3$ y $k = 2$ el speed-up de GPU respecto a CPU es 0.83, con lo que es preferible no usar GPU. Pero para problemas mayores el speed-up aumenta: para $d = 5$, $e = 3$, $i = 3$ y $k = 2$ es 0.92, y para $d = 5$, $e = 3$, $t_y = 4$ y $t_z = 2$ llega a 1.25.

El uso de más GPUs reduce el tiempo de ejecución sólo para tamaños muy grandes, por lo que para este problema es necesario optimizar más la utilización de GPUs. Un ejemplo es la asignación dinámica de trabajo. En una ejecución en **jupiter** con cuatro GPUs (dos de cada tipo) y tamaños mayores que en el experimento anterior ($t = 500$), el tiempo de ejecución fue 1854 segundos cuando se asignó el cálculo del *fitness* a las GPUs con el mismo número de elementos para cada GPU. Al asignar de forma estática el doble de elementos a las GPUs más rápidas (en teoría son el doble de rápidas) el tiempo subió a 2249 segundos. Y el menor tiempo se obtuvo con asignación dinámica, con 1598 segundos.

VI. Conclusiones y Trabajos Futuros

En este trabajo se discute la utilización de paralelismo híbrido multicore+multiGPU en la aplicación de metaheurísticas híbridas basadas en un esquema parametrizado de metaheurísticas. En el esquema hay paralelismo en distintos niveles, y se analiza en qué niveles sería conveniente la utilización de GPUs. Además, sobre el esquema se pueden desarrollar hiperheurísticas que se implementan como metaheurísticas con ese esquema y que buscan en el espacio de metaheurísticas para un problema básico implementadas con el mismo esquema. Como el esquema paralelo parametrizado tiene dos niveles de paralelismo, la implementación de hiperheurísticas da lugar a un máximo de cuatro niveles, y se debe analizar la mejor combinación de hilos en cada nivel y en qué partes del esquema paralelo utilizar GPU.

Se han considerado dos problemas básicos para analizar el esquema: *docking* de moléculas y modelos de autoregresión vectorial. Los dos problemas tienen alto coste computacional y el cálculo del *fitness* se hace con computación matricial que sigue un esquema SIMD, por lo que se puede explotar paralelismo GPU. Para el primer problema, dado el tamaño de las moléculas con las que se realiza el *docking*, el tiempo de ejecución es muy elevado, y se alcanza una gran reducción del tiempo con respecto a CPU cuando se utiliza paralelismo híbrido multicore+GPU. En el otro problema, el uso de GPU es ventajoso sólo para problemas grandes, que no se corresponden con aplicaciones en uso.

El mismo esquema puede ser utilizado en otros problemas, y estamos centrando nuestra investigación en el análisis de cómo determinar en función de los costes del problema de optimización la forma óptima de utilizar el esquema: número de hilos y forma de uso de las GPUs en cada nivel.

Referencias

[1] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*. Springer, 2002.

[2] J. Hromkovič, *Algorithmics for Hard Problems*. Springer, second ed., 2003.

[3] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization*. Springer, 2005.

[4] F. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*. Kluwer, 2003.

[5] E.-G. Talbi, *Metaheuristics - From Design to Implementation*. New York: Wiley, 2009.

[6] G. R. Raidl, "A unified view on hybrid metaheuristics," in *Hybrid Metaheuristics, Third International Workshop, LNCS*, vol. 4030, pp. 1–12, October 2006.

[7] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Appl. Soft Comput.*, vol. 11, no. 6, pp. 4135–4151, 2011.

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

[9] E.-G. Talbi and G. Hasle, "Metaheuristics on GPUs," *J. Parallel Distrib. Comput.*, vol. 73, no. 1, pp. 1–3, 2013.

[10] M. Essaid, L. Idoumghar, J. Lepagnot, and M. Brévilliers, "GPU parallelization strategies for metaheuristics: a survey," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 0, no. 0, pp. 1–26, 2018.

[11] F. Almeida, D. Giménez, J.-J. López-Espín, and M. Pérez-Pérez, "Parameterised schemes of metaheuristics: basic ideas and applications with Genetic algorithms, Scatter Search and GRASP," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 43, no. 3, pp. 570–586, 2013.

[12] L.-G. Cutillas-Lozano, J.-M. Cutillas-Lozano, and D. Giménez, "Modeling shared-memory metaheuristic schemes for electricity consumption," in *Distributed Computing and Artificial Intelligence - 9th International Conference*, pp. 33–40, 2012.

[13] J. Cutillas-Lozano and D. Giménez, "Determination of the kinetic constants of a chemical reaction in heterogeneous phase using parameterized metaheuristics," in *Proceedings of the International Conference on Computational Science*, pp. 787–796, 2013.

[14] J. Cutillas-Lozano, D. Giménez, and F. Almeida, "Hyperheuristics based on parametrized metaheuristic schemes," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 361–368, 2015.

[15] F. Almeida, D. Giménez, and J.-J. López-Espín, "A parameterized shared-memory scheme for parameterized metaheuristics," *The Journal of Supercomputing*, vol. 58, no. 3, pp. 292–301, 2011.

[16] J. Cutillas-Lozano and D. Giménez, "Optimizing a parameterized message-passing metaheuristic scheme on a heterogeneous cluster," *Soft Comput.*, vol. 21, no. 19, pp. 5557–5572, 2017.

[17] J. M. Cecilia, J. Cutillas-Lozano, D. Giménez, and B. Imbernón, "Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem," *The Journal of Supercomputing*, vol. 74, no. 5, pp. 1803–1814, 2018.

[18] B. Imbernón, J. Prades, D. Giménez, J. M. Cecilia, and F. Silla, "Enhancing large-scale docking simulation on heterogeneous systems: An MPI vs rcuda study," *Future Generation Comp. Syst.*, vol. 79, pp. 26–37, 2018.

[19] G. Schneider, "Virtual screening and fast automated docking methods," *Drug Discovery Today*, vol. 7, pp. 64–70, Jan. 2002.

[20] A. L. Castaño, J. Cuenca, J.-M. Cutillas-Lozano, D. Giménez, J. J. López-Espín, and A. Pérez-Bernabeu, "Parallelism on hybrid metaheuristics for vector autoregression models," in *International Conference on High Performance Computing & Simulation*, August 2018.

[21] B. Imbernón, J. M. Cecilia, and D. Giménez, "Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems," in *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, pp. 50–58, 2016.

Optimización evolutiva multiobjetivo distribuida mediante aplicación selectiva de operadores

Pablo García-Sánchez
Departamento de Ingeniería Informática
Universidad de Cádiz
Puerto Real, España
pablo.garciasanchez@uca.es

Julio Ortega, Jesús González, Pedro A. Castillo,
Juan Julián Merelo, Antonio M. Mora y Antonio Fernández-Ares
Departamento de Arquitectura y Tecnología de Computadores
Universidad de Granada
Granada, España
julio@ugr.es

***Resumen*—Este trabajo presenta un nuevo enfoque, llamado *aplicación selectiva de operadores*, para hacer frente a problemas no separables en el entorno de los algoritmos co-evolutivos multiobjetivo: en lugar de compartir secciones de individuos, cada procesador aplica los operadores de variación a un subconjunto específico de todo el individuo. Esto evita pasos adicionales para recomponer los individuos completos de otras islas antes de ser evaluados.**

En este trabajo se pretende demostrar que la decisión automática del tamaño de la sección solapada, es capaz de obtener mejores resultados que la utilización del mismo tamaño independientemente del número de islas. Para ello se ha comparado con otras técnicas evolutivas colaborativas considerando diferentes números de islas y problemas. El análisis de los resultados experimentales obtenidos, utilizando diferentes métricas, muestra que nuestro enfoque puede proporcionar mejoras estadísticamente significativas con respecto al algoritmo base en problemas multiobjetivo de alta dimensionalidad.

***Index Terms*—Algoritmos multiobjetivo, NSGA-II, modelo de islas, algoritmos evolutivos distribuidos**

I. Introducción

Los problemas de optimización multiobjetivo (MOP por sus siglas en inglés) son aquellos en los que varios objetivos tienen que ser optimizados a la vez [1]. Resolver un MOP implica optimizar una función compuesta por varias funciones de coste independientes, una por objetivo. En estos problemas el objetivo es obtener un conjunto de soluciones que sean mejores que el resto, considerando todos los objetivos; este conjunto se conoce como el Frente de Pareto (FP). Las soluciones en este conjunto son *no dominadas*, lo que significa que no hay otra solución que sea igual o mejor para todos los objetivos.

Este aspecto de la búsqueda de soluciones no dominadas, así como, en muchos casos, el tamaño del propio espacio de búsqueda, implica una alta demanda de tiempo computacional, lo que lleva a la propuesta de métodos paralelos y distribuidos para resolverlos [2], [3], uno de los cuales son los algoritmos evolutivos (EAs) [4].

Trabajo financiado por los proyectos SPIP2017-02116 (Dirección General de Tráfico), EphemeCH TIN2014-56494-C4-3-P, TIN2015-67020-P, DeepBio TIN2017-85727-C4-2-P y TEC2015-68752 (Ministerio de Economía y Competitividad y Fondos FEDER) y PR2018-056 (Programa de Fomento e Impulso de la Investigación y de la Transferencia de la Universidad de Cádiz 2018-2019).

Los EAs se han extendido a la optimización multiobjetivo a través de un buen número de Algoritmos Evolutivos Multiobjetivo (MOEAs, por sus siglas en inglés) [5]. Se han utilizado diferentes enfoques para paralelizar los EAs, ya que cada individuo puede ser considerado como una unidad independiente [6], [7]. Los métodos clásicos, como los EAs paralelos globales (Maestro-Esclavo), o los algoritmos espacialmente estructurados (Modelo de isla o EAs celulares) se han aplicado con éxito en el pasado [8]. Sin embargo, en el caso de los MOEAs, estos enfoques [2] necesitan ocuparse de todo el conjunto de soluciones, el FP. Esto implica el uso de diferentes mecanismos de distribución y compartición, ya que existe un equilibrio entre la mejora obtenida a partir de la paralelización y la necesidad de recombinar globalmente los resultados para identificar con precisión el FP [9].

Las MOPs del mundo real normalmente requieren un alto número de variables de decisión, lo que significa que los MOEAs necesitan tratar con individuos grandes y gastar un tiempo significativo adicional para el cruce, mutación y migración. Diferentes autores han propuesto métodos para dividir el espacio de decisión (el cromosoma) para mejorar el rendimiento y la calidad de las soluciones. En este aspecto, el modelo de co-evolución es un modelo distribuido por dimensiones en el que un problema de alta dimensionalidad se divide en otros de dimensiones más bajas [7], que evolucionan por separado. Un ejemplo de aplicación de esta técnica fue descrito en [10]. El método presentado en ese trabajo involucra a diferentes trabajadores que evolucionan subpoblaciones creadas y recombinadas por un proceso maestro, el cual realiza diferentes alternativas de recombinación de las partes devueltas por los procesos de los trabajadores.

Un enfoque similar utilizado para resolver este tipo de problemas es el de la aplicación de la Aplicación Selectiva de Operadores (ASO) presentado en este documento. En este caso, cada isla se ocupa de la totalidad del cromosoma, pero sólo modifica un fragmento en la fase de cruce y mutación en función del número de islas, utilizando todo el cromosoma para el cálculo del fitness. Esto permite hacer frente a problemas que no se pueden descomponer. En nuestro trabajo anterior [11] utilizamos este método de manera preliminar. En ese trabajo demostramos que la aplicación de los operadores de variación sólo sobre secciones específicas de todo el cromosoma mejora

la calidad de las soluciones en el mismo tiempo de cálculo para un algoritmo multi-objetivo basado en islas. Además, en lugar de hacer que cada isla se centre en un subconjunto disjunto del cromosoma, el uso de secciones solapadas (compartidas) del cromosoma puede mejorar la calidad de las soluciones cuando se aumenta el número de islas.

Esto nos motiva a continuar esta línea de investigación utilizando un entorno más adecuado: un cluster real con hasta 128 nodos y una parametrización más completa. Además de comparar los métodos anteriores en esta nueva configuración experimental, en este trabajo proponemos un nuevo método que automáticamente establece el tamaño del solapamiento. en función del número de islas disponibles, comparándolo con versiones anteriores.

El resto del documento está organizado de la siguiente manera: después del estado del arte en MOEAs distribuidos y co-evolutivos, la metodología utilizada y los algoritmos comparados se describen en la Sección III. Después se presentan los resultados de los experimentos (Sección V), Finalmente, se discuten las conclusiones y el trabajo futuro [1].

II. Estado del arte

Desde principios de la década de los 2000, los EAs distribuidos y paralelos se han utilizado principalmente para aprovechar sistemas como clusters o grids [12]. Pero en el caso de los MOEAs, la distribución y paralelización es más difícil que en los EAs con un solo objetivo. Esto se debe a que en diferentes pasos del algoritmo el conjunto completo de soluciones dependientes, el FP, debe ser gestionado como un todo, dedicando tiempo a reunir a todos los individuos de los diferentes procesadores o islas.

Para resolver este problema, algunos autores han propuesto el uso de enfoques Maestro-Esclavo. Por ejemplo, [13] comparó diferentes enfoques maestro-esclavo: síncrono generacional, asíncrono generacional y asíncrono estacionario, siendo esta última la opción más prometedora.

También se ha explorado otro tipo de enfoques. El trabajo de Deb et al. [14] fue uno de los primeros enfoques para MOEAs distribuidos (dMOEAs). En ese trabajo, el dominio de las soluciones se divide en las islas mediante una transformación de coordenadas. En ese trabajo, los autores concluyeron que dividir el espacio de búsqueda es una buena idea, aunque lograr esto no es trivial. La división del espacio de búsqueda ha sido explorada por otros investigadores, por ejemplo, dividiendo la población en élites y subpoblaciones de búsqueda [15], o separándola en procesadores por objetivo [16]. Otros autores, como [17] utilizan la migración para aceptar individuos basados en la diversidad, y emigran desde áreas no superpobladas.

Abordar este tipo de problemas usando co-evolución cooperativa también ha sido estudiado en varios trabajos con enfoques más cercanos al que aquí se presenta. El enfoque de centrarse en una porción del cromosoma, como en nuestro método de solapamiento, fue utilizado en primer lugar en el trabajo de Dorronsoro et al. [18], obteniendo un rendimiento superlinear en varios casos. Recientemente, este enfoque también ha sido probado utilizando un algoritmo de optimización de enjambre de partículas (PSO) [19], obteniendo también mejoras significativas en la velocidad y calidad de la solución.

El enfoque descrito por Dorronsoro et al. también ha sido utilizado por Kimovski et al. [10], pero implementando un método maestro-esclavo que divide la población en varios procesadores. Como en trabajos anteriores, cada nodo ejecuta un MOEA paralelo que sólo afecta a una porción de los individuos y el proceso maestro recibe todas las subpoblaciones a combinar cada cierto número de generaciones. Se utilizaron hasta 8 procesadores y se compararon varias alternativas de combinación. La principal diferencia de nuestro trabajo con respecto a los trabajos anteriores es que nuestro enfoque no difunde todas las soluciones a todas las islas para la recombinación, sino sólo una solución a una isla aleatoria, necesitando menos tiempo de comunicación. Además, los enfoques de Dorronsoro o Kimovsky limitaron el número máximo de islas a 8, mientras que en este documento hemos utilizado hasta 128 islas.

En nuestro trabajo anterior [11] utilizamos algunas de las ideas mencionadas anteriormente para comparar dos dMOEAs diferentes. El primero dividió el cromosoma en P secciones, siendo P el número de islas. Cada isla p sólo realizó la mutación y el cruce en esa parte (p_{th}) del cromosoma (la aplicación selectiva de operadores), mientras que el *fitness* se calculó usando el individuo entero. Después de un cierto número de generaciones, los individuos fueron migrados al azar a otras islas. Las métricas de rendimiento se calcularon al final de la ejecución. El segundo método, la aplicación selectiva de operadores con islas solapadas, usaba las secciones p_{th-1} y p_{th+1} de cada cromosoma, además de la parte p_{th}. Usando la misma cantidad de tiempo, ambos métodos obtuvieron mejores resultados que un algoritmo de *baseline* que se ocupaba de todo el cromosoma en cada isla para el cruce y la mutación. Descubrimos que el rendimiento utilizando uno u otro método depende del número de secciones de los individuos y del número de islas utilizadas. Esto nos motivó a encontrar un nuevo método automático para seleccionar este número de secciones del cromosoma a utilizar, dependiendo del número de islas. Además, se realizaron experimentos previos en un modelo de isla síncrona con un único procesador y con un número limitado de islas (8, 32 y 128). En este trabajo se han utilizado 8, 16, 32, 64 y 128 islas, y en esta ocasión los experimentos se han realizado en un cluster paralelo. Por lo tanto, al mismo tiempo, estamos proponiendo un nuevo método para dividir el espacio de búsqueda individual según el número de subpoblaciones que evolucionan, y también validando el enfoque anterior.

III. Metodología

El objetivo de esta sección es explicar la metodología que hemos seguido para comparar las diferentes versiones de la selección de secciones a modificar en cada isla.

[1] Nota: una versión extendida de este artículo está siendo revisada en la revista *Applied Soft-Computing*

En este trabajo analizamos varios algoritmos basados en ASO que hemos implementado con respecto un algoritmo *baseline* multiobjetivo paralelo. Los métodos propuestos, utilizando diferentes esquemas de solapamiento, se basan en NSGA-II, como casi todos los trabajos discutidos anteriormente [13]–[18]. Por lo tanto, hemos utilizado un algoritmo NSGA-II básico distribuido sin solapar secciones como *baseline* (B).

Este algoritmo básico distribuye la población entre P islas; después de un número fijo de generaciones, un individuo en una isla dada es migrado a otra isla al azar, evitando así sincronizar el FP global cada cierto número de generaciones como lo hacen otros métodos descritos en la Sección 2. Al final de cada ejecución, los FPs de todas las islas se agregan en uno nuevo y se evalúan las medidas de calidad.

Se pueden idear diferentes alternativas de ASO para evolucionar las subpoblaciones de acuerdo con el espacio de decisión a explorar por cada isla. Al igual que en la *baseline* (B), un individuo es migrado a otra isla al azar después de un número fijo de generaciones. En la nueva isla, este individuo será considerado como uno más en la isla, cruzado y mutado de la misma manera, dependiendo del identificador de la nueva isla (de 1 a P). Nótese que, a diferencia de otros trabajos como los descritos por Talbi et al. [12], todas las islas tratan con cromosomas completos para el cálculo del fitness, por lo que nuestro enfoque puede tratar con problemas separables y no separables.

Concretamente hemos comparado las siguientes versiones:

- **ASO con islas disjuntas (D)** En este enfoque, cada individuo del tamaño L se divide en trozos de P del tamaño L/P. Cada isla p sólo realiza crossover y mutación en la parte p_{th} de los individuos.
- **ASO con secciones solapadas (S)** Este enfoque es similar al anterior, pero cada isla también utiliza los trozos de $p+1$ y $p-1$ (usando el módulo) del individuo para el cruce y la migración. Por lo tanto, existe algún tipo de solapamiento de las partes cruzadas y mutadas entre las islas.
- **ASO automático con secciones solapadas (A)** Como en el método anterior, las secciones a tratar por los operadores se solapan, pero en lugar de usar una sección extra a cada lado de la sección p ($p+1$ y $p-1$), usa fragmentos c a cada lado ($p+c$ y $p-c$), siendo c un valor que depende del número de islas.

Como primera aproximación para calcular automáticamente este valor, se han utilizado los resultados mostrados en [11] como base para obtener c. En ese trabajo, el método de solapamiento (cuando $c = 1$) obtuvo mejores resultados si el número de islas era mayor de 8. Por el contrario, cuando el número de islas es pequeño, no es necesario solaparlas. Por lo tanto, hemos usado este conocimiento para proponer la fórmula $c = round(0{,}2 * P - 1)/2$ para calcular las secciones extra a solapar. Por lo tanto, cuando $P = 8$ entonces $c = 0$ (equivalente a D), cuando $P = 16$ entonces $c = 1$ (equivalente a S), y así sucesivamente. La figura 1 explica el método A, asumiendo que $c = 2$, por ejemplo.

IV. Experimentos

En esta subsección se describen los indicadores de calidad utilizados y los experimentos. Los indicadores de calidad elegidos son:

- Hipervolumen (HV): mide el área formada por todas las soluciones no dominadas encontradas con respecto a un punto de referencia. Los valores más altos implican una mejor calidad del FP.
- Distancia Generacional Invertida (IGD): calcula la distancia del conjunto de soluciones obtenidas al FP óptimo. En esta métrica, cuanto más bajo, mejor.
- Spread (S): Mide la dispersión entre soluciones, teniendo en cuenta la distancia euclídea entre soluciones consecutivas. Como en la métrica anterior, cuanto menor sea el valor, mejor, ya que implica soluciones distribuidas a lo largo de todo el FP.

Hemos elegido estas métricas no sólo porque se han utilizado ampliamente, especialmente en algunos de los documentos presentados en la Sección II, como [13], [15], [17], [18], sino también porque cubren diferentes criterios de calidad. La formulación matemática de estas métricas se puede encontrar en [18].

El tamaño del cromosoma (L) es de 2048. También se ha comparado un número diferente de islas (P): 8, 16, 32, 64 y 128. Este número máximo de islas también ha sido utilizado en un trabajo anterior en la literatura [17]. El cruce y la mutación elegidos, SBX y polinomial, también han sido utilizados previamente por otros autores en [13].

El benchmark ZDT [20] ha sido elegido debido a que es el más utilizado en esta área [13]–[15], [17]. Este benchmark incluye varias funciones, con diferentes características que son representativas de los problemas de optimización del mundo real. La formulación matemática de cada función está disponible en [20].

El criterio utilizado para terminar un experimento ha sido el tiempo de ejecución: 100 segundos por ejecución. Según Alba y Luque [21], otros criterios de parada como el número de evaluaciones necesarias para alcanzar una solución, pueden ser engañosos dependiendo del escenario estudiado. En nuestro caso hemos utilizado el tiempo en lugar del número de evaluaciones, en primer lugar porque nuestra hipótesis argumenta que el tiempo ahorrado en el cruce y la mutación se puede gastar en mejorar las subpoblaciones y se pueden lograr más operaciones y migraciones. Además, estamos utilizando un número diferente de islas (con diferentes tamaños de subpoblación) y eso podría llevar a diferentes tiempos de ejecución, por lo que sería difícil comparar diferentes tiempos y calidad de soluciones al mismo tiempo.

Hemos usado el framework ECJ [22] para ejecutar los experimentos. El código fuente usado puede descargarse de nuestro repositorio GitHub[2] bajo una licencia LGPL V3.

El modelo de isla se ha ejecutado de forma asíncrona, utilizando el modelo de intercambio distribuido de interpoblación de ECJ, en un cluster de 16 nodos, cada uno con 16

[2]https://github.com/hpmoon/hpmoon-islands

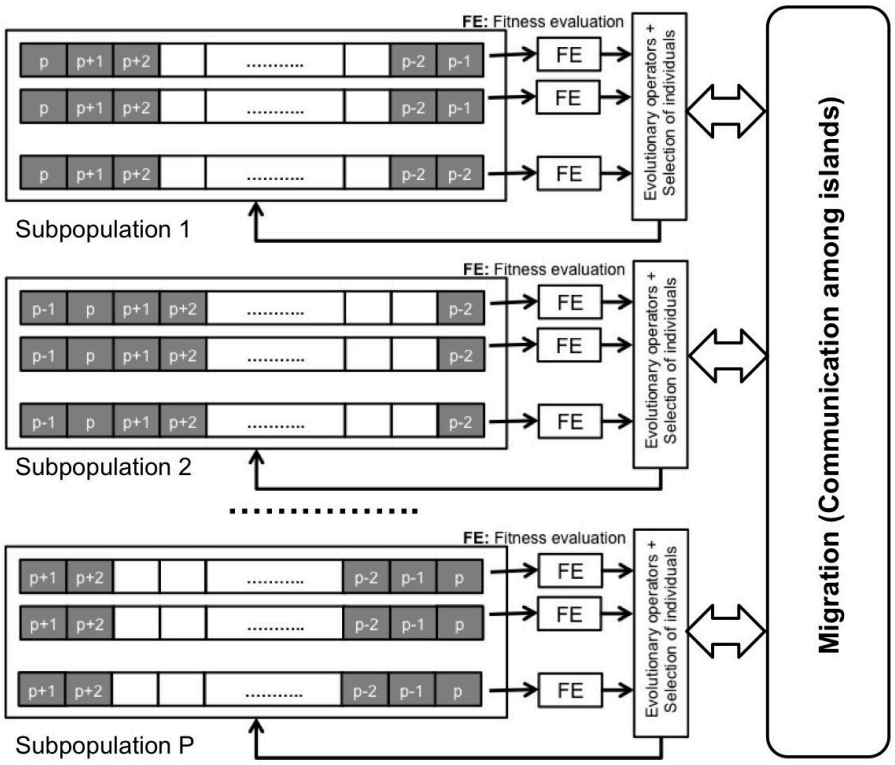


Figura 1. ASO solapado automáticamente (A): cada isla p modifica los $p + c$, p_{th} y $p - c$ componentes (en gris) de los individuos usando operadores genéticos (cruce y mutación). Además, cada isla evalúa a sus propios individuos usando el cromosoma completo. Después de un número dado de generaciones coopera con las otras islas a través de la migración. Se calcula c dependiendo del número de islas. En este caso, $c = 2$.

Nombre del parámetro	*Valor*
Tamaño global de la población (N)	1024
Selección	Torneo Binario
Tipo de reemplazo	Generacional
Tipo de crossover	SBX
Tipo de mutación	Polinomial
Probabilidad de mutación	$1/L$
Individuos por migración	1
Generaciones entre migración	5
Selección para migración	Torneo Binario
Ejecuciones por configuración	30
Número de islas (P)	8, 16, 32, 64 and 128
Tamaño del cromosoma (L)	2048
Tiempo de ejecución (s)	100

Tabla I
PARÁMETROS Y OPERADORES USADOS EN LOS EXPERIMENTOS.

procesadores Intel(R) Xeon(R) CPU E5520 @2.27GHz, 16 GB RAM, tarjetas de red Broadcom NetXtreme II BCM5716 1000Base-T (C0) PCI Express, CentOS 6.8 y Java Versión 1.8.0_80.

El conjunto total de parámetros usados se muestra en la Tabla I.

V. Resultados

Para calcular la calidad de los FPs obtenidos en cada configuración se han utilizado diferentes métricas, explicadas anteriormente. Como el HV requiere que se calcule un punto de referencia, hemos elegido el valor (1,9) ya que ninguna de las soluciones existentes en todos los frentes obtenidas durante todas las ejecuciones están dominadas por él.

Se ha realizado una prueba de significación de Kruskal-Wallis a las métricas de todas las ejecuciones de las configuraciones, ya que la prueba de Kolmogorov-Smirnov detectó distribuciones no normales. Los resultados medios de cada configuración se muestran en la Tabla II. Como se explicó anteriormente, cuando $P = 8$ los resultados de A son equivalentes a los obtenidos por D (porque $c = 0$), y cuando $P = 16$ son equivalentes a S ($c = 1$). Nos referimos a esta equivalencia con la palabra *Equiv-* en las tablas.

Los resultados muestran que la división del cromosoma produce una mejora en todos los indicadores de calidad utilizando la versión automática (A) (Tabla II), superando incluso los métodos solapado (S) y disjunto (D). Por lo tanto, existe algún tipo de punto límite en la longitud de los cromosomas donde un método será preferible a otro, además de depender del número de islas y del tamaño de la población.

Esto puede explicarse comparando el número de soluciones no dominadas de cada frente y el número medio de generaciones (Tabla III). Aumentar el número de islas implica más generaciones con todos los métodos (lógicamente, ya que hay menos individuos en cada isla). Pero también, los métodos ASO se acercan al número de generaciones con la *baseline* al aumentar el número de islas. Sin embargo el tiempo de

	HV				Spread				IGD			
#Island	B	D	S	A	B	D	S	A	B	D	S	A
ZDT1												
8	0.891	**0.953**	0.937	Equiv-D	**0.681**	**0.635** B	0.661 D	Equiv-D	0.015	**0.002**	0.005	Equiv-D
16	0.884	0.850	**0.942**	Equiv-S	**0.705**	0.908	**0.670** B	Equiv-S	0.016	0.022	**0.004**	Equiv-S
32	0.851	0.674	0.859 B	**0.900**	**0.754**	0.868	0.826 D	**0.763** B	0.023	0.062	0.020 B	**0.012**
64	**0.800**	0.608	0.697	**0.824** B	**0.808**	0.880	**0.861** B	**0.823** B	0.033	0.078	0.056	**0.027**
128	0.735	0.582	0.613	**0.745**	**0.841**	0.888	0.878 D	0.865 S	0.047	0.084	0.075	**0.043**
ZDT2												
8	0.832	**0.895**	0.869	Equiv-D	**0.849**	**0.886** B	0.853 D	Equiv-D	0.023	**0.006**	0.013	Equiv-D
16	0.831	0.833 B	**0.884**	Equiv-S	**0.810**	1.001	**0.802** B	Equiv-S	0.023	0.022 B	**0.009**	Equiv-S
32	0.800	0.628	0.800 B	**0.817**	**0.848**	0.974	0.983 D	0.908	0.031	0.082	0.032 B	**0.027**
64	**0.729**	0.491	0.623	0.716	**0.909**	0.967	0.979 D	**0.997** BD	**0.052**	0.121	0.084	**0.055** B
128	**0.630**	0.441	0.500	**0.614** B	**0.957**	0.989	0.978 D	**0.994** BS	**0.080**	0.136	0.119	**0.085** B
ZDT3												
8	0.917	**0.971**	0.960	Equiv-D	**0.843**	**0.854** B	0.868 D	Equiv-D	0.009	**0.001**	0.004	Equiv-D
16	0.911	0.876 B	**0.963**	Equiv-S	**0.864**	0.899 B	**0.837** B	Equiv-S	0.010	0.014	**0.003**	Equiv-S
32	0.884	0.710	0.883 B	**0.931**	**0.856**	**0.870** B	**0.842** B	**0.842** BDS	0.013	0.032	0.013 B	**0.008**
64	0.828	0.645	0.728	**0.854**	**0.878**	**0.896** B	**0.871** BD	**0.873** BDS	**0.019**	0.040	0.030	**0.016** B
128	**0.770**	0.620	0.651	**0.773** B	**0.887**	**0.901** B	**0.890** BD	**0.885** BDS	**0.026**	0.043	0.039	**0.025** B
ZDT6												
8	0.271	**0.398**	0.323	Equiv-D	**0.982**	**0.982** B	0.994	Equiv-D	0.171	**0.115**	0.149	Equiv-D
16	0.275	0.295 B	**0.354**	Equiv-S	**0.981**	**0.970** B	1.006	Equiv-S	0.170	0.161 B	**0.136**	Equiv-S
32	0.239	0.123	0.240	**0.254**	**0.989**	**0.991** B	**0.982** BD	**0.999** BD	0.186	0.235	0.185 B	**0.179**
64	**0.184**	0.068	0.125	**0.178** B	**0.985**	**0.982** B	**0.992** B	**0.995** BS	**0.209**	0.259	0.235	0.212
128	**0.128**	0.051	0.071	**0.124** B	**0.991**	**0.992** B	**0.988** BD	1.003	**0.233**	0.266	0.257	**0.235** B

Tabla II

MÉTRICAS DE CALIDAD MEDIA OBTENIDAS DESPUÉS DE 30 EJECUCIONES POR CONFIGURACIÓN, PARA LOS 4 MÉTODOS COMPARADOS: *baseline* (B), DISJUNTO (D), SOLAPADO (S) Y SOLAPADO AUTOMÁTICO (A). LOS ACRÓNIMOS QUE APARECEN JUNTO A LOS VALORES INDICAN QUE NO HAY DIFERENCIAS SIGNIFICATIVAS CON RESPECTO A ESE MÉTODO PARA ESE VALOR. LOS MEJORES VALORES ESTÁN MARCADOS EN NEGRITA. EQUIV-X IMPLICA QUE EL VALOR ES EL MISMO QUE EL DE EJECUTAR X, YA QUE AMBAS CONFIGURACIONES SERÍAN IGUALES.

	Average solutions per front				Generations			
#Island	B	D	S	A	B	D	S	A
ZDT1								
8	137.733	47.167	119.933 B	Equiv-D	175.067	225.667	207.933	Equiv-D
16	92.633	38.233	47.533 D	Equiv-S	204.867	238.533	232.900	Equiv-S
32	56.167	41.167	28.667	32.567 S	225.433	243.833	242.533	242.000 S
64	50.900	49.667 B	35.600	25.367	236.500	245.967	245.700	244.033 D
128	45.767	64.933	44.167 B	31.867	242.800	247.000	247.000 D	245.733
ZDT2								
8	64.267	22.733	56.867 B	Equiv-D	175.633	226.000	208.100	Equiv-D
16	52.300	10.733	20.067 D	Equiv-S	205.033	238.867	233.100	Equiv-S
32	34.400	10.367	10.433 D	17.900 S	225.600	244.267	242.633	242.000 S
64	24.600	10.900	9.267 D	12.867 DS	236.700	246.000	245.767	244.033 D
128	18.667	17.833 B	10.367	12.933 S	243.367	247.000	247.000 d	245.833
ZDT3								
8	163.767	83.367	119.467	Equiv-D	176.533	226.400	207.700	Equiv-D
16	108.467	49.433	47.700 D	Equiv-S	205.333	238.467	232.933	Equiv-S
32	72.133	44.333	31.833	36.533 S	225.100	243.900	242.433	242.000 S
64	54.867	57.600 B	40.633	29.300	236.333	245.933	245.733	244.000 D
128	50.600	72.133	49.033 B	34.933	243.200	247.000	247.000 D	245.833
ZDT6								
8	22.133	13.833	14.433 D	Equiv-D	175.733	226.100	207.767	Equiv-D
16	20.767	15.867	13.300 D	Equiv-S	205.033	238.433	232.933	Equiv-S
32	22.600	10.700	10.033 D	11.967 DS	225.833	243.667	242.500	242.000 S
64	19.033	11.267	10.533 D	10.567 DS	236.433	245.900	245.833	244.000 D
128	15.800	24.500	11.600	12.233 DS	243.800	247.000	247.000 D	245.533

Tabla III

PROMEDIO DE GENERACIONES Y PROMEDIO DE SOLUCIONES POR FRENTE, OBTENIDOS DESPUÉS DE 30 EJECUCIONES POR CONFIGURACIÓN, PARA LOS 4 MÉTODOS COMPARADOS: *baseline* (B), DISJUNTO (D), SOLAPADO (S) Y SOLAPADO AUTOMÁTICO (A). LOS ACRÓNIMOS QUE APARECEN JUNTO A LOS VALORES INDICAN QUE NO HAY DIFERENCIAS SIGNIFICATIVAS CON RESPECTO A ESE MÉTODO PARA ESE VALOR. EQUIV-X IMPLICA QUE EL VALOR ES EL MISMO QUE EL DE EJECUTAR X, YA QUE AMBAS CONFIGURACIONES SERÍAN IGUALES.

migración es lo suficientemente grande como para no mejorar el número de generaciones con respecto a la *baseline*, incluso en el menor número de islas. Por lo tanto, más generaciones no significan necesariamente mejorar la solución del FP global, sino centrarse en diferentes elementos del cromosoma. Como se ha dicho anteriormente, cada isla desconoce los FPs de las otras islas, y están tratando de optimizar sus soluciones independientemente. Con respecto al número promedio de soluciones por frente, hay una clara diferencia con la *baseline*, donde este valor es en la mayoría de los casos, menos de la mitad. El número de soluciones no dominadas también implica un mejor indicador de Spread, donde la *baseline* obtiene mejores (o no significativamente diferentes) valores en la mayoría de las configuraciones comparadas.

VI. Conclusiones

Los problemas que requieren un alto rendimiento y que tratan con un gran número de variables de decisión pueden aprovecharse de la división del espacio de decisión que proporcionan los algoritmos paralelos y distribuidos. Esto se puede hacer en dMOEAs mediante la aplicación selectiva de operadores (ASO), es decir, dividiendo el cromosoma en diferentes partes, cada una modificada por una isla diferente. Este trabajo compara un NSGA-II distribuido, como *baseline*, con tres estrategias diferentes para separar el cromosoma (partes disjuntas o solapadas), utilizando distintos números de islas. Los resultados muestran que estos métodos pueden lograr

métricas de mejor calidad que el *baseline* en la misma cantidad de tiempo.

Los resultados obtenidos también muestran que al aumentar el número de islas, el método de solapamiento automático (A) mejora significativamente los resultados con respecto a los métodos disjunto y solapado. El estudio de este factor con más tipos de problemas y nuevas configuraciones del tamaño de la población y la longitud de los cromosomas puede abordarse en el futuro. Por ejemplo, comparar diferentes maneras de calcular c usando funciones lineales, logarítmicas o exponenciales dependiendo del tamaño de la población, número de islas, tamaño del cromosoma, u otros valores. Además, se podría realizar un análisis de la diversidad de los individuos de cada isla durante la ejecución del algoritmo para comprender su influencia en los resultados.

Asimismo, podrían utilizarse implementaciones más distribuidas en varios sistemas (como GPUs o clusters heterogéneos) con diferentes cantidades de islas/procesadores para realizar un estudio de escalabilidad de los diferentes métodos, siendo el tiempo de transmisión entre islas un tema relevante a tratar. También pueden compararse otros MOEAs disponibles en la literatura, como SPEA o MOEA/D. Además, podrían utilizarse otros benchmarks y problemas reales para validar este enfoque.

Referencias

[1] A. M. Mora, P. García-Sánchez, J. J. Merelo-Guervós, and P. A. Castillo, "Pareto-based multi-colony multi-objective ant colony optimization algorithms: an island model proposal," *Soft Comput.*, vol. 17, no. 7, pp. 1175–1207, 2013.

[2] F. Luna and E. Alba, "Parallel multiobjective evolutionary algorithms," in *Springer Handbook of Computational Intelligence*, J. Kacprzyk and W. Pedrycz, Eds. Springer, 2015, pp. 1017–1031.

[3] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello, "A survey of multiobjective evolutionary algorithms for data mining: Part I," *IEEE T. Evolut. Comput.*, vol. 18, no. 1, pp. 4–19, 2014.

[4] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer, 2015.

[5] E.-G. Talbi, "A unified view of parallel multi-objective evolutionary algorithms," *J. Parallel Distrib. Comput.*, pp.–, 2018, in press. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S074373151830279X

[6] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *Int. T. Oper. Res.*, vol. 20, no. 1, pp. 1–48, 2013.

[7] Y. Gong, W. Chen, Z. Zhan, J. Zhang, Y. Li, Q. Zhang, and J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Appl. Soft Comput.*, vol. 34, pp. 286–300, 2015.

[8] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[9] J. Branke, H. Schmeck, K. Deb, and R. S. Maheshwar, "Parallelizing multi-objective evolutionary algorithms: cone separation," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2004, 19-23 June 2004, Portland, OR, USA*. IEEE, 2004, pp. 1952–1957.

[10] D. Kimovski, J. Ortega, A. Ortiz, and R. Baños, "Parallel alternatives for evolutionary multi-objective optimization in unsupervised feature selection," *Expert Syst. Appl.*, vol. 42, no. 9, pp. 4239–4252, 2015.

[11] P. García-Sánchez, J. Ortega, J. González, P. A. Castillo, and J. J. Merelo, "Addressing high dimensional multi-objective optimization problems by coevolutionary islands with overlapping search spaces," in *Applications of Evolutionary Computation - 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, G. Squillero and P. Burelli, Eds., vol. 9598. Springer, 2016, pp. 107–117.

[12] E. Talbi, S. Mostaghim, T. Okabe, H. Ishibuchi, G. Rudolph, and C. A. C. Coello, "Parallel approaches for multiobjective optimization," in *Multiobjective Optimization, Interactive and Evolutionary Approaches [outcome of Dagstuhl seminars]*., ser. Lecture Notes in Computer Science, J. Branke, K. Deb, K. Miettinen, and R. Slowinski, Eds., vol. 5252. Springer, 2008, pp. 349–372.

[13] A. J. Nebro and J. J. Durillo, "A study of the parallelization of the multi-objective metaheuristic MOEA/D," in *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*, ser. Lecture Notes in Computer Science, C. Blum and R. Battiti, Eds., vol. 6073. Springer, 2010, pp. 303–317.

[14] K. Deb, P. Zope, and A. Jain, "Distributed computing of Pareto-optimal solutions with evolutionary algorithms," in *Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003, Faro, Portugal, April 8-11, 2003, Proceedings*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds., vol. 2632. Springer, 2003, pp. 534–549.

[15] W. Zhi-xin and G. Ju, "A parallel genetic algorithm in multi-objective optimization," in *Control and Decision Conference, 2009. CCDC '09. Chinese*, June 2009, pp. 3497–3501.

[16] N. Xiao and M. P. Armstrong, "A specialized island model and its application in multiobjective optimization," in *Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 12-16, 2003. Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U. O'Reilly, H. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, Eds., vol. 2724. Springer, 2003, pp. 1530–1540.

[17] M. Märtens and D. Izzo, "The asynchronous island model and NSGA-II: study of a new migration operator and its performance," in *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 1173–1180.

[18] B. Dorronsoro, G. Danoy, A. J. Nebro, and P. Bouvry, "Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution," *Computers & OR*, vol. 40, no. 6, pp. 1552–1563, 2013.

[19] A. Atashpendar, B. Dorronsoro, G. Danoy, and P. Bouvry, "A scalable parallel cooperative coevolutionary PSO algorithm for multi-objective optimization," *J. Parallel Distrib. Comput.*, vol. 112, pp. 111–125, 2018.

[20] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.

[21] E. Alba and G. Luque, "Evaluation of parallel metaheuristics," in *Empirical Methods for the Analysis of Algorithms, Workshop EMAA 2006, Proceedings*, L. Paquete, M. Chiarandini, and D. Basso, Eds., Reykjavik, Iceland, 2006, pp. 9–14.

[22] S. Luke *et al.*, "ECJ: A Java-based Evolutionary Computation and Genetic Programming Research System," 2009, available at http://www.cs.umd.edu/projects/plus/ec/ecj.

Análisis de Diseños Paralelos Multiobjetivo y Políticas de Planificación en Biología Evolutiva

Sergio Santander-Jiménez
INESC-ID, IST, Universidade de Lisboa,
Lisboa 1000-029, Portugal
sergio.jimenez@tecnico.ulisboa.pt

Miguel A. Vega-Rodríguez
INTIA, Universidad de Extremadura,
Cáceres 10003, España
mavega@unex.es

Leonel Sousa
INESC-ID, IST, Universidade de Lisboa,
Lisboa 1000-029, Portugal
leonel.sousa@ist.utl.pt

***Resumen*—Las metaheurísticas paralelas representan una de las técnicas de mayor popularidad para abordar la resolución de problemas de optimización complejos. Sin embargo, una de las cuestiones principales que surgen al emplear estos métodos viene dada por la aparición potencial de problemas de desequilibrio de carga. De hecho, la complejidad de los problemas de optimización actuales obliga a la adopción de múltiples estrategias de búsqueda condicionales dentro de la metaheurística, lo cual incide en el impacto que implica el desequilibrio de carga en el rendimiento paralelo. Este trabajo analiza diferentes políticas de planificación y diseños algorítmicos para abordar el desequilibrio de carga en la metaheurística *Multiobjective Shuffled Frog-Leaping Algorithm*. Tomando como caso de estudio un problema de biología evolutiva, la reconstrucción filogenética, abordamos la evaluación de tres políticas de planificación en el algoritmo original, así como una alternativa de diseño basada en el uso de contadores. Los resultados obtenidos en cuatro bases de datos reales dan cuenta de la influencia de las aproximaciones estudiadas en los tiempos de sobrecarga, destacando los beneficios de integrar políticas dinámicas y diseños que respeten las mecánicas del algoritmo.**

***Index Terms*—Biología evolutiva, desequilibrio de carga, computación paralela, metaheurísticas multiobjetivo**

I. Introducción

La resolución eficiente de problemas de optimización se ha convertido en una de las líneas de investigación predominantes en multitud de dominios científicos. En contextos reales, es obligatorio el diseño de aproximaciones algorítmicas innovadoras para cumplir los requisitos de calidad de solución y, al mismo tiempo, lidiar con la naturaleza NP-completa de estos problemas. Sin embargo, la dificultad de estos procesos de optimización continúa creciendo con la adopción de formulaciones más realistas, escenarios *Big Data*, múltiples funciones objetivo, etc. Así, es habitual la presencia de tiempos de ejecución elevados que resultan inasumibles incluso al emplear técnicas estocásticas de resolución. Las metaheurísticas paralelas han demostrado aplicabilidad potencial para lidiar con estas cuestiones, obteniendo resultados significativos [1].

A pesar de la naturaleza intrínsecamente paralela de las metaheurísticas basadas en población, estos métodos pueden verse afectados por dos fuentes principales de desequilibrio de carga. Al nivel del diseño metaheurístico, los operadores de búsqueda pueden mostrar distintos requisitos temporales, hecho que puede volverse más remarcable en situaciones de cambio de contexto de explotación a exploración (o viceversa). Al nivel de implementación del problema, las soluciones generadas pueden mostrar características divergentes que influyen, por ejemplo, en los cálculos efectuados en las funciones objetivo, dando lugar a tiempos de evaluación no homogéneos. Estas cuestiones implican un aumento de la sobrecarga y, consecuentemente, un empeoramiento del rendimiento paralelo.

Así, las metaheurísticas paralelas deben considerar este tipo de factores para conseguir una explotación precisa de los recursos hardware. Este trabajo se focaliza en el análisis de distintas políticas de planificación y diseños para abordar problemas de desequilibrio de carga. Adoptamos como caso de estudio un problema de biología evolutiva: la inferencia de relaciones ancestrales entre especies [2]. En escenario monoobjetivo, existen propuestas como IQ-TREE [3] y GARLI [4] que han dado cuenta de los beneficios de emplear computación de altas prestaciones y métodos bioinspirados de búsqueda en este problema. Las aproximaciones multiobjetivo también representan una tendencia significativa en el caso de datos de ADN [5], al contribuir potencialmente a la resolución de problemas de incongruencia en contextos reales.

Abordaremos este problema biológico desde la perspectiva multiobjetivo, usando una metaheurística basada en población denominada *Multiobjective Shuffled Frog-Leaping Algorithm* (MO-SFLA) [6]. Bajo implementaciones MPI+OpenMP, examinaremos el impacto de distintas políticas de planificación (estática, guiada y dinámica) en el rendimiento paralelo del diseño algorítmico original, así como diseños alternativos de la aproximación. El análisis planteado será efectuado sobre cuatro instancias reales del problema basadas en secuencias de proteínas, las cuales presentan un nivel superior de complejidad con respecto a otros tipos de datos biológicos [7]. La bondad de los resultados obtenidos será estudiada mediante comparativas con otros métodos del estado del arte.

Este artículo se organiza del siguiente modo. La Sección II detalla la formulación del problema estudiado. La Sección III describe diseños y políticas de planificación para MO-SFLA, examinándose resultados en la Sección IV. Finalmente, la Sección V destaca conclusiones y líneas futuras de trabajo.

Este trabajo ha sido parcialmente financiado por la AEI (Agencia Estatal de Investigación, España) y el FEDER (Fondo Europeo de Desarrollo Regional, UE), bajo el proyecto TIN2016-76259-P (proyecto PROTEIN), así como por la FCT (Fundação para a Ciência e a Tecnologia, Portugal) bajo los proyectos UID/CEC/50021/2013 y LISBOA-01-0145-FEDER-031901 (PTDC/CCI-COM/31901/2017, HiPErBio). Sergio Santander-Jiménez agradece a la FCT el soporte recibido a través del contrato postdoctoral SFRH/BPD/119220/2016.

II. Formulación del Problema

La inferencia de relaciones ancestrales entre organismos representa uno de los problemas más importantes en biología evolutiva [2]. Dicho problema involucra el estudio de un alineamiento múltiple de secuencias de tamaño $N \times M$ (donde N es el número de secuencias y M la longitud de secuencia), el cual es procesado para identificar patrones de divergencia y similitud. Esta información es utilizada para inferir organismos ancestrales hipotéticos y definir el curso de la evolución a través de una estructura arborescente, conocida como árbol filogenético $T = (V, E)$. En particular, los nodos internos del conjunto de nodos V describen los ancestros potenciales cuya evolución dio lugar a los organismos caracterizados en el alineamiento de entrada, organismos que se localizan en los nodos hoja. La definición de relaciones evolutivas se efectúa a través del conjunto de ramas E, donde se definen los enlaces ancestro-descendiente entre nodos emparentados.

La aplicación de procedimientos de optimización en este problema tienen por objeto identificar el árbol filogenético que maximice o minimice un determinado criterio de optimalidad biológico, lo cual implica la exploración de un espacio de filogenias S. La principal cuestión que surge en este contexto viene dada por el crecimiento exponencial de S con el número de secuencias N, definiéndose el número de posibles soluciones candidatas en S de acuerdo al doble factorial $(2N-5)!!$ [2]. Otros factores, tales como la longitud de secuencia M, el tipo de datos contenido en el alineamiento (por ejemplo, aminoácidos) y la consideración conjunta de múltiples criterios de optimalidad, contribuyen al elevado coste computacional del problema. Todo ello impide la aplicación de aproximaciones serie tradicionales.

En este artículo, afrontamos estas cuestiones mediante aproximaciones metaheurísticas paralelas. La formulación adoptada del problema involucra la optimización de dos funciones objetivo: parsimonia y verosimilitud. Por un lado, la parsimonia cuantifica el número de cambios observados entre las secuencias de nodos emparentados. El objetivo radica en inferir la hipótesis evolutiva más simple, dando prioridad a la solución $T = (V, E)$ que minimice el número de cambios medidos a través del valor de parsimonia $P(T)$ [2]:

$$P(T) = \sum_{i=1}^{M} \sum_{(u,v)\in E} C(u_i, v_i), \tag{1}$$

donde $(u, v) \in E$ representa la rama de la topología que conecta dos nodos $u, v \in V$, u_i y v_i son los valores de estado del i-ésimo carácter de las secuencias correspondientes a u y v, y $C(u_i, v_i)$ indica las mutaciones observadas entre u y v ($C(u_i, v_i)$=1 si $u_i \neq v_i$ y $C(u_i, v_i)$=0 en caso contrario).

Por su parte, la función de verosimilitud establece una medida probabilística de la plausibilidad de que el árbol filogenético evaluado haya dado lugar efectivo a la diversidad observada en el alineamiento de entrada. Los cálculos de verosimilitud se realizan conforme a modelos de evolución que definen las probabilidades de observar mutaciones desde cada posible valor de estado de carácter a cualquier otro. Bajo esta función, se da prioridad a la solución $T = (V, E)$ que maximice el valor de verosimilitud $L(T)$ [2]:

$$L(T) = \prod_{i=1}^{M} \sum_{x,y\in\Lambda} \pi_x \left[P_{xy}(t_{ru}) L_p\left(u_i = y\right)\right] \times \left[P_{xy}\left(t_{rv}\right) L_p\left(v_i = y\right)\right], \tag{2}$$

donde Λ es el alfabeto de estados de carácter (aminoácidos en el caso de secuencias de proteínas), π_x la probabilidad estacionaria de observar el valor de estado x, $r \in V$ la raíz de T con hijos $u, v \in V$, $P_{xy}(t)$ la probabilidad de mutación de x a otro estado y en un tiempo t (siendo t_{ru} y t_{rv} la longitud de las ramas $(r, u), (r, v) \in E$), y $L_p(u_i = y)$, $L_p(v_i = y)$ las verosimilitudes parciales de observar y en u_i y v_i.

III. Descripción de Métodos

MO-SFLA es una metaheurística que integra mecanismos multiobjetivo en el método *frog leaping optimization* [8]. En cada iteración, los *popSize* individuos que conforman la población son ordenados (según sus valores de fitness) y distribuidos uniformemente en *m* particiones denominadas memeplexes. Cada memeplex contiene *n=popSize/m* individuos, cuya evolución se produce separadamente durante n_l pasos de aprendizaje. Una vez todos los memeplexes han sido procesados, son combinados para permitir una compartición de información entre todos los memeplexes, actualizando la población para la siguiente generación.

Para la adaptación al problema abordado, MO-SFLA incluye una representación del individuo basada en matrices de distancia filogenéticas. Una solución candidata vendrá codificada por una matriz simétrica δ que contiene en cada entrada $\delta[x, y]$ la distancia evolutiva entre los organismos x e y. El mapeo de soluciones desde este espacio de decisión matricial al espacio de filogenias se lleva a cabo mediante el método de reconstrucción de árboles filogenéticos BIONJ [2]. La etapa de inicialización de la población considerará matrices de distancia calculadas a partir de topologías filogenéticas iniciales generadas mediante técnicas de *bootstraping* [2].

El Pseudocódigo 1 detalla el diseño original de MO-SFLA, el cual realiza el proceso de optimización hasta que un determinado criterio de parada (en nuestro caso, un máximo de evaluaciones) es satisfecho. Al comienzo de cada generación, los individuos de la población son ordenados según su calidad multiobjetivo (línea 7 en el Pseudocódigo 1), manteniendo las *popSize* soluciones más prometedoras conforme a sus valores de ranking Pareto (*fast non-dominated sort*) y densidad (distancia de *crowding*) [9]. Tras la ordenación, se definen los memeplexes mediante la distribución de individuos de manera iterativa, asignando el individuo i-ésimo al memeplex $i\,\%m$.

La generación de nuevas soluciones candidatas dentro de cada memeplex se realiza mediante la definición de distintos operadores de búsqueda (líneas 17-26), que son aplicados según el estado actual del proceso de optimización. Dado un memeplex Mem_i, generamos una nueva solución P'_{new} conforme a la siguiente formulación:

Pseudocódigo 1 MO-SFLA Paralelo

```
1:  MPI_Init /* inicializando proceso MPI #i */
2:  #pragma omp parallel num_threads (num_hilos)
3:  mientras ! criterio de parada (maxEval) hacer
4:      /* Definición y asignación de memeplexes */
5:      #pragma omp single
6:      si proceso maestro entonces
7:          {Mem_1 ... Mem_m} ← Ordenación y Distribución (P, popSize)
8:          BestGlobal ← Identificar Mejor Global ({Mem_1 ... Mem_m})
9:          /* ∀i : i = 2 a m */
10:         MPI_Send (Mem_i, n, i), MPI_Send (BestGlobal, 1, i)
11:     si no
12:         MPI_Recv (Mem_i, n, master_id), MPI_Recv (BestGlobal, 1, master_id)
13:     fin si
14:     BestLocal ← Identificar Mejor Local (Mem_i)
15:     /* Tareas paralelas: evolución del memeplex asignado */
16:     #pragma omp for
17:     para j = 1 to n_l hacer
18:         P'_new ← Aprendizaje Mejor Local (BestLocal, Mem_i(n−j))
19:         si ! P'_new ≻ Mem_i(n−j) entonces
20:             P'_new ← Aprendizaje Mejor Global (BestGlobal, Mem_i(n−j))
21:             si ! P'_new ≻ Mem_i(n−j) entonces
22:                 P'_new ← Búsqueda Local (Mem_i(n−j))
23:             fin si
24:         fin si
25:         Mem_i(n−j) ← P'_new
26:     fin para
27:     /* Entrega de resultados */
28:     #pragma omp single
29:     si proceso maestro entonces
30:         /* ∀i : i = 2 a m */
31:         MPI_Recv (Mem_i, n, i)
32:         P ← Mezclar Memeplexes (P, {Mem_1 ... Mem_m})
33:         ParetoFront ← Actualizar Frente de Pareto (P)
34:     si no
35:         MPI_Send (Mem_i, n, master_id)
36:     fin si
37: fin mientras
38: MPI_Finalize /* finalizando proceso MPI #i */
```

$$D_{xy} = rand() \cdot (Ref.\delta[x,y] - Mem_{ij}.\delta[x,y]), \quad (3)$$

$$P'_{new}.\delta[x,y] = Mem_{ij}.\delta[x,y] + D_{xy}, \quad (4)$$

donde *rand()* es un número aleatorio de una distribución uniforme en el rango $[0, 1]$ y Mem_{ij} el individuo en Mem_i procesado en el paso de aprendizaje j. *Ref* representa una solución que es tomada como referencia durante este proceso. Inicialmente, se toma como *Ref* el mejor individuo local de Mem_i (línea 18). En caso de que la solución resultante P'_{new} no mejore a Mem_{ij}, se repite el proceso anterior tomando como *Ref* el mejor individuo global de toda la población (línea 20). Si sigue sin observarse mejora, se aplica una búsqueda local basada en los operadores topológicos *nearest neighbour interchange* y *subtree pruning and regrafting* [2] (línea 22).

Al final de la generación (líneas 32, 33), los memeplexes son reunificados para actualizar la estructura de la población, la cual es usada a su vez para identificar soluciones no dominadas y así almacenarlas en la estructura *ParetoFront*. Al satisfacerse el criterio de parada, dicha estructura contendrá las soluciones no dominadas encontradas a lo largo de la ejecución, representando la salida del algoritmo.

III-A. Diseños Paralelos y Políticas de Planificación

MO-SFLA presenta dos características clave que hacen a esta metaheurística adecuada para su ejecución en clusters multicore: 1) el procesamiento de memeplexes puede efectuarse de manera independiente de una memeplex a otro; y 2) la generación de soluciones candidatas dentro del mismo memeplex no presenta dependencias de un paso de aprendizaje (iteración del bucle) a otro. Estos dos niveles son apropiados para su explotación mediante MPI+OpenMP [10], [11].

En cada generación, un proceso maestro se encargará inicialmente de la gestión de la población y la definición de memeplexes, distribuyendo adecuadamente los individuos. Los memeplexes resultantes serán asignados a distintos procesos trabajadores mediante las funciones MPI `MPI_Send` y `MPI_Recv` (línea 10 en el Pseudocódigo 1 para el lado emisor - maestro, y línea 12 para el lado receptor - trabajador). Concluida la comunicación, cada proceso (incluyendo el maestro) realizará el procesamiento de los memeplexes asignados (líneas 15-26), usando la directiva `#pragma omp for` para distribuir las iteraciones del bucle entre los hilos de ejecución. Terminado el procesamiento de memeplexes, los resultados obtenidos son comunicados mediante funciones MPI al proceso maestro (líneas 31 -maestro, y 35 - trabajador), efectuándose entonces las tareas finales de la generación.

En este diseño, los procesos se ven afectados por distintas fuentes de desequilibrio de carga. En primer lugar, la metaheurística plantea hasta tres estrategias de búsqueda diferentes que pueden ser ejecutadas en una misma iteración del bucle paralelo definido en la línea 17. Así, cada iteración puede implicar un número variable de operaciones en conformidad con el éxito de las estrategias aplicadas. Además, la generación de una nueva solución candidata involucra dos mecanismos específicos del problema: la reconstrucción del árbol y el cómputo de las funciones objetivo. Estas operaciones se caracterizan no solo por el hecho de incluir cálculos computacionalmente costosos, sino también por presentar divergencias en tiempo de ejecución según las características de la solución candidata.

Para afrontar dichos problemas de desequilibrio de carga, consideraremos primero el uso de las políticas de planificación de bucles paralelos definidas en el estándar OpenMP. Concretamente, examinamos tres aproximaciones diferentes:

1. *Estática*. Las iteraciones son divididas en bloques que se asignan de manera estática a cada hilo en modo *round-robin* según el identificador de hilo. Dado que esto no permite la resolución de desequilibrios de carga potenciales (al tener cada hilo sus iteraciones asignadas a priori), usaremos esta política como punto de referencia.
2. *Dinámica*. En esta aproximación, los hilos solicitan gradualmente nuevas iteraciones al terminar las que han ido manejado previamente. Así, los hilos más rápidos pueden comenzar el procesamiento de nuevos bloques inmediatamente tras acabar su carga de trabajo inicial.
3. *Guiada*. De manera similar a la aproximación dinámica, la planificación guiada permite la asignación gradual de iteraciones a los hilos. Se asignan nuevas iteraciones mediante bloques de tamaño proporcional al número de iteraciones aún no asignadas dividido entre el número de hilos, reduciendo estos bloques de manera exponencial hasta un mínimo de 1 (configuración estándar).

Pseudocódigo 2 MO-SFLA Paralelo Basado en Contadores

```
1:  MPI_Init /* inicializando proceso MPI #i */
2:  #pragma omp parallel num_threads (num_hilos)
3:  mientras ! criterio de parada (maxEval) hacer
4:      #pragma omp single
5:          BestGlobal, {Mem_1 ... Mem_m} ← Gestionar Población (P, popSize)
6:          Asignación Memeplex-Proceso (BestGlobal, Mem_i, i) /*∀i : i = 2 a m*/
7:          Best_local ← Identificar Mejor Local (Mem_i)
8:      #pragma omp for
9:      para j = 1 to n_l hacer
10:         switch (Mem_i(n−j).counter)
11:             caso 0: P'_new ← Aprendizaje Mejor Local (Best_local, Mem_i(n−j))
12:             caso 1: P'_new ← Aprendizaje Mejor Global (Best_global, Mem_i(n−j))
13:             caso 2: P'_new ← Búsqueda Local (Mem_i(n−j))
14:         si P'_new ≻ Mem_i(n−j) || Mem_i(n−j).counter == 2 entonces
15:             Mem_i(n−j) ← P'_new, Mem_i(n−j).counter ← 0
16:         si no
17:             Mem_i(n−j).counter ← Mem_i(n−j).counter + 1
18:         fin si
19:     fin para
20:     #pragma omp single
21:         Comunicación de Resultados (Mem_i, master_id) /*∀i : i = 2 a m*/
22:         P, ParetoFront ← Tareas Finales de la Generación (P, {Mem_1 ... Mem_m})
23: fin mientras
24: MPI_Finalize /* finalizando proceso MPI #i */
```

Una alternativa a esta aproximación radica en la adaptación del diseño para que cada iteración del bucle de procesamiento de memeplexes ejecute solo una de las estrategias de búsqueda definidas. Esta idea viene plasmada en el Pseudocódigo 2, la cual se basa en el empleo de contadores para medir el número de intentos en que un individuo de la población no ha sido mejorado. En función del valor de este contador, se decide la estrategia de búsqueda a ser aplicada sobre dicho individuo (líneas 10-13 del Pseudocódigo 2). Dado un individuo Mem_{ij}, un valor de 0 en su contador implicará la generación de la nueva solución empleando el mejor individuo local de Mem_i, mientras que un valor de 1 conllevará el empleo del mejor individuo global y un valor de 2 la búsqueda local. En caso de observarse mejora, el contador asociado será reinicializado y la solución almacenada en Mem_{ij} (línea 15). En caso contrario, la solución generada es descartada y el contador de Mem_{ij} incrementado (línea 17). De esta manera, en la siguiente generación el proceso trabajador efectuará un nuevo intento sobre Mem_{ij} considerando una estrategia de búsqueda diferente. Como en la versión original, se pueden emplear las políticas de planificación de OpenMP para complementar el mecanismo de balanceo basado en contadores.

IV. Resultados Experimentales

En esta sección abordamos la evaluación experimental de los diseños paralelos y políticas de planificación considerados para MO-SFLA. Para ello, examinaremos el rendimiento paralelo obtenido en cuatro bases de datos reales de proteínas, las cuales son descritas en la Tabla I. La experimentación fue efectuada sobre 4 nodos de cómputo de una infraestructura clúster con interconexión Gigabit Ethernet. Estos nodos están compuestos por procesadores AMD Opteron 6174 de 12 cores a 2,2GHZ (un total de 48 cores disponibles) con 12MB de memoria caché L3 y 16GB de RAM DDR3, Ubuntu 14.04LTS como sistema operativo y el compilador GCC 5.2.1.

De acuerdo a las características del clúster empleado y los estudios paramétricos efectuados, los parámetros de entrada de MO-SFLA fueron establecidos a los siguientes valores: *popSize* = 128, m = 4, n = 32, n_l = 32, *maxEval* = 10000.

Tabla I
Conjuntos de Datos de Proteínas Usados en la Experimentación

Instancia	N	M	Organismo	Modelo Evolutivo [2]
M88x3329	88	3329	Hongos termófilos [12]	LG+Γ
M187x814	187	814	Hongos micorrícicos [13]	LG+Γ
M260x1781	260	1781	Beta vulgaris [14]	JTT+Γ
M355x1263	355	1263	Hemiascomicetos [15]	LG+Γ

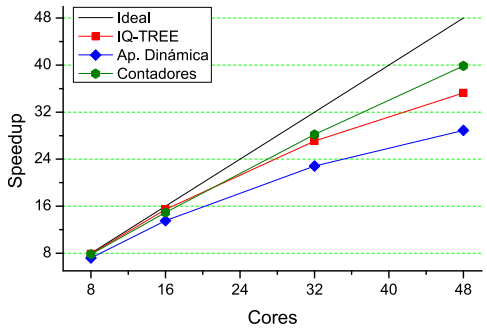
Tabla II
Comparativa Paralela (Speedups *SU* y Eficiencia *EF*)

	Estática		Dinámica		Guiada		Contadores	
Cores	SU	EF(%)	SU	EF(%)	SU	EF(%)	SU	EF(%)
M88x3329 (T_{serie} = 57111,63 segundos)								
8	6,93	86,59	**7,21**	**90,10**	7,09	88,65	**7,81**	**97,67**
16	11,58	72,38	**13,53**	**84,57**	13,31	83,18	**14,94**	**93,38**
32	19,10	59,69	**22,83**	**71,36**	22,34	69,82	**28,18**	**88,06**
48	26,18	54,54	**28,88**	**60,17**	27,84	57,99	**39,86**	**83,04**
M187x814 (T_{serie} = 44171,92 segundos)								
8	6,16	77,06	**6,54**	**81,78**	6,46	80,74	**7,70**	**96,27**
16	10,70	66,88	**12,18**	**76,12**	11,96	74,78	**14,75**	**92,19**
32	18,58	58,06	**21,85**	**68,28**	20,99	65,60	**27,71**	**86,59**
48	23,96	49,92	**28,33**	**59,03**	26,04	54,25	**39,66**	**82,63**
M260x1781 (T_{serie} = 66748,33 segundos)								
8	7,37	92,16	**7,69**	**96,18**	7,55	94,33	**7,86**	**98,28**
16	12,75	79,69	**14,18**	**88,61**	13,70	85,61	**15,65**	**97,81**
32	20,45	63,91	**23,29**	**72,78**	22,51	70,36	**29,20**	**91,25**
48	26,36	54,92	**30,21**	**62,94**	27,90	58,11	**40,56**	**84,49**
M355x1263 (T_{serie} = 96219,62 segundos)								
8	6,79	84,90	**7,47**	**93,40**	7,22	90,21	**7,92**	**98,95**
16	11,92	74,50	**13,76**	**86,03**	13,69	85,56	**15,68**	**98,00**
32	21,96	68,63	**25,59**	**79,97**	23,77	74,28	**29,87**	**93,34**
48	29,30	61,04	**32,97**	**68,69**	30,80	64,17	**42,63**	**88,82**

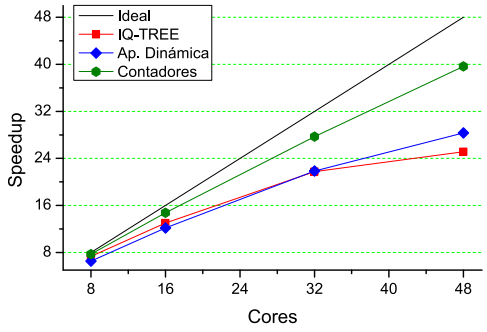
IV-A. Evaluación del Rendimiento Paralelo

Con objeto de examinar el rendimiento paralelo, emplearemos las métricas paralelas de aceleración o *speedup* y eficiencia [11]. Consideraremos los resultados temporales medianos (procedentes de 11 ejecuciones independientes por experimento) obtenidos sobre distintos tamaños del sistema (8, 16, 32 y 48 cores). La Tabla II presenta los resultados observados, así como los tiempos de la versión serie de MO-SFLA T_{serie} usados como referencia en el cálculo de las métricas paralelas.

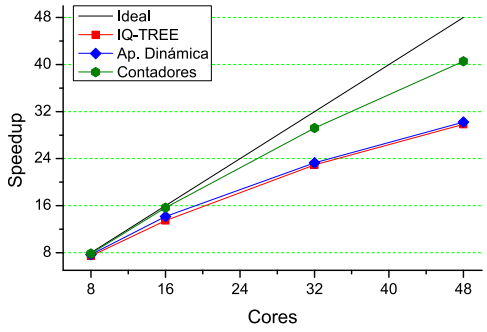
Centrándonos en primer lugar en los resultados del diseño original (columnas 2, 3 - versión estática, 4, 5 - dinámica, y 6, 7 - guiada), es posible observar el efecto del desequilibrio de carga en el rendimiento de la versión inicial estática. De hecho, la política estática no resulta adecuada para superar el umbral de 50 % de eficiencia en el caso de instancias como M187x814 con 48 cores, lo cual sugiere una baja utilización de los recursos hardware disponibles. Por su parte, el empleo de políticas de planificación dinámica da lugar a una reducción considerable en la sobrecarga de la versión estática, reduciendo los tiempos de espera a nivel de hilos hasta un 33,6 % (para la instancia M355x1263) y los de sincronización inter-proceso hasta un 45,4 % (M187x814). Esto implica que, para el diseño original de MO-SFLA, la planificación dinámica consigue el



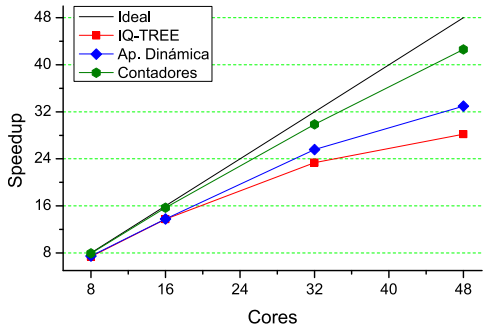
(a) M88x3329



(b) M187x814



(c) M260x1781



(d) M355x1263

Figura 1. Comparativas de Rendimiento Paralelo (MO-SFLA Original Dinámico, MO-SFLA Basado en Contadores e IQ-TREE)

rendimiento paralelo más satisfactorio en la comparativa, con eficiencias medias de 90,4 % (8 cores), 83,8 % (16 cores), 73,1 % (32 cores) y 62,7 % (48 cores). En esta comparativa, la política guiada representa un escenario intermedio, con eficiencias medias de 88,5 % (8 cores), 82,3 % (16 cores), 70,0 % (32 cores) y 58,6 % (48 cores). Con respecto a la versión estática, la planificación guiada reduce hasta un 13,7 % (M355x1263) los tiempos de espera entre hilos, así como hasta un 33,3 % (M187x814) la sincronización entre procesos. Sin embargo, estos resultados no igualan la calidad de la política dinámica, la cual representa la aproximación más adecuada en el contexto del diseño original de MO-SFLA.

Considerando estos resultados, hemos aplicado la política dinámica a su vez en el diseño basado en contadores de MO-SFLA. Los *speedups* y eficiencias obtenidos son mostrados en las columnas 8, 9 de la Tabla II. Conforme a estos resultados, el mecanismo alternativo estudiado consigue mejorar de manera significativa la escalabilidad paralela del diseño original, consiguiendo factores de aceleración medios de 7,8x (8 cores), 15,1x (16 cores), 28,7x (32 cores) y 40,7x (48 cores). A través de un mayor equilibrado de tareas en el bucle paralelo de procesamiento de memeplexes, esta alternativa permite reducir los tiempos de espera entre hilos hasta un 72,1 % (M88x3329), dando lugar a una reducción de hasta 92,0 % (M187x814) en los tiempos de sincronización entre procesos. Como resultado, el diseño basado en contadores obtiene una eficiencia media de 84,7 % para 48 cores, representando una mejora notable con respecto al 62,7 % de la versión original dinámica.

Para validar la bondad de estos resultados, la Figura 1 presenta una comparativa de rendimiento paralelo entre el diseño original con planificación dinámica de MO-SFLA, el diseño basado en contadores, y la herramienta filogenética paralela IQ-TREE (la cual integra técnicas MPI+OpenMP) [3]. Estos resultados dan cuenta de que el diseño original dinámico obtiene una mejoría con respecto a IQ-TREE a partir de escenarios de 32 / 48 cores para M187x814, así como en M260x1781 y, especialmente, M355x1263. Por su parte, el diseño basado en contadores presenta las mejores propiedades de escalabilidad de la comparativa, mejorando de manera significativa los resultados de IQ-TREE en todos los escenarios de evaluación considerados. Todo ello redunda en la relevancia de la estrategia adoptada para minimizar problemas de balanceo de carga en el diseño algorítmico de MO-SFLA.

IV-B. Resultados Biológicos

A continuación abordamos la validación de calidad de solución del método, usando resultados medianos de 31 ejecuciones independientes por instancia. En primer lugar, examinaremos el rendimiento multiobjetivo mediante el empleo de la métrica de hipervolumen (usando los puntos de referencia definidos en [6]). En este contexto, es preciso señalar que, para este problema, no se observaron diferencias estadísticamente significativas entre las muestras de hipervolumen de la versión original de MO-SFLA y la versión basada en contadores, de acuerdo a los tests de ANOVA / Wilcoxon-Mann-Whitney [16]. La Tabla III introduce los valores medianos de hipervolumen observados, en comparación a los obtenidos por el algoritmo estándar NSGA-II [9]. MO-SFLA es capaz de obtener valores de hipervolumen en el rango 75,39 % - 81,99 %, en comparación a los hipervolúmenes de 74,95 % -

Tabla III
Comparativas Multiobjetivo: Hipervolumen

Instancia	MO-SFLA	NSGA-II
M88x3329	**75,39±0,04**	74,95±0,12
M187x814	**77,58±0,58**	76,20±0,58
M260x1781	**80,71±0,23**	77,46±0,91
M355x1263	**81,99±0,19**	80,52±0,22

Tabla IV
Comparativas Filogenéticas: Parsimonia

Instancia	MO-SFLA	TNT
M88x3329	**33490**	**33490**
M187x814	**29847**	**29847**
M260x1781	**43529**	**43529**
M355x1263	**54823**	**54823**

Tabla V
Comparativas Filogenéticas: Verosimilitud

Instancia	MO-SFLA	IQ-TREE	GARLI
M88x3329	**-149094,84**	**-149094,84**	-149111,00
M187x814	**-133871,90**	-133871,96	-133876,72
M260x1781	**-163895,79**	-163899,10	-163982,64
M355x1263	**-231186,34**	-231301,11	-231859,24

80,52 % mostrados por NSGA-II. Estos resultados sugieren que MO-SFLA da lugar a la obtención de frentes de Pareto de calidad significativa en todas las instancias analizadas.

Las Tablas IV y V introducen comparativas de calidad biológica con otros métodos filogenéticos del estado del arte. En la Tabla IV, comparamos dicha calidad desde la perspectiva de parsimonia con respecto a la herramienta heurística TNT [17]. Según este criterio, el método multiobjetivo MO-SFLA es capaz de mantener, en todas las instancias analizadas, la calidad filogenética obtenida por TNT. Por su parte, la Tabla V presenta una comparativa de resultados de verosimilitud con IQ-TREE [3] y el algoritmo evolutivo GARLI [4]. Conforme a los resultados obtenidos, MO-SFLA da lugar a una mejora en verosimilitud en las instancias con mayor número de secuencias, confirmando la bondad de las soluciones generadas.

V. Conclusiones

Este trabajo ha versado sobre la evaluación de diseños paralelos y políticas de planificación para solucionar problemas de balanceo de carga en métodos metaheurísticos. Usando como caso de estudio una metaheurística multiobjetivo para reconstrucción filogenética, MO-SFLA, hemos identificado las distintas fuentes de desequilibrio de carga presentes en su diseño algorítmico. Para afrontarlos, hemos considerado el uso de distintas políticas de planificación del estándar OpenMP (estática, dinámica y guiada), así como una aproximación algorítmica alternativa basada en el empleo de contadores para equilibrar la carga de trabajo de los hilos de ejecución.

Hemos evaluado experimentalmente las técnicas planteadas en un clúster multicore compuesto por 48 núcleos de procesamiento, empleando para ello cuatro instancias del problema basadas en datos de secuencias de proteínas. Para el diseño original, la aproximación dinámica dio lugar a un rendimiento más satisfactorio que el resto de políticas, mejorando la eficiencia media del algoritmo de 55,2 % (versión estática) a 62,7 % para 48 cores. Además, la introducción del mecanismo alternativo de balanceo basado en contadores permitió ir un paso más allá en la explotación de los recursos hardware, demostrando eficiencias hasta de 88,8 % sin afectar significativamente a la calidad de los frentes de Pareto generados.

Como trabajo futuro, pretendemos afrontar la explotación de recursos de computación heterogénea para acelerar el método propuesto, explorando alternativas dependientes e independientes el problema. Se estudiarán a su vez estrategias adaptativas para mejorar las capacidades de búsqueda en espacios de decisión difíciles de abordar, así como la evaluación del diseño en otros problemas bioinformáticos.

Referencias

[1] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.

[2] P. Lemey, M. Salemi, and A.-M. Vandamme, *The Phylogenetic Handbook: a Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge: Cambridge Univ. Press, 2009.

[3] L. Nguyen, H. Schmidt, A. Haeseler, and B. Minh, "IQ-TREE: A fast and effective stochastic algorithm for estimating maximum likelihood phylogenies," *Mol. Biol. Evol.*, vol. 32, no. 1, pp. 268–274, 2015.

[4] A. L. Bazinet, D. J. Zwickl, and M. P. Cummings, "A Gateway for Phylogenetic Analysis Powered by Grid Computing Featuring GARLI 2.0," *Systematic Biology*, vol. 63, no. 5, pp. 812–818, 2014.

[5] W. Cancino, L. Jourdan, E.-G. Talbi, and A. C. B. Delbem, "Parallel Multi-Objective Approaches for Inferring Phylogenies," in *Proc. of EVOBIO'2010*, ser. LNCS, vol. 6023. Springer, 2010, pp. 26–37.

[6] S. Santander-Jiménez, M. A. Vega-Rodríguez, and L. Sousa, "Multi-objective Frog-Leaping Optimization for the Study of Ancestral Relationships in Protein Data," *IEEE Trans. Evol. Comput.*, pp. 1–14 (DOI: 10.1109/TEVC.2017.2 774 599), 2018.

[7] H. Matsuda, H. Yamashita, and Y. Kaneda, "Molecular Phylogenetic Analysis using both DNA and Amino Acid Sequence Data and Its Parallelization," *Genome Informatics*, vol. 5, pp. 120–129, 1994.

[8] A. Sarkheyli, A. M. Zain, and S. Sharif, "The role of basic, modified and hybrid shuffled frog leaping algorithm on optimization problems: a review," *Soft Computing*, vol. 19, no. 7, pp. 2011–2038, 2015.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multi–Objective Genetic Algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[10] W. Gropp, W. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface. 3rd edition*. Cambridge, MA, USA: The MIT Press, 2014.

[11] R. van der Pas, E. Stotzer, and C. Terboven, *Using OpenMP - The Next Step*. Cambridge, MA, USA: The MIT Press, 2017.

[12] I. Morgenstern *et al.*, "A molecular phylogeny of thermophilic fungi," *Fungal Biology*, vol. 116, no. 4, pp. 489–502, 2012.

[13] A. Kovalchuk, A. Kohler, F. Martin, and F. O. Asiegbu, "Diversity and evolution of ABC proteins in mycorrhiza-forming fungi," *BMC Evolutionary Biology*, vol. 15, no. 249, pp. 1–19, 2015.

[14] R. Stracke *et al.*, "Genome-wide identification and characterisation of R2R3-MYB genes in sugar beet (Beta vulgaris)," *BMC Plant Biology*, vol. 14, no. 249, pp. 1–17, 2014.

[15] P. J. Dias and I. Sá-Correia, "The drug:H+ antiporters of family 2 (DHA2), siderophore transporters (ARN) and glutathione:h+antiporters (GEX) have a common evolutionary origin in hemiascomycete yeasts," *BMC Genomics*, vol. 14, no. 901, pp. 1–22, 2013.

[16] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures. 5th edition*. NY, USA: Chapman & Hall/CRC, 2011.

[17] P. A. Goloboff and S. A. Catalano, "TNT version 1.5, including a full implementation of phylogenetic morphometrics," *Cladistics*, vol. 32, no. 3, pp. 221–238, 2016.

Developing Genetic Algorithms using Different MapReduce Frameworks: MPI vs. Hadoop*

*Note: The full contents of this paper have been published in the volume *Lecture Notes in Artificial Intelligence 11160* (LNAI 11160)

Carolina Salto
Facultad de Ingeniería
Universidad Nacional de La Pampa
Argentina
minettig@ing.unlpam.edu.ar

Gabriela Minetti
CONICET
Argentina
saltoc@ing.unlpam.edu.ar

Enrique Alba, Gabriel Luque
Departamento de Lenguajes
Universidad de Málaga
Málaga, Spain
{eat,gabriel}@lcc.uma.es

***Abstract*—MapReduce is a quite popular paradigm, which allows to no specialized users to use large parallel computational platforms in a transparent way. Hadoop is the most used implementation of this paradigm, and in fact, for a large amount of users the word Hadoop and MapReduce are interchangable. But, there are other frameworks that implement this programming paradigm, such as MapReduce-MPI. Since, optimization techniques can be greatly beneficiary of this kind of data-intensive computing modeling, in this paper, we analyze the performance effect of developing genetic algorithms (GA) using different frameworks of MapReduce (MRGA). In particular, we implement MRGA using Hadoop and MR-MPI frameworks. We analyze and compare both implementations considering relevant aspects such as efficiency and scalability to solve a large dimension problem. The results show a similar efficiency level between the algorithms but Hadoop presents a better scalability.**

***Index Terms*—**