Optimización de Enjambre de Partículas aplicada al Problema del Cajero Viajante Bi-objetivo

Joaquín Q. Lima M., Benjamín Barán C.

Universidad Nacional de Asunción
Campus Universitario, San Lorenzo – Paraguay
joaquinlima@gmail.com
bbaran@cnc.una.py

Resumen

La metaheurística de Optimización de Enjambre de Partículas (PSO) ha sido exitosamente utilizada en la resolución de problemas de optimización con espacios de búsqueda continuos. En este trabajo se presentan los resultados de la aplicación de esta metaheurística al Problema del Cajero Viajante (TSP) bi-objetivo. Para ello se implementaron algunas propuestas de enfoques de la metaheurística PSO para problemas multiobjetivos (MOPSO) y se realizó una comparación entre los resultados conseguidos por estos enfoques y los resultados obtenidos al aplicar algoritmos de la metaheurística de Optimización de Colonias de Hormigas (ACO) para problemas multiobjetivos (MOACO). Como resultado, queda demostrado que el uso de técnicas MOPSO junto con las adaptaciones aplicadas constituyen una buena alternativa para la resolución del TSP bi-objetivo.

Palabras clave: Optimización de Enjambre de Partículas, Optimización Multiobjetivo, Problema del Cajero Viajante.

1. Introducción

1.1. Problema del Cajero Viajante

El Problema del Cajero Viajante o TSP por sus siglas en inglés (*Traveling Salesman Problem*) es posiblemente uno de los problemas más utilizados para probar la efectividad de los distintos métodos de optimización en la resolución de problemas combinatorios.

Formalmente, el TSP se define como el problema de encontrar en un grafo dado $G = (V, E)$ completamente conexo el ciclo de menor costo, que inicie en un vértice cualquiera, pase una vez por cada uno de los demás vértices y finalice en el vértice inicial. En donde $V = \{v_1, v_2, ..., v_N\}$ es el conjunto de vértices, N es el número de vértices, $E = \{(v_i, v_j): \forall v_i, v_j \in V\}$ es el conjunto de aristas y cada arista (v_i, v_j) tiene asociada un número real $c_{ij} \geq 0$ que representa el costo de atravesar dicha arista.

Una solución S a un problema TSP puede ser vista como una permutación del conjunto de vértices V, la cual induce una selección de N aristas del conjunto de aristas E, denotada como $S \rightarrow E$. Así, la función de costo para cualquier solución S, denotada por $costo(S)$, se define como la sumatoria de los costos asociados a cada arista en la selección de aristas inducida por tal solución, la cual se expresa como:

$$costo(S) = \sum c_{ij} \quad \forall (v_i, v_j) \in S \rightarrow E. \qquad (1)$$

En problemas TSP de M objetivos, cada arista (v_i, v_j) tiene asociada un vector de costo $\vec{c}_{ij} \geq 0$ de dimensión M, en donde cada componente $1 \leq k \leq M$ de este vector indica el costo de atravesar dicha arista se-

gún el objetivo *k*. Así, la función de costo para el objetivo *k*, denotada por $costo_k(S)$, se define como:

$$costo_k(S) = \sum c_{ij}^k \quad \forall (v_i, v_j) \in S \rightarrow E, \qquad (2)$$

donde c_{ij}^k es el valor de la componente *k* del vector de costo $\vec{c}_{ij}$ correspondiente a la arista (v_i, v_j).

En instancias de problemas TSP multiobjetivo, el costo de una solución *S* en todos los objetivos viene dado por la función vectorial $\overrightarrow{costo}(S)$ definida como:

$$\overrightarrow{costo}(S) = [costo_1(S), costo_2(S), ..., costo_M(S)] \qquad (3)$$

1.2 Optimización de Enjambre de Partículas

La metaheurística de Optimización de Enjambre de Partículas o PSO por sus siglas en inglés (*Particle Swarm Optimization*), fue desarrollada por Kennedy y Eberhart [Kennedy95] y está inspirada en el comportamiento social observado en grupos de individuos tales como parvadas de pájaros, enjambres de insectos y bancos de peces.

Tal comportamiento social se basa en la transmisión del suceso de cada individuo a los demás individuos del grupo, lo cual resulta en un proceso sinergético que permite a los individuos satisfacer de la mejor manera posible sus necesidades más inmediatas, tales como la localización de alimentos o de un lugar de cobijo.

La metaheurística PSO ha mostrado ser muy eficiente para resolver problemas de optimización de un sólo objetivo con rápidas tasas de convergencia [Kennedy01], haciendo atractiva la idea de su aplicación en la resolución problemas de optimización de múltiples objetivos.

Básicamente, la metaheurística PSO consiste en un algoritmo iterativo basado en una población de individuos denominada *enjambre*, en la que cada individuo, llamado *partícula,* se dice que sobrevuela el espacio de decisión en busca de soluciones óptimas.

Así, dado un espacio de decisión *N*-dimensional, cada partícula *i* del enjambre conoce su posición actual $X_i = [x_{i1}, x_{i2}, \ldots, x_{iN}]$, la velocidad $V_i = [v_{i1}, v_{i2}, \ldots, v_{iN}]$ con la cual ha llegado a dicha posición y la mejor posición $P_i = [p_{i1}, p_{i2}, \ldots, p_{iN}]$ en la que se ha encontrado, denominada *mejor personal*. Además, todas las partículas conocen la mejor posición encontrada dentro del enjambre $G = [g_1, g_2, \ldots, g_N]$, denominada *mejor global*. Existe otra variante en la que se definen sobre el enjambre subgrupos de partículas, posiblemente solapados, a los que se denominan *vecindades*, en tal caso las partículas también conocen la mejor posición encontrada dentro de su vecindad $L_i = [l_{i1}, l_{i2}, \ldots, l_{iN}]$, a la que se denomina *mejor local*.

Suponiendo el uso de la información proveniente del mejor global, en cada iteración *t* del algoritmo PSO, cada componente *j* de la velocidad y la posición de cada partícula *i* del enjambre se actualiza conforme a:

$$v_{ij}^{t+1} = \omega \times v_{ij}^t + C_1 \times rand(\,) \times (p_{ij}^t - x_{ij}^t) + C_2 \times rand(\,) \times (g_j^t - x_{ij}^t), \qquad (4)$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1}, \qquad (5)$$

donde ω es el *parámetro de inercia* que regula el impacto de las velocidades anteriores en la nueva velocidad de la partícula, C_1 es el *parámetro cognitivo* que indica la influencia máxima de la mejor experiencia individual de la partícula en su nueva velocidad y C_2 es el *parámetro social* que indica la influencia máxima de la información social en el nuevo valor de velocidad de la partícula. Mientras que, *rand*() es una función que retorna un número aleatorio en el intervalo [0, 1], mediante el cual se determina la influencia real de las informaciones individual y social en la nueva velocidad para la partícula. La actualización de una partícula se ilustra de manera general en la figura 1.

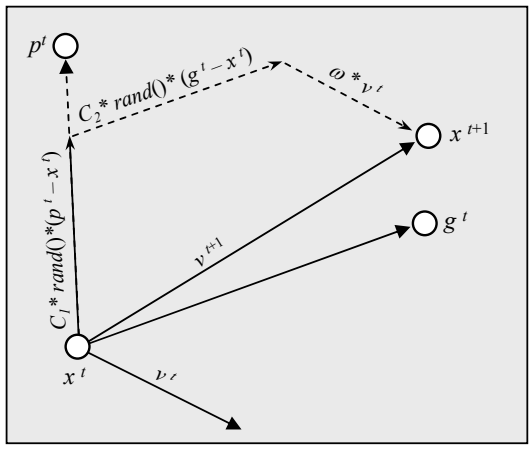

Figura 1. Actualización de una partícula

Típicamente a ω se asigna un valor fijo de 0.8 y en otros casos se le asigna un valor inicialmente entre 1 y 1.5 que se hace decrecer durante la ejecución del

algoritmo. A los pesos C_1 y C_2 generalmente se les asigna el valor 2. Para obtener una mayor información acerca de la influencia de estos parámetros en la efectividad del algoritmo PSO el lector puede referirse a [Shi98, Beielstein02].

Cuando se opta por utilizar la información local proveniente de las vecindades para actualizar las partículas, de manera a evitar que todas ellas sean influenciadas por una única mejor posición, la ecuación (4) se convierte en:

$$v_{ij}^{t+1} = \omega \times v_{ij}^{t} + C_1 \times rand(\,) \times \left(p_{ij}^{t} - x_{ij}^{t}\right) + C_2 \times rand(\,) \times \left(l_{ij}^{t} - x_{ij}^{t}\right), \quad (6)$$

En la Figura 2 puede observarse el algoritmo PSO que utiliza la información del mejor global para realizar una optimización mono-objetivo.

1.3. Organización del trabajo

El resto del presente del trabajo se encuentra organizado como sigue: en la sección 2 se presenta un repaso de la formulación matemática de los problemas de optimización multiobjetivo. La sección 3 consiste en un resumen de algunos enfoques que tratan la aplicación de la metaheurística PSO en la resolución de problemas multiobjetivos. En la sección 4 se presenta la adaptación aplicada a la metaheurística PSO para resolver el TSP. En la sección 5 se exponen los resultados de las comparaciones entre las soluciones encontradas por los algoritmos basados en la metaheurística PSO y las soluciones conseguidas al aplicar algoritmos basados en la metaheurística de Optimización de Colonias de Hormigas (ACO). Finalmente, en la sección 6 se exponen las conclusiones de este trabajo y se presentan propuestas de trabajos futuros.

```
/* Inicializar los parámetros */
C1 := 2;    /* Parámetro cognitivo     */
C2 := 2;    /* Parámetro social        */
ω  := 0.8;  /* Parámetro de inercia    */
g  := 1;    /* Índice del mejor global */
iter := 1;  /* Contador de iteraciones */
max_iter := 10000;  /* Número máximo de iteraciones */
nro_part := 20;     /* Número de partículas         */
FOR i := 1 TO nro_part DO  /* Inicializar las partículas */
   FOR j := 1 TO N DO  /* Para cada componente */
       part[i].pos[j] := random_pos();
       part[i].vel[j] := random_vel();
   END_FOR
END_FOR
WHILE(iter <= max_iter) DO    /* Realizar las iteraciones */
   FOR i := 1 TO nro_part DO  /* Evaluar las partículas   */
       /* Calcular la función objetivo */
       part[i].eval := evaluar(part[i]);
       /* Guardar mejor posición de la partícula */
       IF (part[i].eval ES_MEJOR_QUE mejor[i].eval) THEN
          mejor[i].pos[] := part[i].pos[]; /* copia vectorial */
          mejor[i].eval  := part[i].eval;
          /* Determinar el mejor global */
          IF(part[i].eval ES_MEJOR_QUE mejor[g].eval) THEN
             g := i;
          END_IF
       END_IF
    END_FOR
    FOR i := 1 TO nro_part DO  /* Actualizar las partículas */
       FOR j := 1 TO N DO  /* Para cada componente */
          part[i].vel[j] := ω * part [i].vel[j] +
             C1 * rand() * (mejor[i].pos[j] - part[i].pos[j]) +
             C2 * rand() * (mejor[g].pos[j] - part[i].pos[j]);
          part[i].pos[j] :=  part[i].pos[j] + part[i].vel[j];
       END_FOR
    END_FOR
    iter := iter + 1;
END_WHILE
retornar(mejor[g]);
```

Figura 2. Algoritmo PSO para realizar una optimización mono-objetivo.

2. Optimización Multiobjetivo

Los problemas de optimización multiobjetivo buscan soluciones que optimicen simultáneamente M objetivos, los cuales pueden ser contradictorios e inconmensurables entre si [Veldhiuzen99, Zitzler00] Estos objetivos se evalúan mediante M funciones objetivo de la forma:

$$y_k = f_k(\vec{x}) \quad k = 1,...,M, \tag{7}$$

donde $\vec{x}$ es un vector de dimensión N, en el que cada componente indica el valor de un parámetro o variable de decisión del problema. Estos parámetros pueden estar sujetos a J restricciones de la forma:

$$r_j(\vec{x}) \geq 0 \quad j = 1,...,J. \tag{8}$$

De esta manera, un problema de optimización multiobjetivo puede expresarse formalmente como:

$$\begin{aligned} &\text{Optimizar} \quad \vec{y} = \vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})] \\ &\text{Sujeto a} \quad \vec{r}(\vec{x}) = [r_1(\vec{x}), r_2(\vec{x}), \ldots, r_J(\vec{x})] \geq 0, \end{aligned} \tag{9}$$

siendo $\vec{x} = [x_1, x_2, \ldots, x_N]$ el vector de decisión, $\vec{y} = [y_1, y_2, \ldots, y_M]$ el vector objetivo o evaluación correspondiente a $\vec{x}$ y *Optimizar* se refiere indistintamente a la *minimización* o *maximización* de todas las funciones objetivo.

Al momento de comparar las soluciones surge el concepto de *dominancia Pareto*. Sin pérdida de generalidad, para un caso de minimización se dice que un vector decisión $\vec{u}$ *domina o es mejor* que otro vector decisión $\vec{v}$, denotado por $\vec{u} \succ \vec{v}$, sí y sólo sí:

$$\begin{aligned} &f_i(\vec{u}) \leq f_i(\vec{v}) \ \ \forall\, i \in \{1, \ldots, M\} \ \text{ y además} \\ &f_i(\vec{u}) < f_i(\vec{v}) \ \ \text{para al menos un valor de } i. \end{aligned} \tag{10}$$

Por otra parte, si $\vec{u}$ no domina a $\vec{v}$ y tampoco $\vec{v}$ domina a $\vec{u}$, se dice que son soluciones *no dominadas, no comparables o igualmente factibles*, lo que se denota como $\vec{u} \sim \vec{v}$.

El conjunto de todas las soluciones en el espacio de decisión para las cuales no existen soluciones que las dominen se conoce como *Conjunto Pareto* y a la imagen de este conjunto en el espacio objetivo se la conoce como *Frente Pareto*.

Así, el objetivo de los métodos de optimización multiobjetivo basados en el enfoque de dominancia Pareto es encontrar el Conjunto Pareto del problema, o al menos una buena aproximación a este conjunto. El cual luego será la entrada del proceso de *Toma de Decisiones*, en el que se elige a una o más soluciones para ser aplicadas al caso práctico del problema.

3. Optimización de Enjambre de Partículas para Problemas Multiobjetivos

Adaptar el PSO para realizar una optimización multiobjetivo requiere que en la actualización de las partículas el mejor global sea una solución no dominada encontrada por el grupo y que el mejor personal corresponda a una solución no dominada encontrada por la partícula. En la Figura 3 se muestra el algoritmo PSO genérico para realizar una optimización multiobjetivo.

A continuación se presentan algunos de los enfoques que tratan la aplicación de la metaheurística PSO para resolver problemas de optimización multiobjetivo publicados en la literatura y denominados MOPSO (*MultiObjective Particle Swarm Optimization*).

3.1. Moore y Chapman (1999)

Moore y Chapman [Moore99] proponen un algoritmo MOPSO en el cual se reemplaza el mejor personal de cada partícula por una lista que contiene todas las soluciones no dominadas entre si que ella ha encontrado. Cada vez que la partícula se mueve a una nueva posición comprueba si la solución correspondiente es no dominada por los elementos de su lista y de serla es agregada a su lista, eliminándose todas las soluciones que resulten dominadas por esta nueva solución.

Para su actualización, cada partícula toma como mejor personal a cualquier elemento de su lista y como mejor global a cualquier elemento de su lista no dominado por las soluciones en las listas de las partículas del grupo o de su vecindad (en caso de utilizar un esquema de vecindades).

3.2. Ray y Liew (2002)

Ray y Liew [Ray02] proponen un algoritmo MOPSO basado en un *Ranking* Pareto de los individuos del enjambre. En tal enfoque, se mantiene un conjunto de soluciones llamado *Set Of Leaders* (SOL), el cual es actualizado conforme al *ranking* de los individuos del enjambre.

```
/* Inicializar los parámetros */
C1 := 2;    /* Parámetro cognitivo     */
C2 := 2;    /* Parámetro social        */
ω  := 0.8;  /* Parámetro de inercia    */
iter := 1;  /* Contador de iteraciones */
max_iter := 10000;  /* Número máximo de iteraciones */
nro_part := 20;     /* Número de partículas         */
FOR i := 1 TO nro_part DO  /* Inicializar las partículas */
   FOR j := 1 TO N DO  /* Para cada componente */
      part[i].pos[j] := random_pos();
      part[i].vel[j] := random_vel();
   END_FOR
END_FOR
WHILE (iter <= max_iter) DO  /* Realizar las iteraciones */
   FOR i := 1 TO nro_part DO  /* Evaluar las partículas */
      /* Calcular la función objetivo vectorial */
      part[i].eval[] := evaluar(part[i]);
      IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_global) THEN
         incluir_particula(rep_global, part[i]);
         eliminar_dominados(rep_global);
      END_IF
      IF (part[i] NO_ES_DOMINADA_POR P, ∀P ∈ rep_personal[i]) THEN
         incluir_particula(rep_personal[i], part[i]);
         eliminar_dominados(rep_personal[i]);
      END_IF
   END_FOR
   FOR i := 1 TO nro_part DO /* Actualizar las partículas */
      /* Obtener un mejor global y un mejor personal para
      la partícula i */
      mejor_global := selec_part(rep_global, part[i]);
      mejor_personal := selec_part(rep_personal[i], part[i]);
      /* Actualizar la posición de la partícula */
      FOR j := 1 TO N DO  /* Para cada componente */
         part[i].vel[j] := ω * part[i].vel[j] +
            C1 * rand() * (mejor_global.pos[j] - part[i].pos[j]) +
            C2 * rand() * (mejor_personal.pos[j]  - part[i].pos[j]);
         part[i].pos[j] :=  part[i].pos[j] + part[i].vel[j];
      END_FOR
   END_FOR
   iter := iter + 1;
END_WHILE
Retornar_soluciones_no_dominadas(rep_global);
```

Figura 3. Algoritmo PSO genérico para realizar una optimización multiobjetivo.

Para cada individuo en el conjunto SOL se calcula un radio de *crowding* en el espacio objetivo cuyo valor estima la cantidad de individuos a su alrededor.

En las actualizaciones, cada partícula selecciona como mejor global a un individuo del conjunto SOL mediante el método de la ruleta, utilizando el radio de *crowding* de los individuos del conjunto como valor de adaptabilidad.

Cada partícula mantiene como mejor personal a la última posición no dominada encontrada, la cual es reemplazada al encontrar una nueva solución que la domina o con la cual resulte no comparable.

3.3 Hu y Eberhart (2002)

Hu y Eberhart [Hu02] proponen un algoritmo MOPSO en el cual cada miembro del enjambre elige como mejor global a una de *m* partículas previamente seleccionadas del enjambre, donde *m* es un parámetro propio de este enfoque definido a priori.

La selección del mejor global para cada partícula se realiza seleccionando las *m* partículas del enjambre más cercanas en el valor de evaluación de un objetivo especificado para ese efecto. De estas *m* partículas, aquella con mejor evaluación en los objetivos restantes es seleccionada como mejor global para la partícula en cuestión.
Cada partícula mantiene como mejor personal a la última posición no dominada que ha encontrado, la

cual es actualizada cada vez que la partícula encuentra una solución que la domina.

3.4 Coello y Lechuga (2002)

Coello y Lechuga [Coello02] presentan un algoritmo MOPSO en la cual se mantiene un repositorio que almacena todas las soluciones no dominadas entre si encontradas durante el proceso de búsqueda.

En este repositorio, el espacio objetivo explorado se representa por regiones llamadas hipercubos, donde cada hipercubo recibe una calificación igual al resultado de dividir 10 entre el número de soluciones situadas en él.

El repositorio es actualizado tras cada iteración, insertando en él todos los elementos no dominados de la población actual y eliminando los elementos dominados. Si este repositorio se llena, se eliminan las soluciones situadas en los hipercubos más poblados.

La selección de un mejor global para cada partícula se realiza seleccionando uno de los hipercubos mediante el método de la ruleta a través de sus valores de calificación y luego eligiendo al azar a cualquier elemento dentro del hipercubo seleccionado.

Cada partícula mantiene como mejor personal a la última posición no dominada en la que se ha encontrado, la cual es actualizada cada vez que la partícula encuentra una solución que la domina o con la cual resulta no comparable.

3.5 Fieldsend y Singh (2002)

Fieldsend y Singh [Fieldsend02] proponen un algoritmo MOPSO en el cual se utiliza una estructura de datos llamada *dominated tree* [Fielsend01], que mantiene un repositorio que almacena todas las soluciones no dominadas encontradas.

El mejor global para una partícula se elige del repositorio global haciendo uso del *dominated tree* [Fielsend01]. Además, cada partícula mantiene una lista con las soluciones no dominadas entre si que ha encontrado, eligiendo al azar a una de ellas para ser utilizada como mejor personal en su actualización.

En este enfoque también se introduce un parámetro llamado *turbulencia*, el cual básicamente actúa como operador de mutación sobre el nuevo valor de velocidad de cada partícula. Este parámetro es aplicado en la ecuación (5) de la siguiente manera:

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1} + t. \quad (11)$$

donde t es un valor aleatorio en el intervalo $[-t_{MAX}, t_{MAX}]$ y t_{MAX} es el máximo valor absoluto para el parámetro de turbulencia.

3.6 Mostaghim y Teich (2003)

Mostaghim y Teich [Mostaghim03] proponen un algoritmo MOPSO que hace uso de una función llamada *sigma* para dirigir las partículas hacia las soluciones no dominadas almacenadas en un repositorio global.

La función *sigma* asocia a cada partícula un vector σ, el cual define una línea gradiente a partir del punto solución de la partícula hacia el origen de coordenadas en el espacio objetivo [Mostaghim03]. Además, esta función tiene la característica de asignar el mismo vector σ a todas las partículas cuyos vectores solución se encuentren en la misma línea gradiente hacia el origen de coordenadas.

Cada partícula elige como mejor global al elemento del repositorio para el cual la distancia euclidiana entre sus vectores σ sea la menor.

Todas las partículas mantienen como mejor personal a la última posición no dominada encontrada, la cual es actualizada cada vez que la partícula encuentra una nueva solución que la domina.

4. Adaptación del PSO para el TSP

Para las implementaciones de este trabajo se utilizó la adaptación del PSO para el TSP propuesta por Secrest [Secrest01]. En esta adaptación el espacio de decisión se define como el conjunto de todas las permutaciones posibles de los vértices del grafo del problema. De esta forma, la posición de una partícula corresponde a una permutación y las partículas recorren el espacio de decisión pasando de una permutación a otra.

La adaptación propuesta por Secrest consiste en un algoritmo de construcción de soluciones, el cual conserva la idea del PSO de que la siguiente posición a visitar por una partícula se deduce a partir de su posición actual (X) y de las mejores posiciones, específicamente del mejor global (G) y del mejor

local de la partícula (L). En este algoritmo, los parámetros cognitivo (C_1) y social (C_2) se reemplazan por los siguientes factores de influencia: K_1, que indica la probabilidad de seleccionar información de la posición actual de la partícula, K_2, que indica la probabilidad de seleccionar información de la mejor posición en la vecindad de la partícula, y K_3, que indica la probabilidad de seleccionar información de la mejor posición del enjambre. La suma de estos tres factores debe ser igual a 1 (100%). El algoritmo de actualización de la posición de una partícula se ilustra en la figura 4.

Al observar el algoritmo de Secrest puede notarse que si un vértice *j* aparece luego de un vértice *i* en la mejor posición del enjambre y en la mejor posición de la vecindad, este vértice *j* tiene una probabilidad de $K_2 + K_3$ de aparecer luego del vértice *i* en la nueva posición para la partícula [Secrest01]. Esta característica es similar a la de los algoritmos de Colonia de Hormigas, en los cuales los caminos más utilizados tienen mayor probabilidad de ser empleados nuevamente.

Para poder aplicar esta adaptación en las implementaciones de los enfoques MOPSO se establecieron N vecindades para las N partículas del enjambre, en donde la vecindad de la partícula i estaba formada por las partículas i, $i - 1$ e $i + 1$. Nótese que la vecindad de la partícula 1 está formada por las partículas N, 1 y 2, y que la vecindad de la partícula N esta formada por las partículas N-1, N y 1. A cada vecindad se asoció un repositorio que almacena todas las soluciones no dominadas entre sí encontradas por las partículas de dicha vecindad. De esta manera, cada vez que se requiera un mejor local para la partícula *i* se selecciona una solución al azar del repositorio de su vecindad.

5 Resultados experimentales

Se realizaron implementaciones en C++ de los algoritmos de MOPSO propuestos por Moore y Chapman (mcMOPSO), Hu y Eberhart (heMOPSO), Coello y Lechuga (clMOPSO), Fieldsend y Singh (fsMOPSO), Mostaghim y Teich (mtMOPSO), y de los algoritmos de colonia de hormigas para problemas multiobjetivos (MOACO) MOACS [Barán03], M-MMAS [Pinto05] y PACO [Doerner02]. Todas las implementaciones fueron compiladas utilizando el compilador GCC v3.2.3 (MinGW) y corridas en una máquina con procesador AMD 3200+ que trabaja a una frecuencia de reloj de 2000MHz, 512 MB de memoria RAM y S. O. Windows XP.

```
pos_actual := Leer la posición actual de la partícula;
Leer parámetros K1, K2, K3, nro_vertices y las mejores posiciones G y L;
nueva_pos[1] := pos_actual[nro_vertices];  /* nueva posición */
FOR i := 2 TO nro_vertices DO
   r := rand(100);
   vert_elegido = FALSE;
   IF(r <= K3) THEN
      Elegir el vértice j que aparece luego del vértice indicado
      en nueva_pos[i-1] en el mejor global G;
      IF(vértice j no está en nueva_pos) THEN
         nueva_pos[i] := vértice j; vert_elegido = TRUE;
      END_IF
   END_IF
   IF(r <= (K2 + K3) OR vert_elegido = FALSE) THEN
      Elegir el vértice j que aparece luego del vértice indicado
      en nueva_pos[i-1] en la mejor posición local L;
      IF(vértice j no está en nueva_pos) THEN
         nueva_pos[i] := vértice j; vert_elegido = TRUE;
      END_IF
   END_IF
   IF(vert_elegido = FALSE) THEN
      nueva_pos[i] := ultimo vértice en pos_actual que no está en nueva_pos;
   END_IF
END_FOR
retornar(nueva_pos); /* retornar la nueva posición */
```

Figura 4. Algoritmo de construcción de soluciones de Secrest.

Luego de varias pruebas empíricas se encontró que los valores de 0, 10 y 90 resultaron los más apropiados para los parámetros K_1, K_2 y K_3 del algoritmo de construcción de soluciones de Secrest.

El número de partículas utilizadas en los algoritmos MOPSO fue de 50. Para el algoritmo heMOPSO se estableció el valor del parámetro *m* en 2 y el objetivo de selección se elegía de manera aleatoria en cada actualización de las partículas. En los algoritmos de colonia de hormigas se utilizaron 10 hormigas, con los parámetros α, β, ρ, τ_0 y q_0 puestos a 1, 2, 0.1, 1 y 0.5 respectivamente.

El problema resuelto por todos los algoritmos fue el KROAB100 de 100 ciudades, obtenido de la librería de problemas TSPLIB (http://www.iwr.uni-heidelberg.de/groups/comopt/software/ TSPLIB95).

Los resultados obtenidos en todas las corridas con todos los algoritmos fueron combinados para obtener una aproximación al frente Pareto real del problema KROAB100, denominada Y_{true}.

5.1 Métricas de desempeño

Para evaluar el desempeño de los algoritmos implementados fueron utilizadas las métricas M_1^*, M_2^* y M_3^* propuestas por de Zitzler et al. [Zitzler00] que miden la calidad, distribución y extensión del frente Pareto *Y'* generado por cada algoritmo respectivamente y la métrica de *Error* propuesta por Veldhuizen [Veldhiuzen99] que se refiere al porcentaje de soluciones generadas por cada algoritmo que no pertenecen a Y_{true}. A continuación se presentan las definiciones de cada métrica:

$$M_1^*(Y') = \frac{1}{|Y'|} \sum_{\forall p \in Y'} min\left(d(p,q) \; \forall q \in Y_{true}\right)$$

$$M_2^*(Y') = \frac{1}{|Y'|-1} \sum_{\forall p \in Y'} \left| \{ \; q \mid q \in Y' \wedge d(p,q) > \delta \; \} \right|$$

$$M_3^*(Y') = \sqrt{\sum_{i=1}^{M} max\left(|p_i - q_i|\right)} \quad \forall \, p, q \in Y'$$

$$Error(Y') = \frac{\sum_{i=1}^{|Y'|} e_i}{|Y'|}$$

donde $d(p, q)$ calcula la distancia euclidiana entre los puntos p y q, $|\cdot|$ representa la cardinalidad, M es la dimensión del espacio objetivo, e_i toma valor 1 si la solución p_i de *Y'* no se encuentra en Y_{true} y 0 en caso contrario, mientras que δ se estableció al 10% de la distancia entre los puntos extremos de Y_{true}.

5.2 Resultados de las corridas

Todas las implementaciones fueron corridas 10 veces durante 200 segundos y los resultados fueron combinados de manera a obtener el frente Pareto *Y'* generado por cada algoritmo.

La tabla 1 muestra el ranking de los algoritmos según su evaluación en cada una de las métricas. En la figura 5 puede observarse el frente Pareto generado por los algoritmos clMOPSO, mtMOPSO, MOACS y M-MMAS, obtenido al combinar los resultados de todas las corridas por cada algoritmo.

En general los algoritmos MOPSO obtuvieron mejores soluciones que los algoritmos MOACO hacia el centro del frente, pero en los extremos del frente solo los últimos obtuvieron resultados. Otra diferencia importante entre estos algoritmos es la densidad del frente hallado. En la totalidad de las pruebas los algoritmos MOPSO generaron 2 a 3 veces más soluciones que los algoritmos MOACO.

M_1^*		M_2^*		M_3^*		*Error*	
mtMOPSO	102,37	mtMOPSO	633,72	MOACS	535,51	mtMOPSO	21,07%
heMOPSO	644,41	mcMOPSO	289,13	M-MMAS	524,75	fsMOPSO	33,98%
clMOPSO	2.037,02	clMOPSO	230,89	fsMOPSO	442,49	heMOPSO	77,15%
fsMOPSO	2.044,95	fsMOPSO	230,11	mtMOPSO	417,19	M-MMAS	93,80%
mcMOPSO	2.797,82	heMOPSO	226,62	clMOPSO	399,24	MOACS	94,16%
PACO	4.810,19	MOACS	124,58	mcMOPSO	334,14	clMOPSO	100,00%
M-MMAS	6.643,43	M-MMAS	104,52	heMOPSO	281,57	mcMOPSO	100,00%
MOACS	6.700,11	PACO	24,48	PACO	212,31	PACO	100,00%

Tabla 1. Ranking de los algoritmos según su evaluación en cada una de las métricas.

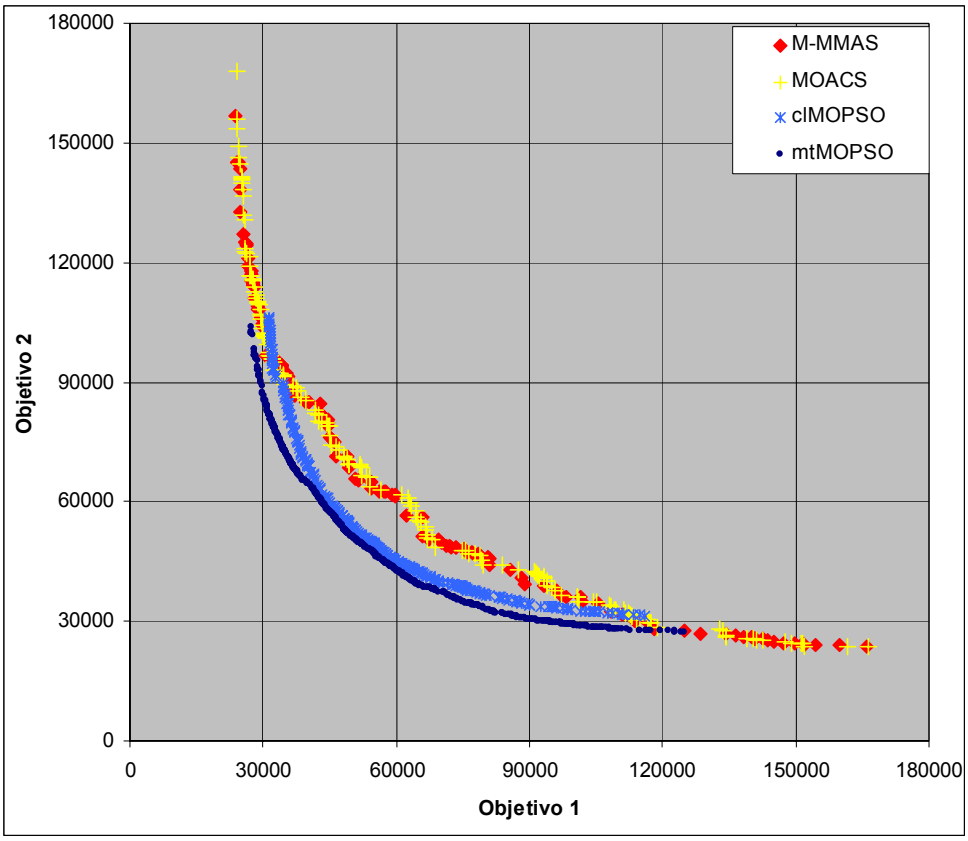

Figura 5. Frentes Pareto de los algoritmos clMOPSO, mtMOPSO, MOACS y M-MMAS.

Los algoritmos MOPSO que no aparecen en la figura 5 obtuvieron soluciones hacia el centro del frente que se encuentran por delante (dominan) y muy próximas a las soluciones obtenidas por los algoritmos MOACO.

6 Conclusiones y Trabajos Futuros

Las pruebas realizadas muestran que los algoritmos MOPSO obtienen mejores soluciones que los algoritmos MOACO considerando al menos 3 de las 4 métricas utilizadas como se muestra en la tabla 1, aunque también existen soluciones no encontradas por los primeros, lo cual puede observarse en la figura 5.

Otro hecho resaltante es que aunque se trate de un problema combinatorio, que en principio es un problema no apropiado para ser resuelto mediante la aplicación del PSO, este logró desenvolverse eficientemente con las adaptaciones aplicadas.

Durante las pruebas también se pudo observar que para una sola corrida, los algoritmos MOACS y M-MMAS obtenían un frente de gran amplitud, mientras que los algoritmos MOPSO encontraban soluciones correspondientes a una región limitada, pero con gran densidad y de mucho más calidad que las soluciones encontradas por los algoritmos MOACO.

Cabe hacer notar también, que los algoritmos MOPSO tienden a concentrase más en las soluciones que optimizan ambos objetivos simultáneamente, razón por la cual les fue difícil encontrar soluciones hacia los extremos del frente.

Como trabajo futuro se plantean aplicar la metaheurística del PSO a otros problemas combinatorios como el VRPTW, QAP y el enrutamiento en redes, como también intentar mejorar la extensión de los frentes hallados por los algoritmos MOPSO.

También se propone combinar los algoritmos MOPSO y MOACO en un algoritmo híbrido, utili-

zando un algoritmo MOACO para obtener una primera aproximación al frente Pareto del problema y luego aplicar un algoritmo MOPSO para refinar tal aproximación.

Otra propuesta consiste en utilizar los algoritmos MOPSO y MOACO en un Equipo de Algoritmos (*Team Algorithm*), pues como se ha visto estos métodos presentan resultados complementarios.

Referencias

[Barán03] B. Barán y M. Schaerer. "A Multiobjective Ant Colony System for Vehicle Routing Problems with Time Windows". Proc. Twenty first IASTED International Conference on Applied Informatics, pp. 97-102. Insbruck, Austria. 2003.

[Beielstein02] T. Beielstein, K. E. Parsopoulos, and M.N. Vrahatis. "Tuning PSO parameters through sensitivity analysis". Technical Report of the Collaborative Research Center, University of Dortmund, 2002.

[Coello02] C. A. Coello Coello and M. S. Lechuga. "MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization". In Congress on Evolutionary Computation CEC 2002, Vol. 2, pp. 1051–1056, Piscataway, New Jersey, May 2002.

[Doerner02] K. Doerner, W. Gutjahr, R. Hartl, C. Strauss and C. Stummer. "Pareto Ant Colony Optimization: A Metaheuristic Approach to Multiobjective Portfolio Selection". Proceedings of the 4th. Metaheuristics International Conference. Porto, pp. 243-248. 2002.

[Fieldsend02] J. E. Fieldsend and S. Singh. "A Multi-Objective Algorithm based upon Particle Swarm Optimization, an Efficient Data Structure and Turbulence". In Proceedings of the 2002 U.K. Workshop on Computational Intelligence, pp. 37–44, Birmingham, UK, September 2002.

[Fieldsend01] J. E. Fieldsend and S. Singh. "Using Unconstrained Elite Archives for Multi Objective Optimization". IEEE Transactions on Evolutionary Computation (Submitted), 2001.

[Hu02] X. Hu and R. Eberhart. "Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization". Computational Intelligence, Hawaii, May 12-17, 2002. IEEE Press.

[Kennedy95] J. Kennedy and R. C. Eberhart. "Particle Swarm Optimization". In Proceedings of the 1995 IEEE International Conference on Neural Networks, pp. 1942–1948, Piscataway, New Jersey, 1995.

[Kennedy01] J. Kennedy and R. Eberhart. "Swarm Intelligence". Morgan Kaufmann Publishers, 2001.

[Moore99] J. Moore and R. Chapman. "Application of Particle Swarm to Multiobjective Optimization". Department of Computer Science and Software Engineering, Auburn University, 1999.

[Mostaghim03] S. Mostaghim and J. Teich. "Strategies for Finding Good Local Guides in Multiobjective Particle Swarm Optimization (MOPSO)". In 2003 IEEE Swarm Intelligence Symposium Proceedings, pp. 26–33, Indianapolis, Indiana, USA, April 2003.

[Pinto05] D. Pinto and B. Barán. "Solving Multiobjective Multicast Routing Problem with a new Ant Colony Optimization approach". LANC'05, Cali, Colombia. 2005.

[Ray02] T. Ray and K. M. Liew. "A Swarm Metaphor for Multiobjective Design Optimization". Engineering Optimization, Vol. 34, No. 2, pp. 141-153, 2002.

[Secrest01] B. Secrest. "Traveling Salesman Problem for Surveillance Mission Using Particle Swarm Optimization". MS Thesis, AFIT/GCE/ENG/01M-03, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 2001.

[Shi98] Y. Shi and R. Eberhart. "Parameter Selection in Particle Swarm Optimization". In Proceedings of the Seventh Annual Conference on Evolutionary Programming, pp. 591-601, 1998.

[Veldhuizen99] D. A. van Veldhuizen. "Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations". PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

[Zitzler00] E. Zitzler, K. Deb and L. Thiele. "Comparison of multiobjective evolutionary algorithms. Empirical result", Evolutionary Computation, pp. 173-195, 2000.