MOPSO : Un Algoritmo Multiobjetivo Basado en Optimización Mediante Cúmulos de Partículas

Carlos A. Coello Coello† and Maximino Salazar Lechuga‡

Resumen— **Este artículo introduce una propuesta para extender la heurística denominada "optimización mediante cúmulos de partículas" (*particle swarm optimization*, o PSO) de forma que pueda resolver problemas de optimización multiobjetivo. Nuestro método usa el concepto de dominancia de Pareto para determinar la dirección de vuelo de una partícula y mantiene vectores no dominados encontrados previamente en un repositorio global que después es utilizado por otras partículas para guiar su vuelo. El método es validado usando diversas funciones de prueba estándares halladas en la literatura especializada. Nuestros resultados indican que el método propuesto es altamente competitivo con respecto a las técnicas evolutivas de optimización multiobjetivo actuales.**

Palabras clave— **optimización evolutiva multiobjetivo, optimización mediante cúmulos de partículas, optimización multiobjetivo.**

I. Introducción

La mayoría de los problemas del mundo real son de naturaleza multiobjetivo. Sin embargo, tienden a ser tratados como problemas de optimización de un solo objetivo, transformando muchos de los objetivos originales en restricciones adicionales. Los algoritmos evolutivos (AEs) son particularmente adecuados para resolver problemas de optimización multiobjetivo (POM) porque son heurísticas poblacionales. Por ende, permiten la generación de varios elementos del conjunto de óptimos de Pareto en una sola ejecución. En contraste, las técnicas de programación matemática tradicionales tienden a generar un solo elemento del conjunto de óptimos de Pareto por cada ejecución realizada. Adicionalmente, los AEs no son susceptibles a la forma o continuidad del frente de Pareto como lo son los métodos tradicionales [1].

El uso de AEs para resolver problemas multiobjetivo (un área llamada optimización evolutiva multiobjetivo) ha generado una considerable cantidad de investigación en los últimos años (ver por ejemplo [7], [1], [17]). El desarrollo de una técnica (heurística) alternativa para optimización multiobjetivo requiere el tener que lidiar con dos problemas principales: cómo generar vectores no dominados que se aproximen al frente de Pareto global del problema y cómo mantener diversidad para que podamos generar tantos vectores como nos sea posible. Para atacar el primero de estos problemas, se ha usado tradicionalmente la denominada jerarquización de Pareto [8]. Esta técnica consiste en clasificar una población de individuos usando como base el concepto de dominancia de Pareto. La jerarquización de Pareto es $O(kM^2)$, donde k es el número de funciones objetivo y M es el tamaño de la población. Adicionalmente, se requiere un mecanismo que permita preservar la diversidad en la población (suele adoptarse una técnica de nichos o de compartición de aptitud [6]). Este mecanismo generalmente implica el uso de otro proceso que es $O(M^2)$.

Conociendo las fuentes de ineficiencia de las técnicas de optimización evolutiva multiobjetivo tradicionales, varios investigadores han enfocado sus esfuerzos recientes en la reducción del chequeo de no dominancia y en el desarrollo de métodos eficientes para mantener diversidad. Respecto al primero de estos puntos, el énfasis principal ha sido utilizar un archivo externo que almacena vectores no dominados encontrados durante el proceso evolutivo. Estos vectores son re-insertados en la población en generaciones posteriores (esto puede ser visto como una forma de elitismo en el contexto de optimización multiobjetivo [9], [18]). Respecto al segundo aspecto, el énfasis principal ha sido usar técnicas de agrupamiento (*crowding*) [4] o métodos basados en posicionamiento geográfico de individuos en una malla adaptativa [13].

En este artículo se presenta una propuesta llamada "Multi-Objective Particle Swarm Optimization" (MOPSO), la cual permite al algoritmo de optimización mediante cúmulos de partículas ser capaz de manejar problemas multiobjetivo. La técnica es relativamente simple de implementar, es poblacional, usa una memoria externa (llamada "repositorio") y un método basado en posicionamiento geográfico para mantener diversidad. MOPSO es validado usando diversas funciones de prueba estándares encontradas en la literatura especializada. La técnica es comparada contra dos algoritmos evolutivos multiobjetivo modernos que son altamente competitivos: la *Pareto Archived Evolution Strategy* [13] (PAES) y el *Non-dominated Sorting Genetic Algorithm II* [4] (NSGA II).

†CINVESTAV-IPN,
Depto. de Ing. Elect./Sección de Computación,
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D. F. 07300,
ccoello@cs.cinvestav.mx

‡Maestría en Inteligencia Artificial,
Sebastián Camacho No. 5,
LANIA-Universidad Veracruzana,
Xalapa, Veracruz, México 91090
msalazar@mail.mia.uv.mx

II. Conceptos Básicos

Definición 1 (MOP General): *Encontrar el vector* $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ *que satisfaga las* m *restricciones de desigualdad:*

$$g_i(\vec{x}) \geq 0 \quad i = 1, 2, \ldots, m \tag{1}$$

las p *restricciones de igualdad*

$$h_i(\vec{x}) = 0 \quad i = 1, 2, \ldots, p \tag{2}$$

y optimice el vector función

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{3}$$

donde $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ *es el vector de las variables de decisión.* □

Definición 2 (Optimalidad de Pareto): *Un punto* $\vec{x}^* \in \Omega$ *(*Ω *es la región factible) es* **óptimo en el sentido de Pareto** *si para cada* $\vec{x} \in \Omega$ *y* $I = \{1, 2, \ldots, k\}$ *tanto,*

$$\forall_{i \in I}(f_i(\vec{x}) = f_i(\vec{x}^*)) \tag{4}$$

o, haya al menos un $i \in I$ *tal que*

$$f_i(\vec{x}) > f_i(\vec{x}^*) \tag{5}$$

□

En palabras, esta definición dice que $\vec{x}^*$ es un óptimo de Pareto si no existe vector $\vec{x}$ factible que haga decrementar algún criterio sin causar un aumento simultáneo en al menos algún otro. El significado de la frase "óptimo de Pareto" se considera con respecto a todo el espacio de las variables de decisión a menos que se indique lo contrario.

Definición 3 (Dominancia de Pareto): *Un vector* $\vec{u} = (u_1, \ldots, u_k)$ *se dice que domina a otro* $\vec{v} = (v_1, \ldots, v_k)$ *(denotado por* $\vec{u} \preceq \vec{v}$*) si y solo si* u *es parcialmente menor que* v*, por ejemplo,* $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$. □

Definición 4 (Conjunto de óptimos de Pareto): *Dado un MOP* $\vec{f}(x)$*, el conjunto de óptimos de Pareto (*$\mathcal{P}^*$*) se define como:*

$$\mathcal{P}^* := \{x \in \Omega \mid \neg\exists\ x' \in \Omega \ \ \vec{f}(x') \preceq \vec{f}(x)\}. \tag{6}$$

□

Definición 5 (Frente de Pareto): *Para un MOP* $\vec{f}(x)$ *y conjunto de óptimos de Pareto* $\mathcal{P}^*$ *dado, el frente de Pareto (*$\mathcal{PF}^*$*) se define como:*

$$\mathcal{PF}^* := \{\vec{u} = \vec{f} = (f_1(x), \ldots, f_k(x)) \mid x \in \mathcal{P}^*\}. \tag{7}$$

□

III. Optimización Mediante Cúmulos de Partículas

Kennedy y Eberhart [10] propusieron un método llamado "optimización mediante cúmulos de partículas" (*particle swarm optimization*) el cual está inspirado en la coreografía de una bandada de aves. La idea de esta técnica es la de simular los movimientos de un grupo (o población) de aves cuyo objetivo es encontrar comida. El método puede ser visto como un algoritmo de comportamiento distribuido que lleva a cabo (en su versión más general) una búsqueda multidimensional. En la simulación, la conducta de cada individuo es afectada por el mejor individuo local y/o por el mejor global. El método usa entonces el concepto de población y una medida de desempeño similar al valor de aptitud adoptado con los algoritmos evolutivos. Los ajustes de los individuos son también análogos al uso de un operador de cruce. Sin embargo, este método hace sobrevolar a las soluciones potenciales a través del hiperespacio a fin de acelerar convergencia. Este mecanismo no parece tener ningún símil en los algoritmos evolutivos tradicionales. Otra diferencia importante es el hecho de que la optimización mediante cúmulos de partículas permite a los individuos beneficiarse de sus experiencias mientras que, en un algoritmo evolutivo, normalmente la población actual es la única "memoria" usada por los individuos. Esta técnica ha sido usada exitosamente para optimización de una sola función objetivo tanto en espacios continuos como en discretos (ha tenido gran éxito sobre todo en el balanceo de pesos de una red neuronal) [10], [11], [12].

La optimización mediante cúmulos de partículas parece particularmente adecuada para la optimización multiobjetivo debido principalmente a la alta velocidad de convergencia que el algoritmo presenta para problemas de optimización con una sola función objetivo [12]. Nuestra hipótesis previa a este trabajo fue que la extensión de este algoritmo para manejar varios objetivos nos proveería de una técnica muy veloz y además relativamente fácil de implementar.

IV. Descripción del método

La analogía de la optimización mediante cúmulos de partículas con los algoritmos evolutivos hace evidente la noción de que usar un esquema como el de jerarquización de Pareto [8] podría permitir extender la técnica de manera sencilla a fin de que pudiese resolver problemas multiobjetivo. El registro histórico de las mejores soluciones encontradas por una partícula (es decir, un individuo) es una opción viable para almacenar soluciones no dominadas generadas en el pasado. El uso de mecanismos globales de atracción combinados con un archivo histórico de vectores no dominados encontrados anteriormente podría motivar convergencia a través de soluciones no dominadas globalmente. Por eso, nuestra propuesta está basada en la idea de tener un repositorio global en el cual cada partícula deposita sus experiencias de vuelo adquiridas después de cada ciclo. Adicionalmente, las actualizaciones al repositorio se realizan considerando un sistema basado en posicionamiento geográfico definido en términos de los valores de la función objetivo de cada individuo. Esta técnica está inspirada en el archivo externo usado por la *Pareto Archived Evolution Strategy* (PAES) [13]. El repositorio antes mencionado es utilizado por las partículas para identificar a un líder que guiará la búsqueda. Hemos implementado un mecanismo tal que cada partícula puede escoger a un guía distinto. Nuestro mecanismo está basado en la generación de hipercubos que se producen dividiendo el espacio de búsqueda explorado. El algoritmo del MOPSO es el siguiente:

1. Inicializamos la población POP:
   (a) Para $i = 0$ hasta MAX /* MAX = número de partículas */
   (b) Inicializa $POP[i]$
2. Inicializamos la velocidad de cada partícula:
   (a) Para $i = 0$ hasta MAX
   (b) Inicializa $VEL[i]$
3. Evaluamos cada una de las partículas en POP.
4. Se guardan las posiciones de las partículas que representen vectores no dominados en el repositorio REP.
5. Se generan hipercubos del espacio de búsqueda explorado hasta el momento, y se localizan las partículas usando estos hipercubos como un sistema de coordenadas en donde la posición de cada partícula es definida de acuerdo a los valores de sus funciones objetivo.
6. Inicializamos la memoria de cada partícula con ella misma (esta memoria servirá como guía para viajar a través del espacio de búsqueda. Esta memoria es también almacenada en el repositorio):
   (a) Para $i = 0$, hasta MAX
   (b) $PBESTS[i] = POP[i]$
7. Mientras no se cumpla con el número de ciclos especificado:
   (a) Calculamos la velocidad de cada partícula[1] con la siguiente fórmula:

$$\begin{aligned} VEL[i] = & W \times VEL[i] + \\ & R_1 \times (PBESTS[i] - POP[i]) + \\ & R_2 \times (REP[h] - POP[i]) \end{aligned} \tag{8}$$

   donde W (el peso de la inercia) toma un valor de 0.4; R_1 y R_2 son números aleatorios en el rango [0..1]; $PBESTS[i]$ es el mejor valor que esa partícula ha tomado[2]; $REP[h]$ es un valor que se toma del repositorio, en donde el índice h es seleccionado de la siguiente manera: a los hipercubos que contengan más de una partícula se les asigna una aptitud (la aptitud que se les otorga es el número de partículas que contengan dividido entre cualquier número x—en este trabajo se adoptó $x = 10$—; esto con la finalidad de que los hipercubos con un mayor número de partículas tengan una aptitud menor que los que albergan a un número menor, y puede ser visto como una forma de compartir aptitudes [6]). Con base en esa aptitud se forma una ruleta para escoger al hipercubo de donde se tomará a la partícula. Una vez que se seleccionó a un hipercubo, se elige aleatoriamente a una partícula dentro del mismo; $POP[i]$ es la posición que tiene actualmente la partícula i en el espacio.
   (b) Se calculan las nuevas posiciones de las partículas sumando la velocidad obtenida en el paso anterior:

$$POP[i] = POP[i] + VEL[i] \tag{9}$$

   (c) Se mantienen las partículas dentro del espacio. Esto es, en caso de que las partículas se hayan salido del hipercubo en el que se encuentran realizando la búsqueda, se reintegran de nuevo (evitando así generar soluciones que no residan en el espacio de búsqueda válido).
   (d) Se evalúan cada una de las partículas POP.
   (e) Se va actualizando el repositorio REP junto con la representación geográfica que existe de las partículas en los hipercubos. La actualización consiste en insertar todas las posiciones actuales en el repositorio en caso de que se traten de posiciones no dominadas, e integrar su posición en el hipercubo correspondiente. En caso de que existan posiciones actuales que dominen a posiciones en el repositorio, las posiciones en el repositorio son eliminadas y se actualiza igualmente el hiperespacio al que pertenecían. En el supuesto de que el repositorio esté a su máxima capacidad, las partículas que se retienen son aquellas que se encuentren en zonas menos pobladas del espacio de búsqueda, con el fin de mantener diversidad.
   (f) Cuando la posición actual de una partícula sea mejor que la que contiene en su memoria, ésta se actualiza de la siguiente manera:

$$PBESTS[i] = POP[i] \tag{10}$$

   El criterio utilizado para decidir qué posición retener en la memoria es simplemente aplicar dominacia de Pareto (por ejemplo, si la posición actual se encuentra dominada por la posición memorizada, se retiene la memorizada; en caso contrario, se actualiza la memoria con la posición actual; en caso de que no sea una mejor o peor posición, se escoge a una de las dos posiciones aleatoriamente).
   (g) Incrementamos el contador del ciclo.
8. Fin del Mientras.

[1] Cada partícula tiene una dimensionalidad que es dependiente del problema. Cuando decimos que calculamos la velocidad de una partícula, nos referimos a que calculamos una velocidad para cada una de sus dimensiones.

[2] Más adelante se explicará el criterio para tomar en cuenta cuál es el mejor valor que esa partícula ha tomado.

V. Comparación de resultados

Diversas funciones de prueba fueron tomadas de la literatura especializada para comparar nuestro método. A fin de poder comparar nuestros resultados de una forma cuantitativa, se adoptaron dos criterios: el tiempo promedio de ejecución del algoritmo (usando el mismo número de evaluaciones de la función objetivo), y la siguiente métrica definida en el espacio de las funciones objetivo por Zitzler et al. [18]:

$$M_1^* = \frac{1}{\mid Y' \mid} \sum_{\mathbf{d}' \in \mathbf{Y}'} \min \left\{ \parallel \mathbf{d}' - \bar{\mathbf{d}} \parallel^*; \bar{\mathbf{d}} \in \bar{\mathbf{Y}} \right\} \quad (11)$$

donde: $Y', \bar{Y} \subseteq Y$ son los conjuntos de vectores objetivo que corresponden a un conjunto de pares de vectores de decisión no dominantes $X', \bar{X} \subseteq X$, respectivamente, y X corresponde a las variables de decisión del problema. Debe ser obvio que M_1^* proporciona la distancia promedio al conjunto de óptimos de Pareto. Es por eso que debemos intentar minimizar este valor (ver [18] para más detalles). Para calcular M_1^*, generamos el frente de Pareto global para cada una de las funciones de prueba (esto se hace mediante enumeración, usando una representación de grano grueso del espacio de variables de decisión), de tal forma que podamos medir la cercanía al conjunto de soluciones deseado. Dado que el principal énfasis del método aquí propuesto ha sido el de incrementar eficiencia, comparamos nuestro MOPSO contra dos algoritmos muy veloces que representan el estado del arte en optimización evolutiva multiobjetivo: NSGA-II [4] y PAES[3] [13].

El pseudo-código del NSGA-II se muestra en la figura 1 [4]. Este algoritmo usa jerarquización de Pareto tal y como lo sugiere Goldberg [8] aplicando este proceso por capas (o niveles) a toda la población. El NSGA-II es una versión mejorada del NSGA [16] en la cual se usa elitismo y un operador de "agrupamiento" (*crowding*) que permite mantener diversidad sin tener que especificar parámetros adicionales (a diferencia de los algoritmos evolutivos multiobjetivo tradicionales [2]). El NSGA-II es un algoritmo ha mostrado ser un algoritmo muy rápido y eficiente en una amplia gama de funciones de prueba como las utilizadas en este artículo, así como en diversos problemas del mundo real [4], [5].

El pseudo-código de PAES se muestra en la figura 2 [13]. PAES consiste de una estrategia evolutiva (1+1) (es decir, se usa un solo padre, el cual genera un solo hijo que compite con su progenitor) combinada con un archivo histórico que registra algunas de las soluciones no dominadas producidas anteriormente. Este archivo se usa como el conjunto de referencia para determinar la no dominancia de un nuevo individuo generado mediante mutación. PAES usa un procedimiento que divide recursivamente el espacio de las funciones objetivo para mantener diversidad. Bajo este esquema, se genera una "rejilla" la cual sirve como un sistema de coordenadas en las que las funciones objetivo sirven como los valores mediante los cuales se ubica cada individuo en ella. PAES mantiene un mapa de esta rejilla para decidir qué individuos retener y cuáles eliminar. Los criterios son la no dominancia y la densidad de población en cada región del espacio de las funciones objetivo. Dada la extrema simplicidad de PAES (conceptualmente, tal vez sea el algoritmo más simple que pueda concebirse para optimización multiobjetivo) y a su esquema para mantener diversidad, esta técnica es muy eficiente (computacionalmente hablando). Así mismo, PAES ha probado ser muy competitivo con respecto a algoritmos tradicionales de optimización evolutiva multiobjetivo [13] y se le ha usado también en algunos problemas del mundo real [14].

En los siguientes ejemplos, el NSGA-II fue ejecutado usando una población de tamaño 200, un porcentaje de cruce de 0.8, selección mediante torneo, y un porcentaje de mutación de $1/vars$, en donde $vars$ es igual al número de variables de decisión del problema. En los siguientes ejemplos, PAES fue ejecutado usando una profundidad de 5, un tamaño de archivo de 200, y un porcentaje de mutación de $1/L$, donde L se refiere a la longitud de la cadena del cromosoma que codifica las variables de decisión.

Nuestro MOPSO usó una población de 40 partículas (excepto por la función 2, en la que se usaron sólo 20 partículas). El repositorio de MOPSO tiene capacidad para 200 partículas y se definió un valor de 30 para el número de divisiones de la rejilla adaptativa. Para permitir una comparación justa en tiempos de ejecución, todos los experimentos se llevaron a cabo en un PC con un procesador Pentium II a 333 MHz, 128 Mb en RAM y un disco duro de 6 Gbytes. Todos los algoritmos se compilaron utilizando GNU C ejecutado en la plataforma Linux Red Hat versión 7.1.

A. *Función de Prueba 1*

Nuestra Primera función de prueba fue propuesta por Deb [3]:

$$\text{Min } f_1(x_1, x_2) = x_1 \quad (12)$$

$$\text{Min } f_2(x_1, x_2) = (1 + 10x_2) \times \left[1 - \left(\frac{x_1}{1+10x_2}\right)^\alpha - \frac{x_1}{1+10x_2}\sin(2\pi q x)\right] \quad (13)$$

donde:

$$0 \leq x_1, x_2 \leq 1 \quad (14)$$

y $q = 4$, $\alpha = 2$. Este problema tiene un frente de Pareto consistente en 4 segmentos desconectados.

Las gráficas de la figura 3 muestran los frentes de Pareto producidos por el NSGA-II, PAES y

[3]El código fuente del NSGA-II y PAES (versiones original de sus autores correspondientes) pueden ser obtenidas del repositorio localizado en: `http://www.lania.mx/~ccoello/EMOO/EMOOsoftware.html`

```
Inicializar la población
    Generar una población aleatoria de tamaño M
    Evaluar las funciones objetivo
    Asignar jerarquías (niveles) con base en dominancia de Pareto - “ordenar”
    Generar población de hijos
        Efectuar selección mediante torneo binario
        Efectuar recombinación y mutación
For i = 1 to Número máximo de generaciones
    Haz lo siguiente con la generación de padres e hijos
        Asignar jerarquías (niveles) con base en dominancia de Pareto - “ordenar”
        Generar conjuntos de frentes no dominados
        Iterar (en el ciclo interno) agregando soluciones a
        la siguiente generación comenzando con el “primer” frente,
        hasta tener clasificada a toda la población
        Determinar la distancia de “agrupamiento” entre los puntos de cada frente
    Seleccionar puntos (elitismo) en el frente inferior
    de manera que se respete un cierto valor para el factor de agrupamiento
    Crear la siguiente generación
        Efectuar selección mediante torneo binario
        Efectuar recombinación y mutación
    Incrementar generación
    Fin del ciclo exterior
```

Fig. 1. Pseudo-código del NSGA-II

```
Generar un padre p aleatoriamente y agregarlo al archivo
Mutar p para producir un hijo c y evaluar su aptitud
    if (p domina a c) desechar c
    else if (c domina a p)
        reemplazar p con c, y agregar c al archivo
    else if (if c es dominado por cualquier otro elemento del archivo)
        desechar c
    else aplicar evaluación (p,c,Archivo) para determinar quién
        será la siguiente solución y si se agrega o no a c al archivo
Hasta que se cumpla la condición de detención, regresar a la línea 2
```

Fig. 2. Pseudo-código de PAES

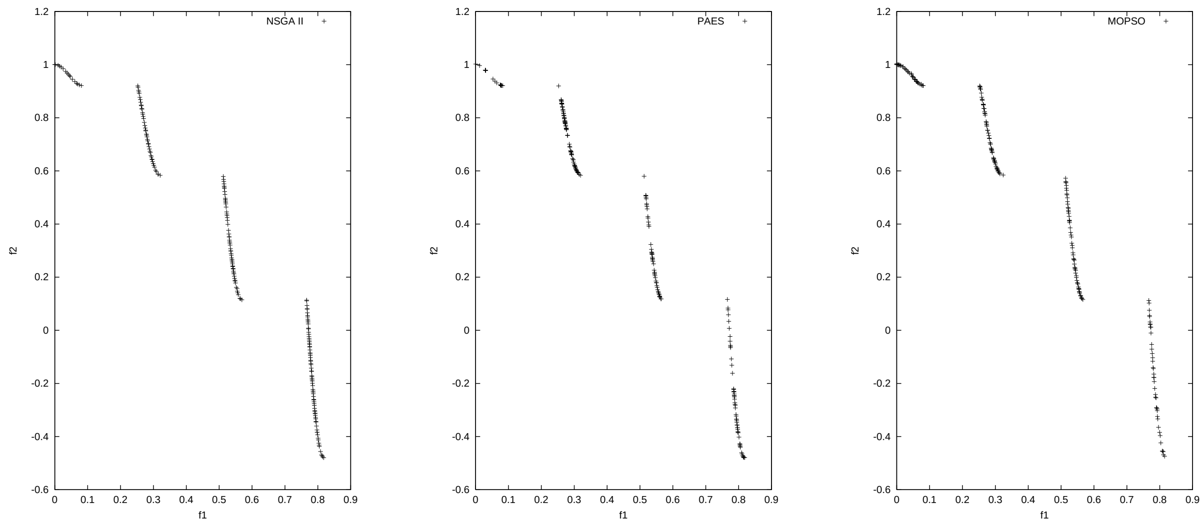

Fig. 3. Frentes de Pareto producidos por el NSGA II (izquierda), PAES (centro) y MOPSO (derecha) para la primera función de prueba

MOPSO, respectivamente para la primera función de prueba. Para este ejemplo, los tres algoritmos realizaron 4000 evaluaciones de la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 ejecuciones independientes) fueron los siguientes: 2.402 segundos para el NSGA-II, 2.80 para PAES y sólo 0.0675 segundos para nuestro MOPSO. Los valores promedio de la métrica M_1^* fueron los siguientes: 0.002536 para el NSGA-II (con una desviación estándar de 0.000138), 0.002881 para PAES (con una desviación estándar de 0.00213), y 0.002092 para MOPSO (con una desviación estándar de 0.000239).

B. Función de Prueba 2

Nuestra segunda función de prueba fue propuesta por Schaffer [15]:

$$\text{Min } f_1(x) = \begin{cases} -x & \text{si } x \leq 1 \\ -2+x & \text{si } 1 < x \leq 3 \\ 4-x & \text{si } 3 < x \leq 4 \\ -4+x & \text{si } x > 4 \end{cases} \tag{15}$$

$$\text{Min } f_2(x) = (x-5)^2 \tag{16}$$

donde: $-5 \leq x \leq 10$.

Este problema también tiene un frente de Pareto discontinuo. Las gráficas de la figura 4 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente para la segunda función de prueba. Para este ejemplo, los tres algoritmos realizaron 1200 evaluaciones a la función objetivo. El promedio de los tiempos de ejecución de cada algoritmo (20 ejecuciones independientes) fueron los siguientes: 0.812 segundos para el NSGA-II, 0.339 segundos para PAES y sólo 0.047 segundos para MOPSO. Los valores promedio de la métrica M_1^* fueron los siguientes: 0.001594 para el NSGA-II (con una desviación estándar de 0.000122), 0.070003 para PAES (con una desviación estándar de 0.158081), y 0.001739 para MOPSO (con una desviación estándar de 0.000499).

C. Función de Prueba 3

Nuestro tercer ejemplo fue propuesto por Deb [3]:

$$\text{Minimizar } f_1(x_1, x_2) = x_1 \tag{17}$$

$$\text{Minimizar } f_2(x_1, x_2) = g(x_1, x_2) \cdot h(x_1, x_2) \tag{18}$$

donde:

$$g(x_1, x_2) = 11 + x_2^2 - 10 \cdot \cos(2\pi x_2) \tag{19}$$

$$h(x_1, x_2) = \begin{cases} 1 - \sqrt{\frac{f_1(x_1, x_2}{g(x_1, x_2}} & \text{si } f_1(x_1, x_2) \leq g(x_1, x_2) \\ 0 & \text{en otro caso} \end{cases} \tag{20}$$

y $0 \leq x_1 \leq 1$, $-30 \leq x_2 \leq 30$. Este problema tiene 60 frentes de Pareto locales a los que la población podría ser fácilmente atraída.

Las gráficas de la figura 5 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente para la tercera función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 3200 evaluaciones a la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 ejecuciones independientes) son los siguientes: 2.1165 segundos para el NSGA-II, 1.641 segundos para PAES y sólo 0.1695 segundos para MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.094644 para el NSGA-II (con una desviación estándar de 0.117608), 0.259664 para PAES (con una desviación estándar de 0.573286), y 0.000842 para MOPSO (con una desviación estándar de 0.000261).

D. Función de Prueba 4

Nuestro cuarto ejemplo fue propuesto por Deb [3]:

$$\text{Minimizar } f_1(x_1, x_2) = x_1 \tag{21}$$

$$\text{Minimizar } f_2(x_1, x_2) = \frac{g(x_2)}{x_1} \tag{22}$$

donde:

$$g(x_2) = 2.0 - \exp\left\{-\left(\frac{x_2 - 0.2}{0.004}\right)^2\right\} - 0.8\exp\left\{-\left(\frac{x_2 - 0.6}{0.4}\right)^2\right\} \tag{23}$$

y $0.1 \leq x_1 \leq 1.0$, $0.1 \leq x_2 \leq 1.0$.

Las gráficas de la figura 6 muestran los frentes de Pareto producidos por el NSGA-II, PAES y MOPSO, respectivamente, para la cuarta función de prueba. Para este ejemplo, los tres algoritmos llevaron a cabo 12000 evaluaciones a la función objetivo. Los tiempos promedio de ejecución de cada algoritmo (20 ejecuciones) fueron los siguientes: 6.86 segundos para el NSGA-II, 3.04 segundos para PAES y sólo 0.82 segundos para MOPSO. Los valores promedio para la métrica M_1^* fueron los siguientes: 0.06 para el NSGA-II (con una desviación estándar de 0.01), 0.18 para PAES (con una desviación estándar de 0.31), y 0.40 para MOPSO (con una desviación estándar de 0.33).

VI. Discusión de Resultados

En todas las funciones de prueba utilizadas (probamos nuestro método con otras funciones que no se incluyen en este artículo debido a las limitantes de espacio), el MOPSO tuvo un desempeño razonablemente bueno con respecto al NSGA II y a PAES en términos de la métrica usada (puede verse que no fue el mejor en todos los casos, pero ha tendido a producir resultados razonables en la mayoría de los problemas en los que se ha usado) y sus tiempos de ejecución han resultado inferiores a los otros dos algoritmos en todos los casos. Más aún, la representación gráfica de los frentes de Pareto producidos en cada caso también muestran que MOPSO produce soluciones con una distribución más uniforme a lo largo del frente del Pareto que los competitivas con respecto a los otros algoritmos utilizados (las distribuciones producidas por MOPSO a lo largo de

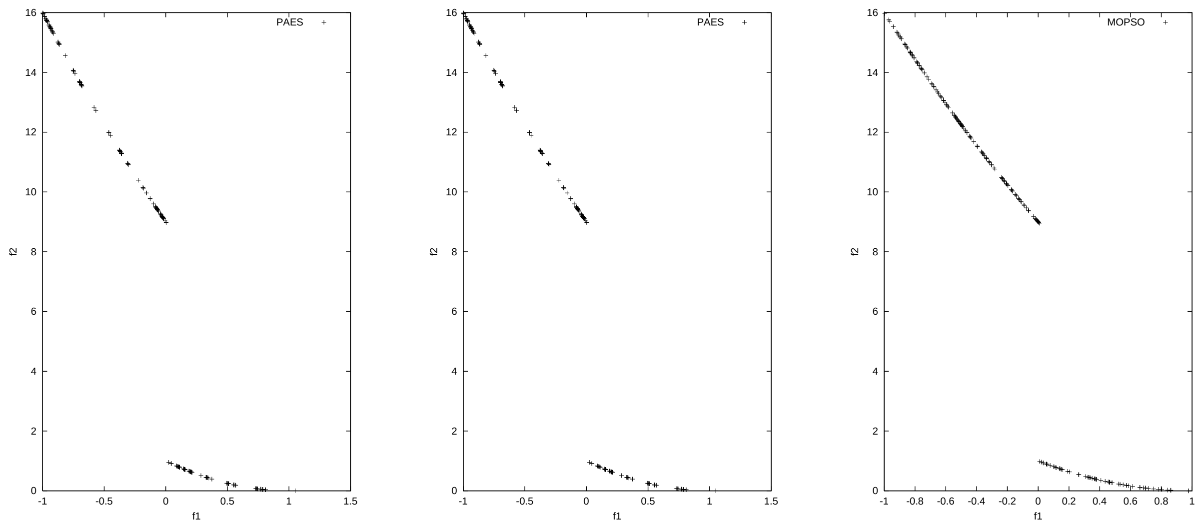

Fig. 4. Frentes de Pareto producidos por el NSGA II (izquierda), PAES (centro) y MOPSO (derecha) para la segunda función de prueba

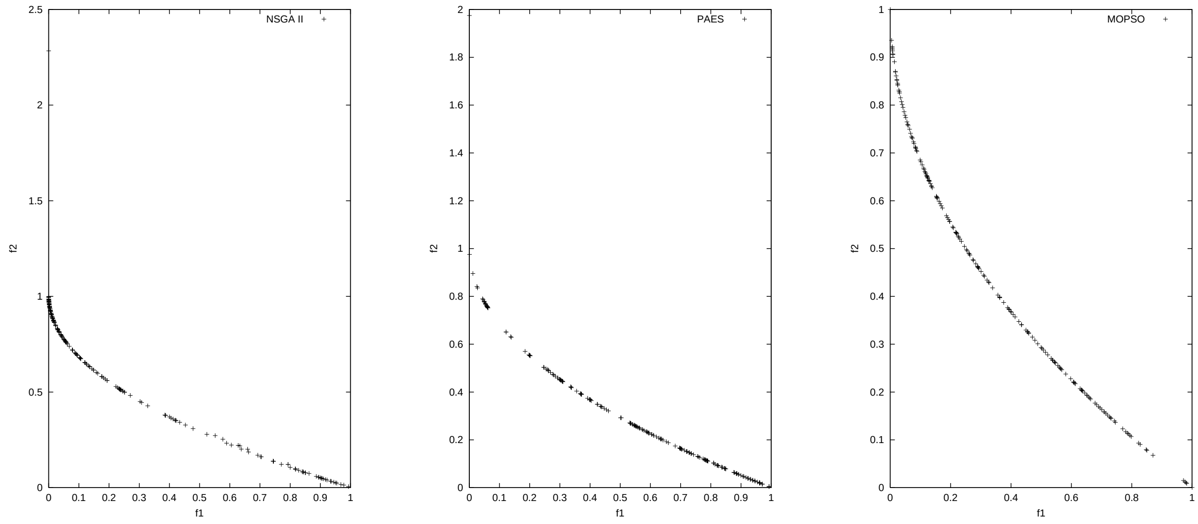

Fig. 5. Frentes de Pareto producidos por el NSGA II (izquierda), PAES (centro) y MOPSO (derecha) para la tercera función de prueba

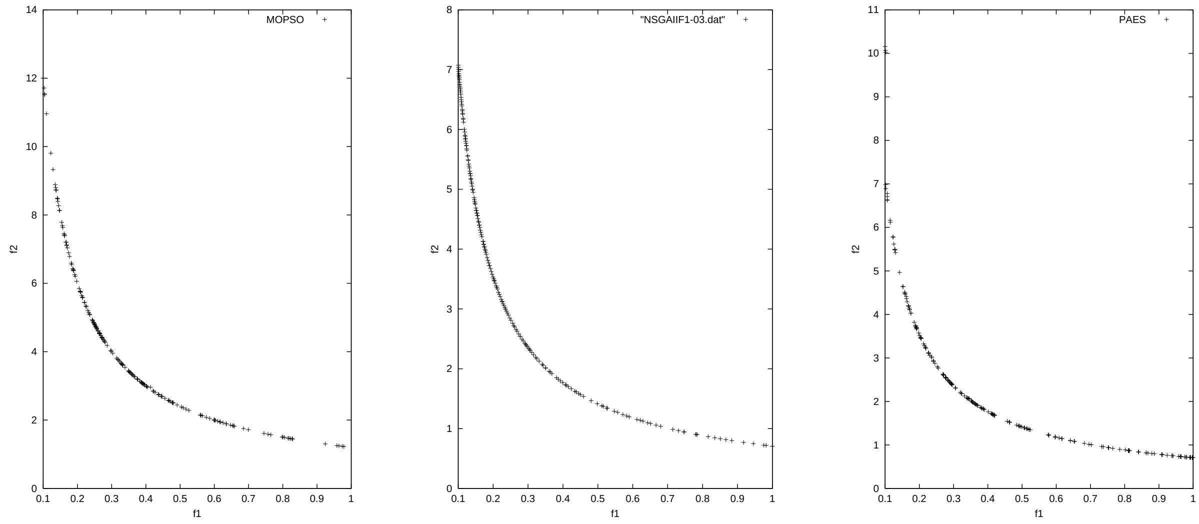

Fig. 6. Frentes de Pareto producidos por el NSGA-II (izquierda), PAES (centro), y nuestro MOPSO (derecha) para la cuarta función de prueba.

los frentes de Pareto obtenidos son bastante buenas, como se aprecia en las figuras 3, 4, 5 y 6).

Finalmente, queremos mencionar brevemente cuales son los parámetros usados por MOPSO y algunas sugerencias de cómo ajustar sus valores correspondientes:

- **Número de partículas**: Esto es equivalente al tamaño de población de un algoritmo genético. Se recomienda usar entre 20 y 80 partículas.

- **Número de ciclos**: Este parámetro está ligado con el número de partículas. La relación tiende a ser inversamente proporcional (es decir, a mayor número de partículas, menor número de ciclos y viceversa). Se recomienda usar entre 80 y 120.

- **Número de divisiones**: Esto nos permite determinar el número de hipercubos que serán generados en el espacio de las funciones objetivo. Recomendamos usar entre 30 y 50 divisiones.

- **Tamaño del repositorio**: Este parámetro se usa para delimitar el máximo número de vectores no dominados que pueden ser almacenados en el repositorio. El valor de este parámetro determina la calidad del frente de Pareto que se producirá.

VII. Conclusiones y Trabajo Futuro

Es importante indicar que la optimización mediante cúmulos de partículas es una técnica de búsqueda sin restricciones. Es por eso que es necesario desarrollar un mecanismo adicional para trabajar con problemas de optimización multiobjetivo restringidos. En este artículo sólo nos hemos enfocado en la generación de vectores no dominados y en el mecanismo para mantener diversidad, pero no hemos abordado la solución de funciones de prueba restringidas. La extensión para manejar problemas restringidos es, de cualquier forma, no muy difícil de desarrollar y la estamos llevando a cabo en estos momentos.

Un análisis de sensibilidad está también en desarrollo, para de esta forma poder determinar el papel de cada uno de los parámetros usados en el desempeño del algoritmo.

Agradecimientos

Agradecemos los comentarios y sugerencias de los dos revisores anónimos, los cuales contribuyeron a mejorar significativamente el contenido de este artículo.

El primer autor agradece el apoyo del CONACyT a través del proyecto 34201-A. El segundo autor agradece el apoyo del CONACyT a través de una beca para cursar estudios de posgrado en la Maestría en Inteligencia Artificial del LANIA y la Universidad Veracruzana.

Referencias

[1] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

[2] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Publishers, Boston, 2002. (in Press).

[3] Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

[4] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000.

[5] Kalyanmoy Deb and Tushar Goel. A Hybrid Multi-Objective Evolutionary Approach to Engineering Shape Design. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 385–399. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[6] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[7] Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[9] Hisao Ishibuchi and Tadahiko Murata. Multi-Objective Genetic Local Search Algorithm. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 119–124, Nagoya, Japan, 1996. IEEE.

[10] James Kennedy and Russell C. Eberhart. Particle Swarm Optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, New Jersey, 1995. IEEE Service Center.

[11] James Kennedy and Russell C. Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the 1997 IEEE Conference on Systems, Man, and Cybernetics*, pages 4104–4109, Piscataway, New Jersey, 1997. IEEE Service Center.

[12] James Kennedy and Russell C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, San Francisco, California, 2001.

[13] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[14] Joshua D. Knowles, Martin J. Oates, and David W. Corne. Multiobjective Evolutionary Algorithms Applied to two Problems in Telecommunications. *BT Technology Journal*, 18(4):51–64, October 2000.

[15] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

[16] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, fall 1994.

[17] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 7(3):1–26, 2000.

[18] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.