Metaheuristics-based frameworks to solve the knapsack problem

Isela Jiménez-Castellano, Betania Hernández-Ocaña, José Hernández-Torruco, Oscar Chávez-Bosquez*

División Académica de Informática y Sistemas, Universidad Juárez Autónoma de Tabasco. Cunduacán, Tabasco, México.

Abstract.- Software frameworks allow the reuse of code designed for diverse problem-solving. Frameworks implementing metaheuristics for optimization problem solving are among the most interesting ones. The Knapsack Problem is one of the classic optimization examples commonly used as a benchmark as it is a simple yet complex problem: it belongs to the NP-Complete complexity class, considered more difficult than NP problems in general. In this paper, we implemented algorithms from three metaheuristics frameworks of different families in order to solve a form of the problem: the Knapsack Problem 0-1. The selected frameworks were JACOF (Ant Colony Optimization), JAMES (Trajectory-based Algorithms), and MOEA (Evolutionary Algorithms). We develop a software prototype including the most representative algorithms of each framework to solve a public benchmark set of the problem. We designed one stop criteria for all algorithms, and we perform 30 runs per algorithm to conclude that JAMES obtained the best results, although with higher standard deviation. However, MOEA is the most easy-to-implement framework, since it requires fewer lines of code to solve the problem.

Keywords: heuristics; optimization; software.

Frameworks basados en metaheurísticas para resolver el problema de la mochila

Resumen.- Actualmente existen frameworks que permiten reutilizar código diseñado ex profeso para resolver diversos problemas, entre los que destacan aquellos que implementan metaheurísticas para resolver problemas de optimización. El problema de la mochila es uno de los ejemplos de optimización clásicos usado a menudo como problema de prueba debido a su sencillez y al mismo tiempo su complejidad: pertenece a la categoría de problemas NP-Completos, considerados difíciles computacionalmente. En este artículo se implementaron los algoritmos de tres frameworks de metaheurísticas de diferentes familias para resolver una variante del problema: el problema de la mochila 0-1. Los frameworks seleccionados fueron: JACOF (Algoritmos de Colonia de Hormigas), JAMES (Algoritmos basados en Trayectoria), y MOEA (Algoritmos Evolutivos). Se desarrolló un prototipo de software incluyendo las metaheurísticas más representativas de cada framework y se utilizaron como benchmark un conjunto de instancias públicas del problema. Se diseñó un factor de finalización común para los algoritmos de cada framework, y a partir del diseño experimental se ejecutaron 30 pruebas por algoritmo cuyos resultados demuestran que JAMES obtienen mejores soluciones, aunque con mayor desviación estándar. Sin embargo, MOEA es el framework más sencillo de implementar, ya que implica menos líneas de código necesarias para resolver el problema.

Palabras claves: heurísticas; optimización; software.

Recibido: 20 diciembre 2018
Aceptado: 04 marzo 2019

1. Introducción

Las metaheurísticas surgen con la motivación de mejorar los procesos de búsqueda en optimización, las cuales son métodos de aproximación matemática que permiten encontrar más de una solución a un problema en poco tiempo sin tener que transformar el modelado matemático o el dinamismo del problema [1]. Aunque distintos autores sugieren diferentes clasificaciones, en este artículo se manejan estas tres categorías de metaheurísticas:

- los Algoritmos Evolutivos (AE), basados en la evolución natural de las especies [2].
- los Algoritmos de Inteligencia Colectiva (AIC), que simulan el comportamiento de

*Autor para correspondencia:
*Correo-e:*oscar.chavez@ujat.mx (Oscar Chávez-Bosquez)

ciertas especies simples e inteligentes en la búsqueda de alimento o refugio [3]

- los Algoritmos basados en Trayectoria (AT), que utilizan solamente una solución durante el proceso de búsqueda, describiendo una trayectoria desde la solución inicial hasta la solución final.

Los más populares son los AE debido a su éxito en la solución de problemas de optimización, entre los cuales destacan los Algoritmos genéticos, Estrategias evolutivas y Evolución diferencial. Sin embargo, algunas propuestas de AIC han tomado relevancia en estos últimos años tales como el Algoritmo de optimización mediante cúmulos de partículas, Colonia de hormigas, y el Algoritmo de optimización basado en el forrajeo de bacterias. Con respecto a los AT, la Búsqueda tabú, el Recocido simulado y la Búsqueda local iterativa constituyen los algoritmos más representativos [4].

Dada la popularidad de las metaheurísticas para resolver problemas de optimización de manera aproximada en tiempos razonables, se han desarrollado bibliotecas o frameworks que implementan un conjunto de algoritmos con el objetivo de encontrar una solución factible al problema y con los cuales se busca reducir el tiempo en que se desarrolla un software con código existente [5]. En la web se encuentran frameworks tanto de uso libre (aquellos en los que el código fuente se encuentra disponible al público) como de acceso limitado (en los cuales no todas las funciones son gratuitas o el código fuente no está disponible). Estos pueden ser implementados y adaptados por el usuario final para resolver problemas de optimización específicos.

Ahora bien, un problema interesante desde el punto de vista computacional lo constituye el problema de la mochila. Este problema ha sido estudiado exhaustivamente, encontrando soluciones tanto con AE como AIC y AT [6, 7, 8], entre otras muchas técnicas.

En este trabajo se seleccionaron tres frameworks de metaheurísticas de vanguardia, de libre licencia y con versiones *out-of-the-box* de metaheurísticas, es decir, sin necesidad de instalación o configuración previa. Estos tres frameworks: JACOF, JAMES y MOEA incluyen algoritmos de inteligencia Colectiva, evolutivos, y basados en trayectoria (respectivamente), lo cual les hace interesantes de comparar.

Los algoritmos más relevantes de cada framework se implementaron en un prototipo que denominamos *Lanzador de Frameworks de Metaheurísticas* (Prototipo LFM), con el objetivo de resolver un conjunto de instancias del problema de la mochila y así comparar su desempeño y la calidad de sus soluciones.

2. El problema de la mochila

El problema de la mochila (KP, por las siglas en inglés *Knapsack Problem*), es un problema clásico de Optimización combinatoria que pertenece a la familia de problemas NP-Completos. Se puede decir que esta clase de problemas (también conocidos informalmente como "intratables"), son los más difíciles de resolver, ya que un algoritmo de fuerza bruta para problemas NP-Completos utiliza tiempo exponencial con respecto al tamaño de la entrada. Se desconoce si hay mejores algoritmos, por lo cual es necesario utilizar diferentes enfoques que permitan obtener una solución adecuada en un tiempo razonable [9].

El KP consiste en un conjunto de elementos con un peso y valor específicos, que son seleccionados para maximizar el valor obtenido y el peso total de los elementos elegidos sin exceder la capacidad de la mochila [10]. Existen diversas variantes del problema estándar:

- Multidimensional: consiste en encontrar un subconjunto de objetos que maximicen el beneficio total mientras se satisfacen ciertas restricciones. Se trata de una variación computacionalmente más dura que el problema estándar [11].

- Múltiple: surge de la generalización del problema estándar cuando se tienen varias mochilas. Es usado en problemas de carga y programación de operaciones [12].

- Cuadrático: esta variante maximiza una función objetivo cuadrática sujeta a una restricción de capacidad lineal o binaria [13].
- Suma de subconjuntos: en esta variante el beneficio es igual al peso de cada elemento, $w_j = x_j$, y todas las mochilas tienen la misma capacidad. Esto no implica que el beneficio (o peso) sea el mismo para cada uno de los elementos [14].

De las variantes presentadas, el problema de la mochila 0-1 es considerado clásico en la literatura [15].

2.1. *El problema de la mochila 0-1 (KP 0-1)*

El KP 0-1 [16] constituye uno de los problemas más simples de la programación lineal. Aparece como subproblema en otros problemas más complejos y tiene diversas aplicaciones prácticas en la toma de decisiones del mundo real, tales como la búsqueda de patrones de corte para materias primas que generen el menor desperdicio posible [17], la selección de inversiones de capital y portafolios financieros [12] o la optimización de recursos computacionales [18]. Matemáticamente se expresa como [19]:

$$\text{maximizar} \quad \sum_{j=1}^{n} w_j x_j, \tag{1}$$

$$\text{sujeto a} \quad \sum_{j=1}^{n} w_j x_j \leq c, \tag{2}$$

$$\text{tal que } x_j \in \{0, 1\}, j = 1, ..., n \tag{3}$$

donde:

x_j : Variables de decisión,
w_j : Peso w del ítem j,
c : Capacidad total de la mochila,
n : Número de ítems.

El modelo matemático define que cada elemento corresponde a una variable x_j cuyo valor puede ser 1 si el elemento j se introduce a la mochila y 0 si se descarta. Además, se debe considerar que un elemento puede ser elegido si no se ha excedido el peso w_j.

2.2. *Instancias de prueba*

Al ser un problema vigente de optimización, existen múltiples instancias del problema de la mochila, específicamente del KP 0-1. Estas instancias se encuentran mayormente en archivos con formato de texto plano y varían en dificultad dependiendo el número de ítems a colocar en la mochila.

En este artículo se utiliza un conjunto de instancias disponibles en línea como instances_01_KP (adaptadas de David Pisinger's problems) [20]. Estas instancias son usadas como benchmark para comparar el desempeño de diferentes algoritmos. El conjunto consiste de 21 instancias con diferente cantidad de ítems, de los cuales se seleccionó un subconjunto de 7 instancias que pueden considerarse representativo del total.

Cada instancia (archivo de texto) se compone de valores enteros separados por espacios en blanco y saltos de línea. El formato es como sigue:

```
[Número de ítems] [Capacidad de la mochila]
[Peso del ítem 1] [Ganancia del ítem 1]
...
[Peso del ítem n] [Ganancia del ítem n]
```

La primera línea del archivo contiene el número total de ítems y la capacidad máxima de la mochila. Las líneas subsecuentes contienen el peso y la ganancia de cada ítem. Cabe mencionar que existen otros formatos comúnmente utilizados en la literatura, pero todos incluyen de alguna manera los datos mostrados. Las características de las instancias seleccionadas son:

- KP1: 100 ítems, 995 de capacidad máxima,
- KP2: 200 ítems, 1008 de capacidad máxima,
- KP3: 500 ítems, 2543 de capacidad máxima,
- KP4: 1000 ítems, 5002 de capacidad máxima,
- KP5: 2000 ítems, 25016 de capacidad máxima,
- KP6: 5000 ítems, 10011 de capacidad máxima,
- KP7: 10000 ítems, 49877 de capacidad máxima.

3. Metaheurísticas

Las metaheurísticas son estrategias para diseñar y mejorar los procedimientos heurísticos orientados a obtener un alto rendimiento. El término metaheurística fue introducido por Fred Glover [21] en 1986 y desde entonces se han presentado nuevas propuestas como alternativas de solución a problemas. Estos algoritmos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en donde las heurísticas clásicas no son efectivas [22].

Las metaheurísticas constituyen un campo disciplinar de gran auge en la Inteligencia Artificial. Han alcanzado un alto prestigio, como demuestran la amplia gama de problemas a los que se han aplicado con éxito en la literatura, así como el gran número de revistas, libros y conferencias dedicados a este tema. Además, proporcionan un marco general en la creación de nuevos algoritmos híbridos, combinando conceptos de áreas como: biología, matemáticas, neurología, física, entre otras [23]. A continuación se describen las metaheurísticas empleadas en esta investigación.

3.1. Algoritmos Evolutivos (AE)

Los AE están basados en la evolución biológica, incluyendo elementos como la reproducción, la mutación, la recombinación y la selección. Se trata de métodos de optimización y búsqueda estocásticos inspirados en la teoría de la evolución de Darwin. Estos algoritmos a menudo realizan soluciones aproximadas y utilizan la evolución simulada para explorar soluciones en problemas complejos del mundo real [24]. Los AE son una herramienta muy popular para buscar, optimizar y proporcionar soluciones a problemas complejos [25].

3.2. Optimización basada en Colonias de Hormigas (OCH)

La OCH, también conocida como ACO por las siglas en inglés de *Ant Colony Optimization*, es una técnica que pertenece a los AIC y fue introducida como herramienta para la solución de problemas complejos [26]. Esta técnica es utilizada principalmente para solucionar problemas que buscan los mejores caminos o rutas en grafos, aunque puede adaptarse en general para cualquier problema de optimización. Las hormigas se comunican a través de sus feromonas, las cuales son sustancias que les permiten encontrar los caminos más cortos entre su nido y la fuente de alimentos. OCH es una metaheurística basada en el comportamiento real de este insecto [27].

3.3. Algoritmos basados en Trayectoria (AT)

Los AT, también conocidos como TBM por las siglas en inglés de *Trajectory-based Metaheuristics* inician con una solución y buscan en el espacio de soluciones candidatas (el espacio de búsqueda) por una mejor solución. Si la encuentran, reemplaza su solución actual por la nueva y continúa con el proceso hasta que se encuentre una solución óptima [28]. Se caracterizan por partir de un punto específico para mejora continua de la solución actual mediante la inspección de un vecindario. En general, la búsqueda finaliza cuando se alcanza un número máximo de iteraciones y se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso [29].

4. Frameworks de Metaheurísticas

Un framework (definido de manera general como entorno o marco de trabajo) es un conjunto de prácticas empleadas en el desarrollo de software para resolver problemas de forma sencilla y segura. Los frameworks mantienen un comportamiento útil, definido e identificable, lo que les permite proporcionar funcionalidades específicas [30].

Existen frameworks para multitud de rutinas y algoritmos en prácticamente cualquier lenguaje de programación, el caso de las metaheurísticas no es la excepción. Con los frameworks se busca utilizar y reutilizar el código existente, agilizar el proceso de desarrollo de software y promover mejores prácticas de programación [5].

Los frameworks de metaheurísticas existentes implementan gran variedad de algoritmos heurísticos de diferente tipo.. A continuación se presentan los frameworks seleccionados para resolver el KP 0-1, los cuales se escogieron

principalmente por su madurez, actualización reciente, buena documentación, multiplataforma, desarrollados bajo la plataforma Java y de licencia libre.

4.1. Java Ant Colony Framework (JACOF)

JACOF es un framework para la OCH que implementa las variantes más importantes de esta categoría: *Ant System* (AS), *Elitist Ant System* (EAS), *Ant Colony System* (ACS), *Rank-based Ant System* (ASRank), y el *Max-Min Ant System - MMAS*.

Esta compuesto por la especificación de algoritmos junto con su implementación, colección de problemas clásicos en la literatura y compatibilidad con bibliotecas que incluyen instancias para estos problemas.

Para utilizar JACOF se elige un problema, una variante de OCH y se ejecuta el algoritmo. La documentación y los archivos de descarga son distribuidos en la plataforma GitHub con licencia de uso libre.

4.2. JAva MEtaheuristics Search framework (JAMES)

JAMES es un framework que utiliza AT y proporciona guías de implementación para un conjunto de problemas. En este framework se puede analizar fácilmente el rendimiento de los algoritmos y la influencia de los valores de los parámetros. Una manera sencilla de hacerlo es definir el problema, seleccionar una estrategia de optimización adecuada y aplicar un algoritmo de búsqueda para encontrar la mejor solución.

JAMES está compuesto de tres módulos: principal, extensiones y ejemplos. El primer módulo proporciona una amplia variedad de metaheurísticas como: *Parallel Tempering*, *Variable Neighbourhood Search*, *Random Descent*, entre otros.

El segundo módulo tiene herramientas adicionales para la especificación de problemas (permutación y análisis automático). El tercer módulo proporciona ejemplos de problemas y soluciones con diferentes algoritmos.

El código fuente está bajo Licencia Permisiva de Apache 2.0, la documentación y el sitio web se encuentran publicados bajo la Licencia Internacional Creative Commons Atribution 4.0.

4.3. MultiObjective Evolutionary Algorithms (MOEA)

MOEA es un framework para desarrollo y experimentación con algoritmos evolutivos multi-objetivo (MOEAs). Su objetivo es proporcionar una colección completa de algoritmos y herramientas para la optimización de objetivos únicos y multi-objetivos.

MOEA incluye múltiples variantes de algoritmos genéticos, entre las que destacan: *Non-dominated Sorting Genetic Algorithm II* (NSGA-II), *Reference-Point Based Nom-dominated Sorting Genetic Algorithm* (NSGA-III), *Genetic Algorithm with Elitism (Single Objective)* (GA), *Vector Evaluated Genetic Algorithm* (VEGA). Además, es posible incluir algoritmos de los frameworks JMetal y PISA.

MOEA es de código abierto, proporciona herramientas para el diseño rápido, desarrollo, ejecución y estadísticas de prueba en algoritmos de optimización. Adicionalmente, cuenta con un módulo de descargas que proporciona los recursos necesarios para el desarrollo de aplicaciones y una guía de inicio rápido con los pasos para configurar y ejecutar ejemplos de problemas. La documentación y el sitio web son publicados bajo la licencia GNU Lesser General.

5. Prototipo LFM

El prototipo LFM (“Lanzador de Frameworks de Metaheurísticas”) fue desarrollado bajo la plataforma Java (Open JDK 7) usando exclusivamente bibliotecas de licencia libre. El prototipo resuelve el KP 0-1 utilizando los algoritmos mencionados de cada uno los tres frameworks.

La Figura 1 muestra el diagrama de clases del Prototipo LFM. Las clases principales son:

- `Main`: clase principal que consiste de la ventana principal del Prototipo.
- `Framework`: clase abstracta que contiene la solución (número de ítems en la mochila, peso

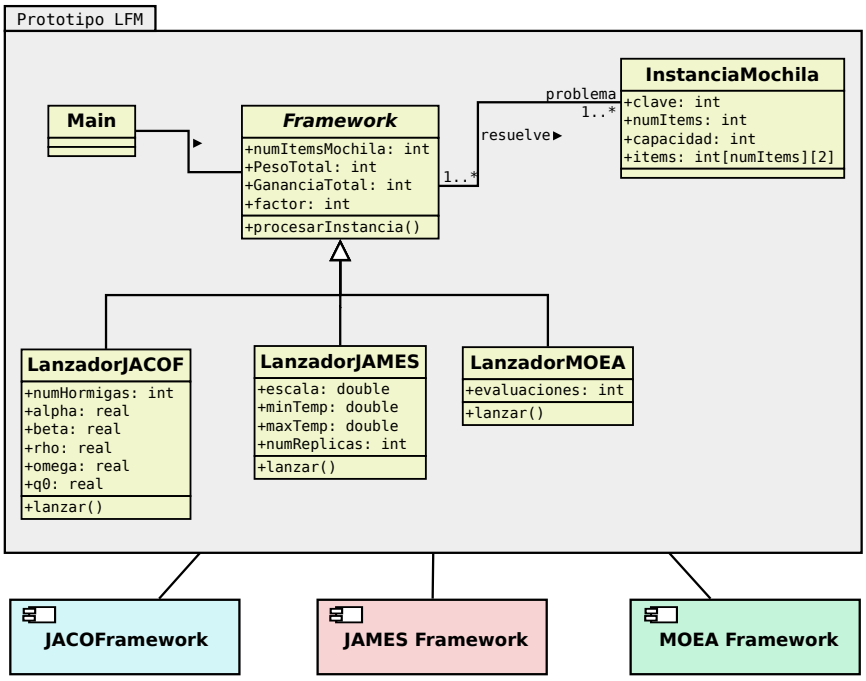

Figura 1: Diagrama de clases

y ganancia totales), así como el factor (criterio de finalización) de cada algoritmo.

- `InstanciaMochila`: representa una instancia particular del KP 0-1. Incluye los datos del problema: total de ítems, capacidad de la mochila, y una matriz con el peso y ganancia de cada ítem.

- `LanzadorMOEA`: hereda de `Framework`, incluye el número de evaluaciones y los métodos `convertir()` para traducir una instancia al formato requerido por MOEA y el método `lanzar()` para ejecutar uno de los AE.

- `LanzadorJAMES`: hereda de `Framework`, incluye los atributos escala, temperatura mínima, temperatura máxima, número de réplicas (atributos requeridos por el algoritmo de *Parallel Tempering*). Incorpora los métodos `convertir()` para traducir una instancia al formato requerido por JAMES y el método `lanzar()` para ejecutar un AT.

- `LanzadorJACOF`: hereda de `Framework`, incluye el número de hormigas, el rastro de feromona, las iteraciones, alfa, beta y la tasa de evaporación. Incorpora los métodos `convertir()` para traducir una instancia al formato requerido por JACOF y el método `lanzar()` para ejecutar un algoritmo de OCH.

La clase `Main` utiliza la clase genérica `Framework` para ejecutar las metaheurísticas seleccionadas por el usuario. A su vez, la clase `Framework` utiliza la clase `InstanciaMochila` para resolver una instancia particular del KP 0-1. Las clases `LanzadorJACOF`, `LanzadorJAMES` y `LanzadorMOEA` invocan los algoritmos de los frameworks JACOF, JAMES y MOEA, respectivamente.

En la Figura 2 se muestra la interfaz del Prototipo LFM, la cual contiene los siguientes elementos que describirán a detalle en la siguiente sección:

- Elección de la instancia del KP 0-1 a resolver.

- Factor (criterio de finalización) de cada algoritmo.

- Selección del framework o de los algoritmos individualmente a ejecutar.

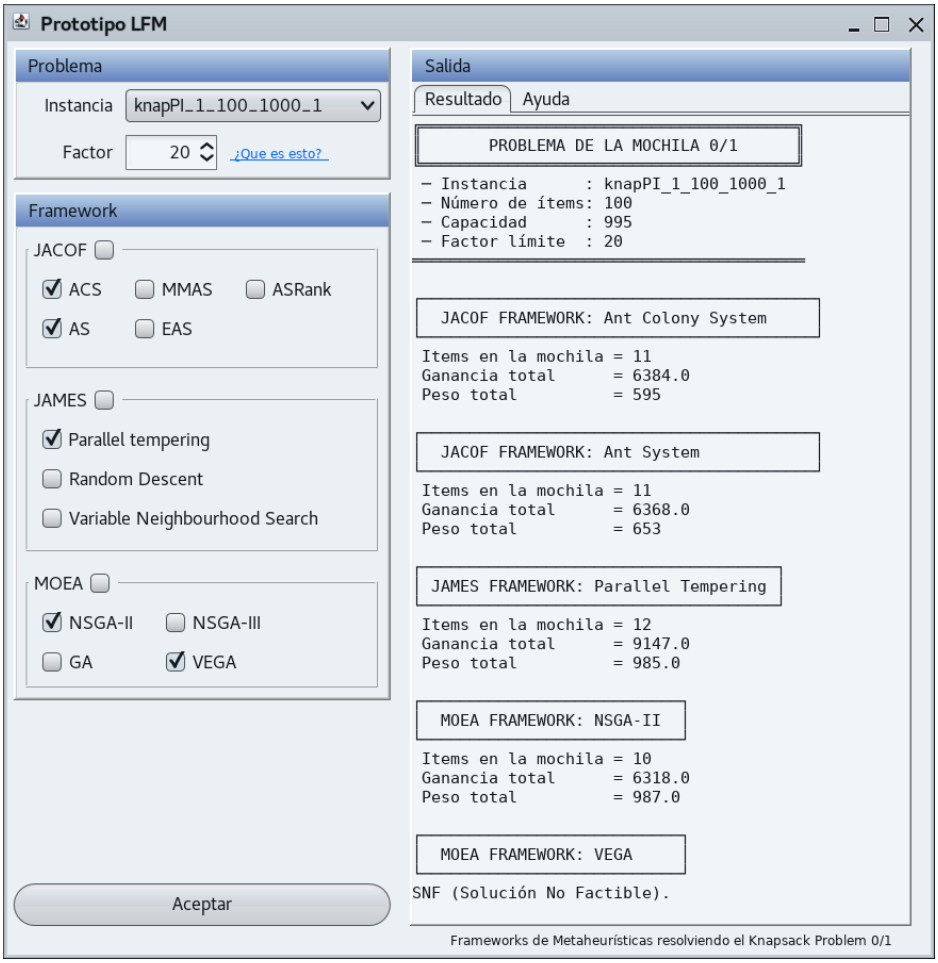

Figura 2: Prototipo LFM (Lanzador de *Frameworks* de Metaheurísticas)

- Panel de salida que muestra el resultado obtenido por cada algoritmo y módulo de ayuda.

El prototipo LFM es software libre y se encuentra disponible en la plataforma GitHub.

6. Pruebas y resultados

6.1. Diseño experimental

El prototipo LFM se desarrolló y probó en una computadora portátil Alienware M17x Intel Core i7-2670QM CPU @ 2.20GHz bajo el sistema operativo Ubuntu 16.04.3 LTS de 64-bits. Se realizaron 30 ejecuciones independientes para cada framework usando los mismos valores en los parámetros. La sintonización de parámetros del Prototipo LFM usada fue:

MOEA : Los 4 algoritmos genéticos de este framework tienen la misma configuración de parámetros:

- Evaluaciones: `factor`×100000, evalúa la aptitud de cada individuo en la población, los individuos más aptos son seleccionados y el genoma de cada individuo es modificado para formar una nueva generación.

JAMES : El algoritmo *Parallel Tempering* tiene los siguientes parámetros:

- $temp_{min}$: Temperatura mínima que muestrea una región más pequeña y puede quedar en mínimos locales.
- $temp_{max}$: Temperatura máxima donde la temperatura más alta intercambia soluciones con la temperatura más baja

Tabla 1: Resultados de los algoritmos de JACOF.

		ACS			AS			ASRank			MMAS			EAS		
Instancia	Métrica	Ítems	G	Peso	Ítems	G	Peso	Ítems	G	Peso	Ítems	G	Peso	Ítems	G	Peso
KP1	Mejor	11	6384	595	11	6384	595	11	6384	595	11	6384	595	11	6384	595
Ítems: 100	Mediana	11	6384	595	11	6368	629	11	6251.5	608	11	6361	595	11	6279	623
Capacidad:	Peor	11	6198	639	10	6059	617	11	5996	662	10	5943	550	11	6142	630
995	Media	-	6369.36	-	-	6305.16	-	-	6243.66	-	-	6311.16	-	-	6287.63	-
	STD	-	40.75	-	-	86.13	-	-	110.87	-	-	97.91	-	-	74.73	-
KP2-KP5		SNF			SNF			SNF			SNF			SNF		
KP6-KP7		EMI			EMI			EMI			EMI			EMI		

G: Ganancia SNF: Solución no factible EMI: Error de memoria insuficiente

y puede muestrear una mayor cantidad de espacio.

- Número de réplicas: Tiene un valor de 10 y se ordena de acuerdo a la temperatura.

JACOF : Dependiendo la variante de OCH, los valores de los parámetros son:

- *numHormigas*: 10,
- α: 1.0 Factor de influencia de feromonas,
- β: 2.0 Información heurística,
- ρ: 0.1 Coeficiente de evaporación de feromonas,
- ω: 0.1 Parámetro local de decaimiento de feromonas,
- Q_0: 0.9. Parámetro adicional que corresponde al nivel de exploración de las hormigas.

La variable denominada `factor` representa el criterio de finalización de cada framework. En el caso de JACOF equivale al número de iteraciones de cada algoritmo de OCH; para JAMES equivale al número de segundos de ejecución de cada AT; en MOEA equivale al número de evaluaciones multiplicado por 100000. Esta configuración del criterio de finalización fue calculada a partir de pruebas empíricas, ejecutando todos los algoritmos para definir un criterio de finalización justo dependiendo el framework.

6.2. Resultados

Se realizaron 30 ejecuciones de cada uno de los 12 algoritmos seleccionados para resolver las 7 instancias del KP 0-1 y así identificar qué framework contiene los algoritmos más eficientes.

Las Tablas 1, 2, y 3 muestran los resultados obtenidos por los algoritmos de cada framework en las 7 instancias del problema de la mochila. La Tabla 4, muestra los resultados del mejor algoritmo de cada framework en cada una de las 7 instancias. El formato de las cuatro tablas es el mismo: en la primera columna se muestran las características de cada instancia (número de ítems y capacidad), la segunda columna contiene las métricas utilizadas (mejor solución, peor, media, mediana y desviación estándar), y finalmente se muestra el número de ítems en la mochila, la ganancia y el peso, por cada algoritmo. Los algoritmos se muestran ordenados de izquierda a derecha de acuerdo a la mejor solución obtenida.

En la Tabla 1 se muestran los resultados obtenidos por las 5 variantes de OCH incluídas en JACOF, en las 7 instancias del problema. Como se puede observar, los 5 algoritmos basados en colonia de hormigas únicamente resuelven la instancia más sencilla del problema, el KP1. En cuanto a las instancias KP6 y KP7, consideradas las más difíciles, los 5 algoritmos lanzan una `OutOfMemoryException`, que ocurre cuando la máquina virtual de Java carece de la memoria suficiente para instanciar nuevos objetos. Cabe mencionar que no se realizó ninguna configuración adicional para ejecutar el Prototipo, se utilizó la

Tabla 2: Resultados de los algoritmos de JAMES.

		Parallel Tempering			Random Descent			Variable Neighbourhood Search		
Instancia	Métrica	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso
KP1	Mejor	12	9147	985	8	6919	981	8	6871	959
Ítems: 100	Mediana	12	9147	985	8	6895	975.5	7	6150	965.5
Capacidad:	Peor	12	9147	985	7	6295	970	6	5429	972
995	Media	-	4268.6	-	-	900.13	-	-	410	-
	STD	-	0	-	-	304.84	-	-	1019.64	-
KP2	Mejor	16	11238	987	11	9245	998	12	9738	1004
Ítems: 200	Mediana	16	11238	987	11	9245	998	11	9134	1000.5
Capacidad:	Peor	16	11238	987	11	9245	998	10	8530	997
1008	Media	-	1123.8	-	-	308.16	-	-	608.93	-
	STD	-	0	-	-	0	-	-	854.18	-
KP3	Mejor	42	28857	2543	40	35844	5001		SNF	
Ítems: 500	Mediana	42	28857	2543	40	35844	5001		SNF	
Capacidad:	Peor	42	28857	2543	40	35844	5001		SNF	
2543	Media	-	3847.6	-	-	1194.8	-		SNF	
	STD	-	0	-	-	0	-		SNF	
KP4	Mejor	81	54403	4998		SNF			SNF	
Ítems: 1000	Mediana	77	53894	4998		SNF			SNF	
Capacidad:	Peor	71	52653	4998		SNF			SNF	
5002	Media	-	8937.56	-		SNF			SNF	
	STD	-	842.66	-		SNF			SNF	
KP5	Mejor	153	109759	10011		SNF			SNF	
Ítems: 2000	Mediana	153	109759	10011		SNF			SNF	
Capacidad:	Peor	153	109759	10011		SNF			SNF	
10011	Media	-	3658.63	-		SNF			SNF	
	STD	-	0	-		SNF			SNF	
KP6	Mejor	287	239149	25009		SNF			SNF	
Ítems: 5000	Mediana	271	228332	24955		SNF			SNF	
Capacidad:	Peor	26	14070	14926		SNF			SNF	
25016	Media	-	16051.7	-		SNF			SNF	
	STD	-	126942.09	-		SNF			SNF	
KP7	Mejor	503	432592	49868		SNF			SNF	
Ítems:	Mediana	296	273071	49314		SNF			SNF	
10000	Peor	45	23026	22170		SNF			SNF	
Capacidad:	Media	-	24289.63	-		SNF			SNF	
49877	STD	-	206443.6	-		SNF			SNF	

SNF: Solución no factible.

memoria que utiliza de manera predeterminada la máquina virtual de Java.

La Tabla 2 contiene los resultados obtenidos por los 3 algoritmos de JAMES en las 7 instancias del problema. En primer lugar, el algoritmo de *Parallel Tempering* resuelve efectivamente las 7 instancias del problema, resaltando el hecho que la desviación estándar es igual a 0 para las primeras 5 instancias, pero muy alta para las otras 2 instancias, consideradas más difíciles. Por otra parte, el algoritmo *Random Descent* resuelve únicamente las 3 primeras instancias del problema, eso sí, con desviación estándar igual a 0. Para el caso del algoritmo *Variable Neighbourhood Search*, éste resuelve únicamente las 2 primeras instancias y con alta variabilidad, como lo demuestra su desviación estándar.

Los resultados obtenidos por los AE de MOEA se muestran en la Tabla 3. Los 4 algoritmos seleccionados fueron ejecutados con el mismo número de evaluaciones, de los cuales solamente el NSGA-II y NSGA-III encuentran soluciones para las primeras 4 instancias del problema. GA solamente resuelve 2 instancias, mientras que VEGA no encontró solución para ninguna de las instancias. En cuanto a las instancias KP5 a KP7, ninguno de los algoritmos encontró una solución valida.

En la Figura 3 se muestra el gráfico con las mejores soluciones para el KP 0-1 obtenidas por el mejor algoritmo de cada framework, derivado de las 30 ejecuciones por instancia. En general,

Tabla 3: Resultados de los algoritmos de MOEA.

		NSGA-II			NSGA-III			GA			VEGA		
Instancia	Métrica	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso
KP1	Mejor	11	6384	993	11	6384	993	11	6384	993		SNF	
Ítems: 100	Mediana	11	6368	989	10.5	6251.5	977	10	6318	981		SNF	
Capacidad:	Peor	11	6116	992	9	5753	962	10	6132	973		SNF	
995	Media	-	6334.76	-	-	6210.03	-	-	6310	-		SNF	
	STD	-	75.60	-	-	160.04	-	-	62.81	-		SNF	
KP2	Mejor	18	10463	993	18	10463	993	17	10001	996		SNF	
Ítems: 200	Mediana	18	10390	997.5	18	10263.5	1000	16	9196	985		SNF	
Capacidad:	Peor	17	10021	1005	16	9926	1001	15	8080	847		SNF	
1008	Media	-	10337.46	-	-	10250.46	-	-	9148.63	-		SNF	
	STD	-	139.33	-	-	182.60	-	-	457.36	-		SNF	
KP3	Mejor	13	29505	2537	42	29217	2540		SNF			SNF	
Ítems: 500	Mediana	42	29315	2537	42	28481.5	2534.5		SNF			SNF	
Capacidad:	Peor	41	28868	2510	40	27774	2541		SNF			SNF	
2543	Media	-	29271.63	-	-	28445.96	-		SNF			SNF	
	STD	-	162.500	-	-	340.73	-		SNF			SNF	
KP4	Mejor	71	44484	5001	64	36642	4963		SNF			SNF	
Ítems: 1000	Mediana	68	40919.5	4936	61	32339	4939		SNF			SNF	
Capacidad:	Peor	64	36663	4827	50	23164	4889		SNF			SNF	
5002	Media	-	40915.4	-	-	30475.76	-		SNF			SNF	
	STD	-	2026.91	-	-	2867.59	-		SNF			SNF	
KP5 - KP7			SNF			SNF			SNF			SNF	

SNF: Solución no factible.

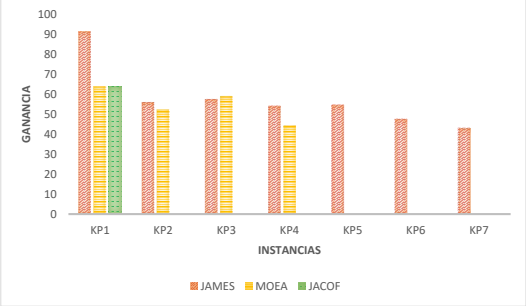

Figura 3: Mejores resultados por instancia

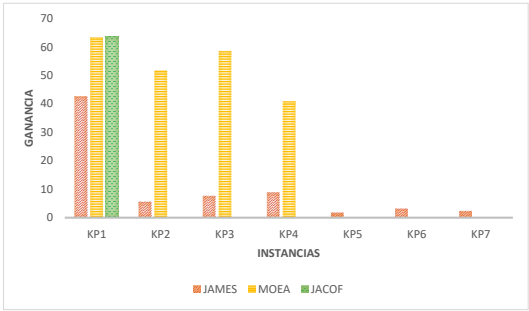

Figura 4: Media de los resultados por instancia

JAMES obtiene los mejores resultados. Aunque MOEA supera a JAMES en la instancia KP3, éste no encuentra soluciones a partir de la instancia KP5. En el caso de JACOF, solo encuentra solución para la primera instancia del problema con el mismo resultado que MOEA.

En la Figura 4 se presenta la media de los resultados del mejor algoritmo de cada framework. Se puede observar que, a pesar de que el framework JAMES obtuvo los mejores resultados, MOEA sobresale en la distribución de la media aritmética.

7. Conclusiones

Resolver problemas de optimización de forma eficiente es una tarea compleja. Afortunadamente, existen algoritmos e implementaciones de terceros que pueden apoyar en esta situación.

En este artículo se utilizaron tres frameworks de metaheurísticas integrados por diferentes familias de algoritmos para resolver el problema de la mochila 0-1. Los frameworks empleados fueron (en orden alfabético) JACOF, JAMES y MOEA. En JAMES se seleccionaron 3 metaheurísticas basadas en trayectoria de acuerdo al número de trabajos en los que han sido abordados en la

Tabla 4: Resultados generales.

		Parallel Tempering (JAMES)			NSGA-II (MOEA)			ACS (JACOF)		
Instancia	Métrica	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso	Ítems	Ganancia	Peso
KP1	Mejor	12	9147	985	11	6384	993	11	6384	595
Ítems: 100	Mediana	12	9147	985	11	6368	989	-	6384	-
Capacidad:	Peor	12	9147	985	11	6116	992	11	6198	639
995	Media	-	4268.6	-	-	6334.76	-	-	6369.36	-
	STD	-	0	-	-	75.60	-	-	40.75	-
KP2	Mejor	16	11238	987	18	10463	993		SNF	
Ítems: 200	Mediana	16	11238	987	18	10390	997.5		SNF	
Capacidad:	Peor	16	11238	987	17	10021	1005		SNF	
1008	Media	-	1123.8	-	-	10337.46	-		SNF	
	STD	-	0	-	-	139.33	-		SNF	
KP3	Mejor	42	28857	2543	43	29505	2537		SNF	
Ítems: 500	Mediana	42	28857	2543	42	29315	2537		SNF	
Capacidad:	Peor	42	28857	2543	41	28868	2510		SNF	
2543	Media	-	3847.6	-	-	29271.63	-		SNF	
	STD	-	0	-	-	162.50	-		SNF	
KP4	Mejor	81	54403	4998	71	44484	5001		SNF	
Ítems: 1000	Mediana	77	53894	4998	68	40919.5	4936		SNF	
Capacidad:	Peor	71	52653	4998	64	36663	4827		SNF	
5002	Media	-	8937.56	-	-	40915.4	-		SNF	
	STD	-	842.66	-	-	2026.91	-		SNF	
KP5	Mejor	153	109759	10011		SNF			SNF	
Ítems: 2000	Mediana	153	109759	10011		SNF			SNF	
Capacidad:	Peor	153	109759	10011		SNF			SNF	
10011	Media	-	3658.63	-		SNF			SNF	
	STD	-	0	-		SNF			SNF	
KP6	Mejor	287	239149	25009		SNF			EMI	
Ítems: 5000	Mediana	271	228332	24955		SNF			EMI	
Capacidad:	Peor	26	14070	14926		SNF			EMI	
25016	Media	-	16051.7	-		SNF			EMI	
	STD	-	126942.09	-		SNF			EMI	
KP7	Mejor	503	432592	49868		SNF			EMI	
Ítems:	Mediana	296	273071	49314		SNF			EMI	
10000	Peor	45	23026	22170		SNF			EMI	
Capacidad:	Media	-	24289.63	-		SNF			EMI	
49877	STD	-	206443.6	-		SNF			EMI	

SNF: Solución no factible. EMI: Error de memoria insuficiente

literatura. En MOEA se seleccionaron 4 variantes de algoritmos genéticos nativos del framework, sin considerar algoritmos que pueden incluirse a partir de otros frameworks. De JACOF se incluyeron las 5 variantes de OCH presentes en el framework.

Estos algoritmos se implementaron en un Prototipo multiplataforma denominadado LFM: "Lanzador de Frameworks de Metaheurísticas", con el objetivo de contar con una interfaz de usuario adecuada para resolver el problema. El Prototipo LFM es software libre disponible en línea.

Para medir la eficiencia de los frameworks se realizaron 30 ejecuciones independientes de cada algoritmo en el prototipo LFM, resolviendo un subconjunto de 7 instancias públicas del KP 0-1. Los resultados obtenidos se compararon de acuerdo a la ganancia total, ya que el objetivo es maximizar este valor con los ítems disponibles.

El comportamiento de cada framework es distinto, y dado que cada algoritmo tiene una configuración de parámetros distinta, esto influye en el desempeño de cada uno de los frameworks analizados. JAMES inicia con una solución y realiza modificaciones locales para mejorarla, por lo que encuentra soluciones rápidamente. Sin embargo, su solución final depende de la solución inicial. MOEA simula la evolución natural, donde existe un conjunto de entidades que representan posibles soluciones, se mezclan y compiten entre sí. Las más aptas son capaces de prevalecer durante más tiempo, evolucionando hacia mejores soluciones. JACOF emula el comportamiento de

las hormigas cuando están en busca de un camino entre la colonia y una fuente de alimentos.

En los resultados experimentales de la Tabla 4 se observa que el algoritmo de *Parallel Tempering* de JAMES resuelve las 7 instancias del problema, a diferencia del NSGA-II de MOEA que resuelve únicamente 4 instancias del problema y el ACS de JACOF que solo resuelve la instancia KP1. Sin embargo, aunque JAMES obtuvo los mejores resultados al resolver todas las instancias del problema, MOEA tiene una mejor distribución de la media, es decir, no hay mucha desviación entre el mejor y el peor resultado. En otras palabras, JAMES en algunas ejecuciones no encontró una solución factible, mientras que MOEA en todas las ejecuciones logró encontrar una solución válida (en 4 instancias del problema).

Por otro lado, para medir la usabilidad de los frameworks se consideró la facilidad de codificación para implementar cada algoritmo seleccionado. Con respecto al uso, los algoritmos de MOEA son los más fáciles de implementar de acuerdo con el número de líneas de código que se emplean, seguido por el JACOF que solo diferencia en la configuración de parámetros de cada algoritmo. No obstante JAMES es el más complicado de implementar debido al número de líneas de código que requiere la implementación de cada metaheurística.

De cierta manera, los resultados obtenidos fueron los esperados, ya que los algoritmos basados en colonias de hormigas son generalmente los de menor rendimiento y los algoritmos genéticos no son tan eficientes en comparación con los basados en Trayectoria. Como trabajo futuro se tiene contemplado realizar pruebas con otros frameworks que incluyan metaheurísticas de distintas familias.

8. Referencias

[1] P. Siarry and Z. Michalewicz. *Advances in Metaheuristic methods for hard optimization*. Springer, 2008.

[2] Z. Michalewicz and D. B. Fogel. *How to solve it: modern heuristics*. Springer, 2 edition, 2004.

[3] A. P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 1 edition, 2006.

[4] S. Luke. *Essentials of Metaheuristics*. Lulu, 2 edition, 2013. Available for free at https://cs.gmu.edu/~sean/book/metaheuristics/.

[5] J. M. Galindo-Haro. Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE. Trabajo Especial de Grado, Universitat Oberta de Catalunya, España, 2010.

[6] X. Kong, L. Gao, H. Ouyang, and S. Li. A simplified Binary Harmony Search algorithm for large scale 0-1 knapsack problems. *Expert Systems with Applications*, 42(12):5337–5355, 2015.

[7] Y. Kim, J-H. Kim, and K-H. Han. Quantum-inspired multiobjective evolutionary algorithm for multiobjective 0/1 knapsack problems. In *IEEE Congress on Evolutionary Computation, 2006. CEC 2006*, pages 2601–2606. IEEE, 2006.

[8] H. Shi. Solution to 0/1 knapsack problem based on improved Ant Colony Algorithm. In *Information Acquisition, 2006 IEEE International Conference on*, pages 1062–1066. IEEE, 2006.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press & McGraw Hill, 2 edition, 2001.

[10] A. Fuentes-Penna, D. Vélez-Díaz, S. Moreno-Gutiérrez, M. Martínez-Cervantes y O. Sánchez-Muñoz. Problema de la mochila (*Knapsack problem*). *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, 3(6), 2015.

[11] D. Soto, W. Soto y Y. Pinzón. Algoritmo de Optimización de Colonia de Hormigas Multiobjetivo aplicado al problema de la mochila multidimensional. *Revista Programación Matemática y Software*, 3(2):20–31, 2012.

[12] H. Kellerer, U. Pferschy, and D. Pisinger. *Introduction to NP-Completeness of Knapsack Problems*. Springer Berlin Heidelberg, 2004.

[13] D. Pisinger. The quadratic knapsack problem—a survey. *Discrete applied mathematics*, 155(5):623–648, 2007.

[14] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[15] S. Martello, D. Pisinger, and P. Toth. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, 123(2):325–332, 2000.

[16] B. Silva y L. Torres. Acerca de una versión dinámica del problema de la mochila. *Revista Politécnica*, 34(1):142, 2014.

[17] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem. *Operations research*, 9(6):849–859, 1961.

[18] R. Parra-Hernandez, D. Vanderster, and N. Dimopoulos. Resource management and knapsack formulations on the grid. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 94–

101. IEEE Computer Society, 2004.

[19] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.

[20] C. Cobos, H. Dulcey, J. Ortega, M. Mendoza, and A. Ordoñez. A Binary Fisherman Search Procedure for the 0/1 Knapsack Problem. In O. Luaces, J.A. Gámez, E. Barrenechea, A. Troncoso, M. Galar, H. Quintián, and E. Corchado, editors, *Advances in Artificial Intelligence*, pages 447–457, Cham, 2016. Springer International Publishing.

[21] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. Applications of Integer Programming.

[22] I. Osman and J. Kelly. *Metaheuristics: Theory and Applications*. Springer, US., 1 edition, 1996.

[23] O. Suarez. Una aproximación a la heurística y metaheurísticas. *INGE UAN-Tendencias en la Ingeniería*, 1(2), 2011.

[24] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co, Reading, Massachusetts, 1 edition, 1989.

[25] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265. IEEE, 2016.

[26] M. Dorigo and C. Blum. Ant Colony Optimization Theory: A Survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005.

[27] C. Robles-Algarín. Optimización por colonia de hormigas: aplicaciones y tendencias. *Revista Ingeniería Solidaria*, 6(10):83–89, 2010.

[28] J. Shi and Q. Zhang. A new cooperative framework for parallel trajectory-based metaheuristics. *Applied Soft Computing*, 65:374 – 386, 2018.

[29] N. Alancay, S.M. Villagra y N.A. Villagra. Metaheurísticas de trayectoria y poblacional aplicadas a problemas de optimización combinatoria. *Informes Científicos-Técnicos UNPA*, 8(1):202–220, 2016.

[30] D. Riehle. *Framework Design: A Role Modeling Approach*. PhD thesis, Universidad de Hamburgo, Zurich, Suiza, 2000.