Estudio de escalabilidad de implementaciones multiobjetivo en problemas con muchos objetivos

Aurora Ramírez, Rafael Barbudo, José Raúl Romero, Sebastián Ventura
Dpto. Informática y Análisis Numérico, Universidad de Córdoba
{aramirez, rbarbudo, jrromero, sventura}@uco.es

***Resumen*—Los problemas de optimización de varios objetivos aparecen con frecuencia en aplicaciones reales, lo que ha llevado a la adaptación de la mayoría de las metaheurísticas existentes. El creciente interés en los problemas que presentan muchos objetivos también ha propiciado la aparición de nuevos métodos especializados. Para conseguir una mayor adopción de estas técnicas por parte de usuarios no familiarizados con el uso de metaheurísticas, es necesario que los nuevos desarrollos sean incluidos en librerías software que faciliten su configuración y adaptación. No obstante, la librería escogida puede repercutir en el rendimiento computacional, pues cada una sigue unos principios de diseño diferentes. Este trabajo analiza algunas de las implementaciones multiobjetivo disponibles en seis librerías con el propósito de estudiar su comportamiento frente a configuraciones de creciente complejidad y que incluyen problemas de hasta 50 objetivos. La experimentación realizada confirma que distintas implementaciones de un mismo algoritmo pueden presentar diferencias considerables en cuanto al tiempo de ejecución y la memoria consumida.**

***Index Terms*—Optimización de muchos objetivos, metaheurísticas, librerías software**

I. Introducción

Los problemas multiobjetivo han sido objeto frecuente de estudio en el campo de las metaheurísticas, pues son habitualmente aplicables en escenarios reales [1]. Este tipo de problemas requiere la definición de un conjunto de variables cuyo valor ha de determinarse, un conjunto de restricciones a cumplir y, al menos, dos objetivos independientes a optimizar [2]. La existencia de varios objetivos, a menudo contrapuestos, implica que la solución a un problema multiobjetivo no es única, sino que debe encontrarse un conjunto de soluciones donde cada una alcanza un compromiso distinto entre los objetivos. El estudio de enfoques bioinspirados para resolver problemas multiobjetivo ha dado lugar a algoritmos ya clásicos como NSGA-II y SPEA2, de los que es posible encontrar implementaciones en distintos lenguajes de programación [3].

Recientemente, la investigación en este campo se ha centrado en abordar la resolución de problemas con un número elevado de objetivos. Según la literatura, los denominados "problemas de muchos objetivos" (en inglés, *many-objective optimisation problems*) son aquellos que presentan cuatro o más objetivos [4]. Su resolución presenta nuevos retos, tales como el crecimiento exponencial del número de soluciones no dominadas o la ineficacia de operadores genéticos y técnicas de preservación de la diversidad [5]. De hecho, existen estudios que demuestran el bajo rendimiento de algoritmos clásicos a la hora de resolver este tipo de problemas, así como el incremento en el coste computacional [6]. Estos hechos han propiciado el desarrollo de métodos especializados [7], así como la realización de estudios comparativos [8].

A pesar de que algunas librerías software están incorporando algunos de los algoritmos para optimización de muchos objetivos que van adquiriendo mayor relevancia, lo cierto es que aún no existe un gran número de implementaciones disponibles. Este hecho no solo dificulta su uso con fines comparativos, sino que limita su aplicación por parte de usuarios con menos experiencia en el campo de las metaheurísticas. Por otro lado, características como el lenguaje de programación y el diseño estructural de estas librerías podría suponer ciertas diferencias en el uso de los recursos computacionales, como el tiempo de ejecución y la memoria, que también deben considerarse ante la creciente complejidad tanto de los problemas como de los algoritmos.

En este contexto, este trabajo presenta un estudio preliminar del rendimiento de algoritmos multiobjetivo implementados en diferentes librerías software. Tras realizar un análisis de las implementaciones disponibles, se han elegido seis librerías Java: ECJ, EvA, JCLEC-MO, jMetal, MOEA Framework y Opt4J. El objetivo de la comparación es por tanto analizar el rendimiento desde una perspectiva puramente computacional, realizando mediciones del tiempo de ejecución y la memoria RAM consumida. Para ello se han seleccionado varios algoritmos y problemas de entre los disponibles en las librerías. Además, se ha estudiado la escalabilidad de las distintas implementaciones de NSGA-II respecto al número de individuos, generaciones y objetivos. Los resultados muestran que existen pequeñas diferencias entre las implementaciones cuando se consideran parámetros estándar, y que éstas se incrementan a medida que se consideran valores más extremos.

El resto del artículo se estructura como sigue. La sección II describe los algoritmos y problemas más utilizados en optimización multiobjetivo, mientras que la sección III recopila las principales librerías software que los implementan. La sección IV detalla la metodología diseñada para el estudio comparativo, cuyos resultados son analizados en la sección V. Finalmente, la sección VI presenta las conclusiones.

II. Optimización con múltiples objetivos

Esta sección introduce los principales tipos de algoritmos multiobjetivo existentes, destacando aquellos especializados en

Trabajo financiado por los ministerios de Economía (TIN2017-83445-P) y de Educación (FPU13/01466), y fondos FEDER.

la optimización de muchos objetivos. Después se describen algunos de los problemas comúnmente utilizados para evaluar el rendimiento de estos algoritmos.

II-A. *Familias de algoritmos bioinspirados*

Desde la publicación de VEGA, considerado el primer algoritmo evolutivo multiobjetivo [2], la aparición de nuevos algoritmos o mejoras sobre los ya propuestos ha sido constante [9]. En los inicios del área, los algoritmos iban siendo clasificados en generaciones [10]. Así, la conocida como *primera generación* se basa fuertemente en el concepto de dominancia y el uso de estrategias de nichos. La incorporación de mecanismos de elitismo es la principal característica de la *segunda generación*, a la que pertenecen SPEA2 y NSGA-II.

En el ámbito de la optimización de muchos objetivos, los algoritmos se clasifican atendiendo al enfoque que siguen para abordar los retos mencionados anteriormente [7]. En primer lugar, los algoritmos basados en descomposición definen un conjunto de pesos para establecer diferentes direcciones de búsqueda. MOEA/D es el algoritmo más conocido dentro de esta familia. Por otro lado, los algoritmos basados en indicadores utilizan una medida de rendimiento, como el hipervolumen, para guiar la búsqueda. IBEA fue el primer algoritmo de esta familia, donde también destacan HypE o SMS-EMOA. Otro conjunto de algoritmos se caracterizan por la adopción de criterios de dominancia relajados, como la ϵ-dominancia utilizada tanto en algoritmos evolutivos (ϵ-MOEA) como de optimización de partículas (OMOPSO). El uso de un conjunto de puntos de referencia es una técnica habitual para garantizar la diversidad en espacios de objetivos multidimensionales. En este grupo se encuentran algoritmos como NSGA-III y RVEA, entre otros. Finalmente, los métodos basados en preferencias, como WASF-GA o PICEA-g, permiten restringir la búsqueda a una zona de interés para el usuario. Los detalles de estos algoritmos pueden encontrarse en artículos de referencia como [7], [9].

II-B. *Problemas para el estudio del rendimiento*

A la hora de comparar el rendimiento de distintos algoritmos, es frecuente utilizar problemas tipo o *benchmarks*. Por ejemplo, problemas tan conocidos como el de la mochila (*Knapsack Problem*, KP) o el viajante de comercio (*Travelling Salesman Problem*, TSP) han sido extendidos respecto a sus formulaciones originales para un único objetivo. Otro *benchmark* a destacar es LOTZ (*Leading Ones, Trailing Zeros*), un problema simple de codificación binaria con dos objetivos contrapuestos: maximizar el número de unos al comienzo del genotipo y maximizar el número de ceros al final.

Para codificación real existen varias familias de problemas cuyos frentes de Pareto óptimo presentan diferentes propiedades (discontinuidad en el frente, proximidad de óptimos locales, etc.). Además, algunas permiten establecer el número de objetivos que se desea optimizar, por lo que son especialmente útiles para estudiar la escalabilidad. Entre ellas cabe destacar ZDT, un conjunto de seis problemas con dos objetivos cada uno [11], y DTLZ, un conjunto de nueve problemas cuyo número de objetivos es adaptable [12].

III. Implementaciones disponibles en librerías

La popularidad de las técnicas metaheurísticas ha llevado a la creación de librerías y *frameworks* software que dan soporte a la investigación y facilitan su aplicación práctica [3]. Este tipo de herramientas no solo contemplan la configuración y ejecución de los algoritmos y sus componentes, sino que también permiten adaptarlos a nuevos problemas.

La Tabla I muestra un listado de algunas librerías que incluyen soporte para la optimización multiobjetivo, indicando la última versión y su año de lanzamiento, el lenguaje de programación y los algoritmos que proporcionan[1]. Como puede observarse, entre ellas hay herramientas de propósito general muy conocidas, como HeuristicLab y ECJ. A pesar de su madurez, el número de implementaciones multiobjetivo que ofrecen se limita a los algoritmos evolutivos más populares. Otras librerías Java son Opt4J y EvA, las cuales disponen de mayor número y variedad de algoritmos. Más recientemente, han ido apareciendo librerías en Python, como DEAP (especializada en algoritmos distribuidos) y PyGMO (especializada en algoritmos paralelos), si bien el número de algoritmos multiobjetivo en ambas es aún reducido.

ParadisEO-MOEO y JCLEC-MO se asemejan en cuanto a su estructura, pues ambas librerías disponen de un módulo específico para optimización multiobjetivo. Por un lado, ParadisEO ha sido posiblemente la librería más popular de entre las desarrolladas en C++, si bien lleva tiempo sin ser actualizada. Por otro lado, JCLEC-MO ofrece un mayor conjunto de algoritmos, incluyendo algunos especialmente indicados para problemas de muchos objetivos.

Finalmente, también es posible encontrar librerías especializadas en optimización multiobjetivo. La primera de ellas fue PISA, si bien se trata de un proyecto ya abandonado. jMetal y MOEA Framework, ambas desarrolladas en Java, destacan por sus continuas actualizaciones y su amplio catálogo de algoritmos. Dos proyectos muy recientes y activos son PlatEMO y Platypus, desarrollados en Matlab y Python, respectivamente.

IV. Metodología experimental

Para el estudio comparativo de implementaciones multiobjetivo se han seleccionado las seis librerías Java detalladas en la Tabla I. La elección de un único lenguaje de programación facilita la comparación pues evita posibles diferencias debidas al proceso de compilación. Además, estas librerías cuentan con, al menos, dos algoritmos en común, lo cual reduce la posibilidad de alterar el rendimiento debido a la diferencia de implementación entre desarrollos propios y nativos. Tan solo ha sido necesario implementar algunos problemas para disponer de un conjunto lo suficientemente variado, así como operadores genéticos estándar.

Tras analizar las herramientas, se han planteado dos experimentos. El primero tiene como objetivo estudiar distintas

[1]Las posibles variantes de dichos algoritmos no son mostradas por motivos de espacio. El lector es remitido a la documentación de cada herramienta.

Tabla I
Librerías y algoritmos disponibles para optimización multiobjetivo

Librería	Versión	Año	Lenguaje	Principales algoritmos
DEAP	1.2.2	2017	Python	CMAES-MO, NSGA-II, SPEA2
ECJ	26	2018	Java	NSGA-II, NSGA-III, SPEA2
EvA	2.2	2015	Java	CMAES-MO, MOGA, NSGA, NSGA-II, PESA, PESA2, SPEA, SPEA2
HeuristicLab	3.3.15	2018	C#	CMAES-MO, NSGA-II
JCLEC-MO	1.0	2018	Java	ϵ-MOEA, GrEA, HypE, IBEA, MOCHC, MOEA/D, OMOPSO, NSGA-II, NSGA-III, PAES, PAR, RVEA, SMS-EMOA, SMPSO, SPEA2
jMetal	5.4	2017	Java	AbYSS, CellDE, dMOPSO, GDE3, GWASF-GA, IBEA, MOCell, MOCHC, MOEA/D, MOEADD, MOMBI, MOMBI-II, NSGA-II, NSGA-III, OMOPSO, PAES, PESA2, SMS-EMOA, SMPSO, SPEA2, WASF-GA
MOEA Framework	2.12	2017	Java	CMAES-MO, DBEA, ϵ-MOEA, GDE3, IBEA, MOEA/D, MSOPS, NSGA-II, NSGA-III, OMOPSO, PAES, PESA2, RVEA, SMPSO, SMS-EMOA, SPEA2, VEGA
Opt4J	3.1.4	2015	Java	NSGA-II, OMOPSO, SMS-EMOA, SPEA2
PaGMO/PyGMO	2.8	2018	C++/Python	MOEA/D, NSGA-II
ParadisEO-MOEO	2.0.1	2012	C++	MOGA, IBEA, NSGA, NSGA-II, SPEA2
PlatEMO	1.5	2017	Matlab	AGE-II, dMOPSO, ϵ-MOEA, GDE3, GrEA, HypE, I-DBEA, IBEA, LMEA, MOEA/D, MOMBI-II, MOPSO, MSOPS-II, NSGA-II, NSGA-III, PICEA-g, PSEA-II, RPEA, RVEA, SMPSO, SMS-EMOA, SPEA2, Two_Arch2
Platypus	1.0.2	2018	Python	ϵ-MOEA, GDE3, IBEA, MOEA/D, NSGA-II, NSGA-III, OMOPSO, SMPSO, SPEA2.
PISA	-	2009	C	ϵ-MOEA, FEMO, HypE, IBEA, MSOPS, NSGA-II, SEMO2, SHV, SPAM, SPEA2

combinaciones de algoritmos y problemas, donde estos últimos varían en cuanto al tipo de codificación y al número de objetivos. La Tabla II muestra la configuración propuesta, donde el número entre paréntesis representa el número de problema dentro de la familia correspondiente. Para las tres librerías que disponen de un número menor de algoritmos se ha seleccionado un número mayor de problemas. En cuanto a las tres últimas librerías, se han seleccionado algoritmos representativos de distintas familias. En el caso de MOEA/D, únicamente se han considerado problemas de optimización real debido a que es el tipo de problema asumido en jMetal. Igualmente, la generación automática de los pesos se limita a problemas biobjetivo. Por ser un algoritmo de partículas, OMOPSO también queda restringido a problemas de optimización real. La Tabla III recoge los valores de los parámetros, tanto generales como específicos de cada algoritmo.

Tabla II
Algoritmos y problemas seleccionados

Algoritmo	Problema		
	2 obj.	4 obj.	6 obj.
ECJ, EvA, Opt4J			
NSGA-II	LOTZ, ZDT (1, 4)	KP	DTLZ (1, 2, 4)
SPEA2	LOTZ, ZDT (1, 4)	KP	DTLZ (1, 2, 4)
JCLEC-MO, jMetal, MOEA Framework			
IBEA	LOTZ	KP	DTLZ (1, 2)
MOEA/D	ZDT (1, 4)		DTLZ (1, 2)
OMOPSO	ZDT (1, 4)		DTLZ (1, 2)
NSGA-II	LOTZ	KP	DTLZ (1, 2)

El segundo experimento pretende analizar la escalabilidad de las implementaciones en base a tres factores: el tamaño de la población (100, 500, 1000), el número de generaciones (100, 500, 1000) y el número de objetivos (2, 5, 10, 25, 50). Estas configuraciones serán identificadas como Px-Gy-Oz, donde P representa el tamaño de la población, G, las generaciones y O, los objetivos. El algoritmo escogido es NSGA-II, ya que está presente en todas las librerías y no presenta parámetros adicionales. Puesto que se desea variar el número de objetivos, se ha seleccionado DTLZ1 como problema a resolver. Los operadores genéticos y sus probabilidades se configuran de acuerdo a los valores mostrados en la Tabla III.

Tabla III
Configuración de parámetros

Parámetro	Valor
Parámetros comunes	
Tamaño de la población	100
Número de generaciones	100
Probabilidad de cruce	0,9
Probabilidad de mutación	0,1 (1/longitud_genotipo)
Operador de cruce	*1-Point* (binario), SBX (real)
Operador de mutación	*Bit flip* (binario), *Polynomial* (real)
SPEA2	
Tamaño de la población (P)	50
Tamaño del archivo (A)	50
Selector de padres	Torneo binario
Parámetro k	$\sqrt[2]{P+A}$
MOEA/D	
Tamaño de vecindad (τ)	10
Máx. Núm. Reemplazos (nr)	2
Modo de evaluación	*Tchebycheff*
Generación de pesos	Uniforme
IBEA	
Selector de padres	Torneo binario
Indicator para evaluación	Hipervolumen
Parámetro κ	0,05
Parámetro ρ	2
OMOPSO	
Tamaño del archivo	100
Parámetro ϵ	0,0075

En ambos experimentos se ha utilizado Syrupy[2] como herramienta *profiler* para medir el tiempo de ejecución y el consumo de memoria a intervalos regulares. Dichos intervalos se han establecido en base a ejecuciones previas con el fin de

[2]https://github.com/jeetsukumaran/Syrupy

obtener un mínimo de 30 muestras para cada configuración. La experimentación se ha realizado en una máquina con Debian 8, ocho núcleos Intel Core i7-2600, CPU a 3,4 GHz y 16 GB de memoria RAM. En el caso del primer experimento, cada configuración ha sido ejecutada cinco veces en un único núcleo para evitar posibles interferencias. Además, se ha alternado el orden de ejecución de cada librería para mantener condiciones similares respecto al arranque de la JVM. En el segundo experimento, cada configuración ha sido lanzada seis veces utilizando dos núcleos para que el experimento no tuviese una duración excesiva. No obstante, las configuraciones también se planificaron de forma que no siempre las mismas librerías estuviesen ejecutando en paralelo a fin de reducir el sesgo lo más posible. Tras obtener el tiempo medio de ejecución de cada configuración, se ha aplicado el test de Friedman para analizar si existen diferencias significativas entre aquellas librerías para las que se ejecutaron los mismos algoritmos y problemas. Es decir, para el primer experimento se analizan dos grupos por separado, uno formado por ECJ, EvA y Opt4J, y otro formado por JCLEC-MO, jMetal y MOEA Framework. En caso de encontrar diferencias significativas, se aplica el post-procedimiento de Holm. Además, se ha ejecutado el test Cliff's Delta para analizar el tamaño del efecto, esto es, la magnitud de la diferencia observada.

V. Resultados y discusión

En esta sección se presentan los resultados obtenidos para cada uno de los experimentos planteados. Por motivos de espacio, solo los resultados más relevantes son discutidos.

V-A. Comparación de algoritmos y problemas

Las Tablas IV y V muestran los tiempos medios de ejecución y la desviación estándar para los dos grupos de librerías. La ejecución más rápida (0,24 s) se corresponde con la resolución de DTLZ2 con MOEA/D en jMetal. Por el contrario, el problema de la mochila con NSGA-II en ECJ es la combinación que más tiempo ha necesitado (4,88 s).

En el primer grupo de librerías (Tabla IV), ECJ es la que requiere menos tiempo, empleando más de un segundo en solo cuatro configuraciones. EvA es la librería que reporta tiempos de ejecución más altos, siempre por encima de un segundo, mientras que Opt4J mantiene tiempos más estables, entre 0,5 y 1,5 s. Para confirmar si estas diferencias son significativas, se ha aplicado el test de Friedman, donde la hipótesis nula establece que todas las librerías tienen un rendimiento similar. El valor del estadístico de *Iman and Daveport*, z, es 30,71, mientras que el valor crítico de acuerdo a una distribución F con 2 y 30 grados de libertad es igual a 5,39 ($\alpha = 0,01$). Dado que z es mayor que el valor crítico, la hipótesis nula puede ser rechazada. El test de Holm devuelve un $p - value$ igual a 0,05, lo que confirma que ECJ es significativamente superior a EvA, pero equivalente a Opt4J. En cuanto al efecto del tamaño, el test Cliff's Delta indica que la diferencia en tiempo entre las tres librerías es siempre amplia (*large*).

En el segundo grupo de librerías (JCLEC-MO, jMetal y MOEA Framework), IBEA es el algoritmo más costoso, requiriendo varios segundos de ejecución. La diferencia con respecto al resto de algoritmos es más evidente en JCLEC-MO y jMetal, especialmente cuando se resuelven problemas combinatorios. Para este conjunto de librerías, el test de Friedman no encuentra diferencias significativas. De hecho, el test Cliff's Delta siempre cataloga las diferencias entre cada pareja de librerías como insignificante (*negligible*), salvo entre jMetal y JCLEC-MO, que es pequeña (*small*).

Dado que todas estas configuraciones son simples y se ejecutan en pocos segundos, tan solo se pueden realizar algunas apreciaciones respecto a la memoria RAM consumida. Las seis librerías Java analizadas consumen un mínimo de 12.000 KB en cada ejecución. EvA y Opt4J son las dos librerías que muestran un comportamiento más estable, con un máximo de 56.459 KB y 126.084 KB de media, respectivamente. Entre el resto, el uso máximo de memoria es más disperso llegando a presentar "picos" de más de 100.000 KB en JCLEC-MO (76.130 KB de media), jMetal (88.326 KB) y MOEA Framerwork (89.005 KB), y más de 500.000 KB en ECJ (149.265 KB) cuando se resuelve el problema de la mochila.

Valorando los resultados del primer experimento en global,

Tabla IV
Tiempo de ejecución en segundos para el primer grupo de librerías (experimento 1)

Configuración	ECJ	EvA	Opt4J
NSGA-II – LOTZ(2)	$0,245 \pm 0,014$	$1,443 \pm 0,037$	$0,773 \pm 0,050$
NSGA-II – ZDT1(2)	$0,299 \pm 0,015$	$1,827 \pm 0,019$	$0,785 \pm 0,056$
NSGA-II – ZDT4(2)	$0,295 \pm 0,017$	$1,537 \pm 0,014$	$0,765 \pm 0,068$
NSGA-II – ZDT6(2)	$0,284 \pm 0,034$	$1,466 \pm 0,023$	$0,758 \pm 0,058$
NSGA-II – KP(4)	$4,879 \pm 0,022$	$1,587 \pm 0,041$	$1,142 \pm 0,076$
NSGA-II – DTLZ1(6)	$0,344 \pm 0,021$	$1,658 \pm 0,043$	$0,896 \pm 0,067$
NSGA-II – DTLZ2(6)	$0,336 \pm 0,022$	$1,803 \pm 0,040$	$0,900 \pm 0,043$
NSGA-II – DTLZ4(6)	$0,351 \pm 0,034$	$1,834 \pm 0,039$	$0,922 \pm 0,110$
SPEA2 – LOTZ(2)	$0,444 \pm 0,017$	$1,445 \pm 0,022$	$0,609 \pm 0,038$
SPEA2 – ZDT1(2)	$0,368 \pm 0,028$	$2,684 \pm 0,032$	$0,999 \pm 0,087$
SPEA2 – ZDT4(2)	$0,291 \pm 0,022$	$1,380 \pm 0,029$	$1,020 \pm 0,060$
SPEA2 – ZDT6(2)	$0,300 \pm 0,021$	$1,385 \pm 0,047$	$0,735 \pm 0,032$
SPEA2 – KP(4)	$3,219 \pm 0,033$	$1,736 \pm 0,105$	$1,173 \pm 0,065$
SPEA2 – DTLZ1(6)	$0,959 \pm 0,050$	$1,646 \pm 0,107$	$1,633 \pm 0,236$
SPEA2 – DTLZ2(6)	$1,442 \pm 0,023$	$2,702 \pm 0,038$	$1,583 \pm 0,070$
SPEA2 – DTLZ4(6)	$1,435 \pm 0,088$	$2,897 \pm 0,059$	$1,699 \pm 0,103$
Friedman (ranking)	**1,250**	2,875	1,875
Holm (α/i)	-	0,025	0,050

Tabla V
Tiempo de ejecución en segundos para el segundo grupo de librerías (experimento 1)

Configuración	JCLEC-MO	jMetal	MOEA Fram.
NSGA-II – LOTZ(2)	$0,410 \pm 0,009$	$0,624 \pm 0,023$	$0,328 \pm 0,018$
NSGA-II – KP(4)	$0,519 \pm 0,027$	$0,558 \pm 0,042$	$0,374 \pm 0,016$
NSGA-II – DTLZ1(6)	$0,537 \pm 0,028$	$0,540 \pm 0,066$	$0,444 \pm 0,011$
NSGA-II – DTLZ2(6)	$0,587 \pm 0,064$	$0,595 \pm 0,055$	$0,389 \pm 0,008$
IBEA – LOTZ(2)	$4,754 \pm 0,037$	$1,288 \pm 0,039$	$0,844 \pm 0,019$
IBEA – KP(4)	$3,012 \pm 0,036$	$1,678 \pm 0,142$	$1,036 \pm 0,029$
IBEA – DTLZ1(6)	$1,829 \pm 0,030$	$2,556 \pm 0,034$	$1,254 \pm 0,011$
IBEA – DTLZ2(6)	$1,843 \pm 0,018$	$2,538 \pm 0,032$	$1,285 \pm 0,007$
MOEA/D – ZDT1(2)	$0,429 \pm 0,018$	$0,337 \pm 0,020$	$0,458 \pm 0,008$
MOEA/D – ZDT4(2)	$0,413 \pm 0,030$	$0,277 \pm 0,010$	$0,414 \pm 0,010$
MOEA/D – DTLZ1(2)	$0,380 \pm 0,022$	$0,260 \pm 0,013$	$0,434 \pm 0,023$
MOEA/D – DTLZ2(2)	$0,492 \pm 0,035$	$0,236 \pm 0,004$	$0,412 \pm 0,016$
OMOPSO – ZDT1(2)	$0,556 \pm 0,034$	$0,412 \pm 0,050$	$1,269 \pm 0,040$
OMOPSO – ZDT4(2)	$0,530 \pm 0,043$	$0,284 \pm 0,010$	$1,117 \pm 0,032$
OMOPSO – DTLZ1(2)	$0,663 \pm 0,009$	$0,484 \pm 0,042$	$1,510 \pm 0,016$
OMOPSO – DTLZ2(2)	$0,715 \pm 0,035$	$1,466 \pm 0,086$	$2,139 \pm 0,072$
Friedman (ranking)	2,125	**1,938**	**1,938**

todas las librerías presentan unos requisitos de tiempo y memoria asumibles ante un escenario de uso normal. Aún así, y a pesar de que las configuraciones probadas son poco exigentes, se perciben ciertas diferencias. En concreto, no todas las librerías responden igual ante la ejecución de un mismo algoritmo en problemas de distinta naturaleza (combinatorios frente a optimización real). En este sentido, los problemas combinatorios son más costosos, aún cuando el número de objetivos es inferior al considerado para los problemas DTLZ. Esto puede deberse a que la evaluación requiere la interpretación del genotipo, frente al uso directo de las variables reales. Por otro lado, la formulación concreta de los diferentes problemas ZDT no parece tener una gran influencia en el tiempo, mientras que sí se observan algunas diferencias respecto a los problemas DTLZ (véase por ejemplo OMOPSO en jMetal y MOEA Framework, o SPEA2 en ECJ y EvA). Respecto a los algoritmos, NSGA-II suele ser el más rápido en la mayoría de librerías. La única excepción parece ser la implementación de MOEA/D en JCLEC-MO y jMetal, si bien hay que considerar que este algoritmo solo ha sido probado con problemas biobjetivo de codificación real.

V-B. Test de escalabilidad

El segundo experimento, donde se ejecutan configuraciones con hasta 50 objetivos, permite detectar mayores diferencias entre las librerías. La Tabla VI muestra los tiempos mínimos y máximos medios, con su desviación estándar, para cada tamaño de población. También se incluye el ranking obtenido con el test de Friedman. Cabe señalar que los tiempos mínimos y máximos corresponden, por lo general, con las configuraciones P100-G100-O2 y P1000-G5000-050. Las excepciones son algunos tiempos mínimos en Opt4J, JCLEC-MO y jMetal, que se consiguen para cinco objetivos. No obstante, los valores son muy similares a los obtenidos para dos objetivos, por lo que pueden deberse a pequeñas fluctuaciones en la carga de la máquina. En base a estos resultados, ECJ es la librería que mejor responde ante configuraciones más exigentes, seguida de MOEA Framework, si bien el test de Holm no indica la existencia de diferencias significativas entre ambas. Les siguen jMetal y JCLEC-MO, si bien cabe señalar que JCLEC-MO responde algo mejor que jMetal ante el aumento del número de objetivos. Lo mismo ocurre si se comparan las dos librerías más lentas según este estudio, pues aunque Opt4j y EvA reportan tiempos mínimos similares, Opt4J escala mejor.

Por otro lado, los datos extraídos muestran que el incremento en el tiempo de ejecución no depende únicamente del número de generaciones. Por ejemplo, configuraciones con 1000 generaciones y dos objetivos pueden requerir menos tiempo de cómputo que configuraciones con 500 generaciones y 50 objetivos. Aunque hay algunas excepciones, sí se aprecia un incremento lineal cuando dos de los parámetros son fijados y el otro aumenta progresivamente. Los dos parámetros más determinantes son el número de generaciones y el número de objetivos, especialmente cuando se consideran valores superiores a 500 y 10, respectivamente. A modo de ejemplo, la Figura 1 muestra el tiempo medio de ejecución para las distintas combinaciones de objetivos y generaciones, fijando el tamaño de población en 1000 (el valor más alto probado). Para facilitar la comparación entre librerías, se ha mantenido la misma escala respecto al tiempo. A excepción de EvA, las librerías presentan una tendencia similar y no suelen requerir más de 30 minutos por ejecución.

Con respecto al consumo de memoria, el alto número de combinaciones probadas permite obtener mayores diferencias. A modo de resumen, la Tabla VII recoge el valor mínimo y máximo de memoria registrado para cada librería. Como puede verse, no existen grandes diferencias en cuanto a los valores mínimos, pero sí en los máximos alcanzados. También se muestra el porcentaje de veces que cada librería obtiene el mínimo y máximo global para las 60 combinaciones probadas. En este caso, ECJ y Opt4J son las librerías que menor y mayor memoria consumen, respectivamente.

Este test de escalabilidad confirma que las implementaciones Java actuales pueden sufrir una degradación considerable en su rendimiento si se las somete a configuraciones extremas. Aunque quizás no sean los valores más habituales en la práctica, también debe considerarse que NSGA-II no es de los algoritmos más costosos y que el problema es sintético. En una aplicación real, donde fuese necesario ejecutar algoritmos específicos para problemas de muchos objetivos cuya evaluación fuese costosa, la elección de una implementación u otra podría ser determinante. Entre las librerías analizadas, EvA y Opt4J presentan mayores problemas para controlar el tiempo y la memoria, respectivamente. Por el contrario, ECJ y MOEA Framework presentan el mejor rendimiento considerando ambos factores a la vez, ya que tanto jMetal como JCLEC-MO experimentan mayores oscilaciones de memoria.

VI. Conclusiones

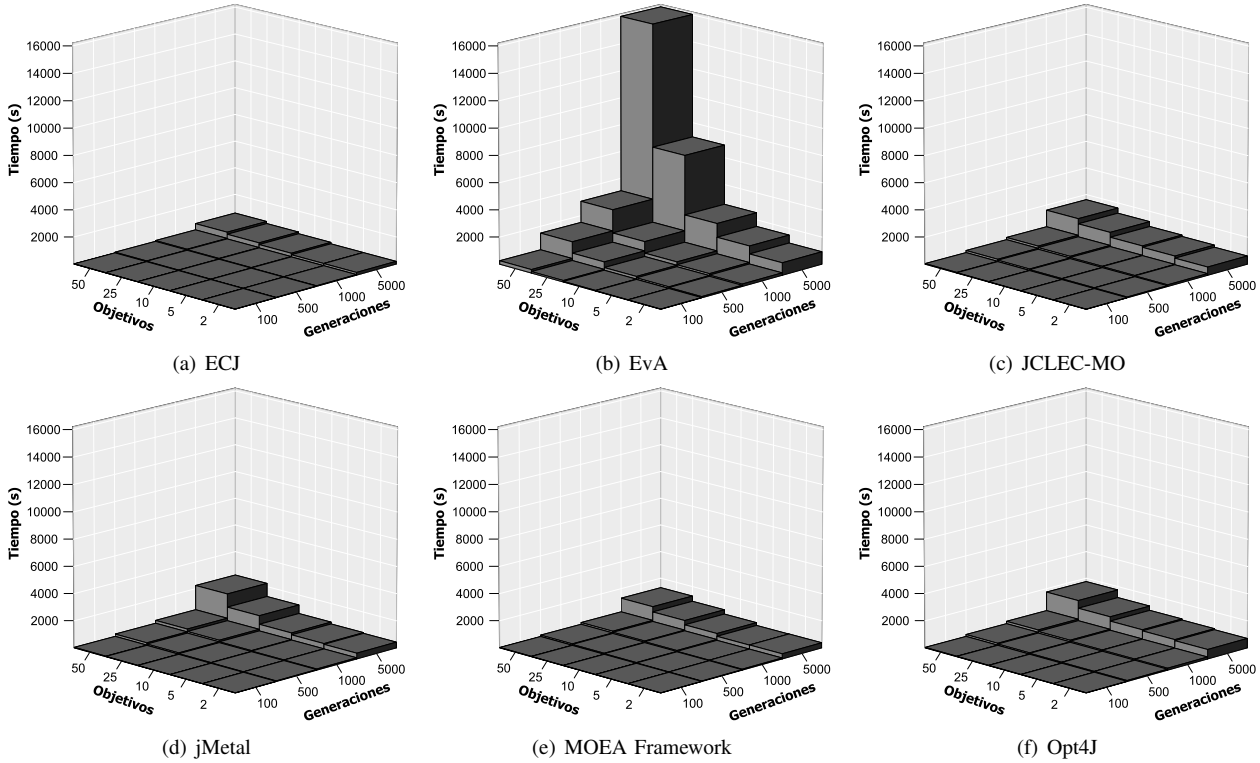
Este trabajo ha presentado una comparativa experimental de implementaciones de algoritmos multiobjetivo disponibles en seis librerías Java con el fin de analizar sus requisitos de tiempo y memoria. Los distintos algoritmos han sido ejecutados considerando configuraciones que difieren en el tipo de problema, el número de objetivos, el número de generaciones y el tamaño de la población. Los dos experimentos realizados indican que estos parámetros afectan en mayor o menor medida a todas las librerías. Para un uso estándar, las diferencias son poco apreciables y están condicionadas a la complejidad del algoritmo o del problema a resolver. Sin embargo, si se necesitan valores elevados para los parámetros anteriores, es conveniente analizar la forma en que cada librería implementa el algoritmo, pues las diferencias pueden ser notables. En este sentido, este estudio puede extenderse para abarcar un mayor número de librerías, algoritmos y problemas. Asimismo, sería interesante analizar la capacidad de las distintas librerías para paralelizar la evaluación de soluciones.

Referencias

[1] T. Stewart, O. Bandte, H. Braun, N. Chakraborti, M. Ehrgott, M. Göbelt, Y. Jin, H. Nakayama, S. Poles, and D. Di Stefano, "Real-World Applications of Multiobjective Optimization," in *Multiobjective Optimization*, vol. 5252 of *LNCS*, pp. 285–327, springer Berlin Heidelberg, 2008.

Tabla VI
Tiempos mínimos y máximos en segundos (experimento 2)

Librería	P = 100		P = 500		P = 1000		Ranking
	Min.	Máx.	Min.	Máx.	Min.	Máx.	
ECJ	$0,34 \pm 0,04$	$14,37 \pm 0,92$	$0,79 \pm 0,05$	$170,61 \pm 2,44$	$2,73 \pm 0,09$	$589,33 \pm 3,58$	1,00
EvA	$1,79 \pm 0,02$	$226,77 \pm 2,25$	$5,74 \pm 0,62$	$3830,01 \pm 94,69$	$16,38 \pm 0,49$	$16030,96 \pm 512,88$	6,00
JCLEC-MO	$0,64 \pm 0,07$	$30,55 \pm 0,58$	$3,25 \pm 0,10$	$475,77 \pm 4,83$	$11,86 \pm 0,07$	$1611,35 \pm 91,90$	4,11
jMetal	$0,58 \pm 0,06$	$30,74 \pm 1,21$	$2,64 \pm 0,07$	$576,81 \pm 3,86$	$9,01 \pm 0,17$	$2237,61 \pm 30,34$	3,35
MOEA Framework	$0,37 \pm 0,01$	$19,26 \pm 0,21$	$1,43 \pm 0,08$	$339,96 \pm 1,59$	$4,27 \pm 0,14$	$1284,00 \pm 7,17$	2,02
Opt4J	$0,93 \pm 0,08$	$39,18 \pm 0,68$	$4,12 \pm 0,09$	$533,96 \pm 6,95$	$13,98 \pm 0,31$	$1743,74 \pm 12,82$	4,52



(a) ECJ (b) EvA (c) JCLEC-MO

(d) jMetal (e) MOEA Framework (f) Opt4J

Figura 1. Variación en el tiempo de ejecución de NSGA-II para una población de 1000 individuos (experimento 2)

Tabla VII
Resumen del consumo de memoria (experimento 2)

Librería	Valores (KB)		Porcentajes	
	Min.	Máx.	Min.	Máx.
ECJ	123960	2064160	26,67	0,00
EvA	111080	1676000	15,00	0,00
JCLEC-MO	123840	6450680	13,33	0,00
jMetal	124080	14485480	23,33	23,33
MOEA Framework	123760	2589000	15,00	0,00
Opt4J	124480	3909920	6,67	76,67

[2] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd ed., 2007.

[3] J. A. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernández, "Metaheuristic optimization frameworks: a survey and benchmarking," *Soft Comput.*, vol. 16, no. 3, pp. 527–561, 2012.

[4] S. Chand and M. Wagner, "Evolutionary many-objective optimization: A quick-start guide," *Surv. Oper. Res. Manag. Sci.*, vol. 20, no. 2, pp. 35–42, 2015.

[5] A. López Jaimes and C. A. Coello Coello, *Springer Handbook of Computational Intelligence*, ch. Many-Objective Problems: Challenges and Methods, pp. 1033–1046. Springer Berlin Heidelberg, 2015.

[6] V. Khare, X. Yao, and K. Deb, "Performance Scaling of Multi-objective Evolutionary Algorithms," in *Proc. 2nd Int. Conf. Evolutionary Multi-Criterion Optimization*, pp. 376–390, 2003.

[7] B. Li, J. Li, K. Tang, and X. Yao, "Many-Objective Evolutionary Algorithms: A Survey," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 13:1–35, 2015.

[8] J. Maltese, B. M. Ombuki-Berman, and A. P. Engelbrecht, "A Scalability Study of Many-Objective Optimization Algorithms," *IEEE T Evolut. Comput.*, vol. 22, no. 1, pp. 79–96, 2018.

[9] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evolut. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[10] C. A. Coello Coello, "Evolutionary Multiobjective Optimization: Current and Future Challenges," in *Advances in Soft Computing*, pp. 243–256, Springer London, 2003.

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

[12] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, *Scalable Test Problems for Evolutionary Multiobjective Optimization*, ch. Evolutionary Multiobjective Optimization, pp. 105–145. Springer London, 2005.