Optimización de Algoritmos PSO Mediante Regresión Exponencial

Miguel Augusto Azar, Fabiola Patricia Paz, Analía Herrera Cognetta

Facultad de Ingeniería - Universidad Nacional de Jujuy
auazar@live.com, lic.faby@gmail.com, anihco@yahoo.com

Resumen. El número de iteraciones ideal que los algoritmos basados en enjambres de partículas requieren para la obtención del óptimo de una función fitness es un factor indeterminado. En algunos casos, ya sea por limitaciones de procesamiento o de capacidad de memoria, es necesario reducir el costo computacional que implica un número excesivo de iteraciones necesarias para obtener dicho valor óptimo. El presente trabajo propone un algoritmo PSO modificado que mediante la regresión exponencial de los puntos de velocidad de las partículas permite obtener una rápida convergencia al óptimo para funciones lineales.

Palabras Clave. Optimización, metaheurísticas, inteligencia de enjambre, particle swarm optimization, convergencia rápida, PSO.

1 Introducción

La técnica de optimización por enjambre de partículas PSO (Particle Swarm Optimization) es una de las metaheurísticas en el campo de la inteligencia artificial que permite encontrar el valor óptimo de una función objetivo determinada y constituye una alternativa excelente para la resolución de funciones multidimensionales en forma aproximada. Esta metaheurística utiliza un conjunto de partículas que se desplazan y colaboran entre sí en un espacio n-dimensional. Este proceso demanda un costo computacional que depende de la función analizada y de la estrategia empleada. La reducción de dicho costo significa siempre una mayor eficiencia de los recursos y por ende un mejor aprovechamiento del hardware, esto es, el procesador y la memoria.

Los sistemas inteligentes basados en metaheurísticas se caracterizan por la cualidad de obtener resultados óptimos aproximados con un error que está en función del número de iteraciones. Cuando la función objetivo es más compleja es necesario incrementar en número de iteraciones para obtener un valor óptimo aceptable con el consecuente aumento del costo computacional. Es por ello que, en estos casos, es necesario plantear una alternativa al algoritmo PSO convencional que permita un mejor aprovechamiento de los recursos de hardware.

2 Inteligencia de enjambres y el coeficiente de inercia en PSO

2.1 Inteligencia de enjambres

La inteligencia de enjambre (Swarm Intelligence) es un campo de la informática que se dedica a los diseños y estudios de métodos computacionales eficientes para resolver problemas complejos, inspirados por el comportamiento de los enjambres reales o colonias de insectos [1] [2].

La inteligencia de enjambres (SI) está basada en poblaciones de individuos (soluciones potenciales candidatas) que cooperan entre sí y estadísticamente son cada vez mejores a través de las iteraciones con las que, finalmente, encuentran una o varias soluciones [3]. El empleo de algoritmos PSO flexibiliza el amplio conjunto de problemas de optimización, sin la necesidad de tener un conocimiento profundo y detallado de las características implícitas de la función que se desea optimizar [4]. El comportamiento real y complejo de las colonias de insectos, que surgen de las acciones e interacciones de los individuos mejores adaptados, muestran las habilidades para dar solución a problemas complejos [5].

La técnica PSO imita el comportamiento social y comunicacional que se establece en grupos de individuos, tales como aves y peces, y utilizan la inteligencia colectiva como recurso para hallar alimento o refugio; esto significa que ajustan sus movimientos para evitar depredadores o buscar comida [6].

En estos algoritmos cada ave se representa mediante una partícula que está siempre en continuo movimiento dentro del espacio de búsqueda; almacenando, y en algunos casos, comunicando a todo el enjambre la mejor solución que han encontrado hasta el momento [7]. Es por ello que la inteligencia de enjambres es una de las técnicas de la inteligencia artificial más utilizada en optimización.

Los algoritmos generados mediante la SI tienen una gran ventaja con respecto a otros métodos de optimización matemáticos. No necesitan conocer todas las características de la función a optimizar, de allí que los algoritmos son más flexibles por lo que se puede aplicar a un gran número de problemas de optimización [8].

2.2 El coeficiente de inercia en PSO

El algoritmo PSO puede ser puramente cognitivo si la tendencia de la partícula depende de las mejores posiciones encontradas en su pasado personal; o bien, puede ser puramente social si tal tendencia es proporcional al pasado del cúmulo. Si el modelo tiene ambas componentes (tanto cognitiva como social) la velocidad de cada partícula puede determinar la forma en que estas convergen al valor óptimo. Shi y Eberhart [9] propusieron una mejora del algoritmo básico, modificando la fórmula de actualización de la velocidad mediante la introducción de una nueva variable denominada factor de inercia *w*.

Tanto el proceso de exploración y explotación del espacio de búsqueda, como así también, el control que permite que la velocidad no exceda sus límites establecidos, son equilibrados mediante el coeficiente de inercia. Dicho coeficiente regula la velocidad, multiplicando su peso por la velocidad previa.

Al incorporar el factor de inercia *w* la actualización de la velocidad queda determinada por:

$$v_{id} = w * v_{id} + r_1 * p_1 * (pb_{id} - pos_{id}) + r_2 * p_2 * (gb_d - pos_{id}) \quad (1)$$

En donde la v_{id} es la velocidad de la iteración anterior de la partícula *i* en la dimensión *d* (número de variables), r_1 y r_2 son valores aleatorios en el rango [0,1], p_1 y p_2 son los coeficientes de aprendizaje personal y social, pb_{id} (personal best) es la mejor posición personal encontrada por la partícula *i* en la dimensión *d*, pos_{id} es la posición actual de la partícula *i* en la dimensión *d* y gb_d (global best) es el vector de posición global, mejor encontrado por todas las partículas, obtenido en la función objetivo y se actualiza después de cada iteración del algoritmo.

El valor del factor de inercia *w* puede permanecer fijo o bien puede ser linealmente decrementado en cada ciclo de vuelo.

Si este factor tiene características dinámicas, *w* es reducido linealmente utilizando dos valores límites: $w_{inicial}$ y w_{final} que generalmente toman los valores: 0.9 y 0.4, respectivamente.

$$w = \frac{(ciclo_{max} - ciclo) * (w_{inicial} - w_{final})}{ciclo_{max}} + w_{final} \quad (2)$$

La ecuación (2) muestra la dinámica del factor de inercia *w* el cual es actualizado en cada ciclo del proceso, hasta el final de las iteraciones ($ciclo_{max}$).

3 Aplicación de la regresión exponencial

El factor de inercia *w*, como se indicó anteriormente, posee características lineales por lo que el movimiento de cada partícula debiera acompañar tal cualidad. Sin embargo, en general las partículas en PSO se desplazan con movimiento exponencial en función de los ciclos; esto es, durante las primeras iteraciones de exploración la velocidad de cada partícula toma valores extremos mientras que en los últimos ciclos se reduce hasta alcanzar una convergencia aproximada. En [10], basado en este concepto, se propone el cálculo de un peso de inercia adaptativa exponencial (Exponential Particle Swarm Optimization) que mejora el rendimiento de la búsqueda de las funciones de referencia de manera significativa. Por otro lado, mediante esta cualidad, en [11] se exponen diferentes estrategias novedosas para la dinámica del peso de inercia exponenciales. En [12] y [13] se plantea el uso del coeficiente de inercia *w* como alternativa a la clásica fórmula (2) de cálculo de dicho factor de inercia en forma lineal.

Ahora bien, dependiendo de las características del problema a optimizar, el número de ciclos necesarios para una convergencia segura es siempre indeterminado. Por lo general se adopta un número fijo de iteraciones de manera que el algoritmo obtenga una convergencia aceptable en cuanto a su valor de aproximación. Dicha indeterminación obliga en muchos casos a adoptar un número de iteraciones convenientemente alto con el consecuente incremento del costo computacional, dado que no es posible conocer con antelación, a partir de que iteración la convergencia es confiable.

Frente a esta problemática, se propone la regresión exponencial de las posiciones de las partículas para obtener una curva que permita inferir el valor óptimo de la función fitness en forma anticipada y así lograr una rápida convergencia.

La regresión exponencial básica consiste en encontrar aquellos coeficientes pertenecientes a una función exponencial de manera que satisfagan, con la máxima aproximación posible, a los puntos de interés.

Una función exponencial simple de la forma:

$$a * exp^{b*x} \tag{3}$$

Requiere para su resolución la determinación de los coeficientes a y b tal que los puntos conocidos y definidos por x sean representados en forma aproximada por la función.

Una función simple de estas características constituye la base que posibilita una regresión con un error que depende del nivel de dispersión de los puntos de interés. Es por ello que resulta conveniente ampliar los términos de esta expresión a una función de la forma:

$$a * exp^{b*x} + c * exp^{d*x} + e * exp^{f*x} \tag{4}$$

Precisamente, a mayor cantidad de términos exponenciales representativos, mejor será el ajuste entre la curva exponencial hallada y el conjunto de puntos de interés. En el presente estudio, se empleó una función como la presentada en (4) para representar las posiciones de las partículas del algoritmo PSO en el espacio de búsqueda.

En la Figura 1 se observa la función exponencial implementada en MATLAB® y aplicada a las posiciones de las partículas. El empleo de la función lsqnonlin permite encontrar los valores de los coeficientes que representan la función exponencial. En tanto que en la Figura 2, se presenta la función *regresión_exp* que obtiene las diferencias entre los puntos de posición del algoritmo PSO y la función exponencial presentada en (4).

```
x1=lsqnonlin(regresion_exp,X0,[],[],[],X,Y1);
Y1_regresion = x1(1).*exp(x1(2).*X) + x1(3).*exp(x1(4).*X) + x1(5).*exp(x1(6).*X);
```

Fig. 1. Código en MATLAB® de la aplicación de la regresión exponencial.

```
function resta = regresion_exp(vector_coef,vx,vy)
    a = vector_coef(1);
    b = vector_coef(2);
    c = vector_coef(3);
    d = vector_coef(4);
    e = vector_coef(5);
    f = vector_coef(6);
    resta = a.*exp(b.*X)+c*exp(d.*X)+e.*exp(f.*X) - Y;
```

Fig. 2. Diferencias de las posiciones del algoritmo PSO y la función exponencial.

Los parámetros utilizados para la definición del algoritmo PSO son los siguientes: Cantidad de partículas 20, Dimensión (cantidad de variables que utiliza la función objetivo) 2, Factor de inercia inicial $w_{inicial}$ 0.9, Factor de inercia final w_{final} 0.4, Constante de aprendizaje cognitivo y social 1.49445.

Para la implementación inicial y posterior obtención de una convergencia rápida se utilizó una función objetivo simple lineal con dos variables y sujeta a tres restricciones. La adquisición de los puntos de interés a los que se aplica la regresión exponencial son los valores que toma cada partícula en cada iteración. En este caso, se separan todos aquellos puntos que forman la curva cóncava hacia abajo antes de aplicar la regresión.

En las Figuras 3 y 4 se observan los valores que toman las variables x1 y x2 en función del número de ciclos del algoritmo PSO para diferentes ejecuciones. Los puntos de interés son representados a través de la curva de línea continua en donde aproximadamente en el ciclo 80 se intersectan.

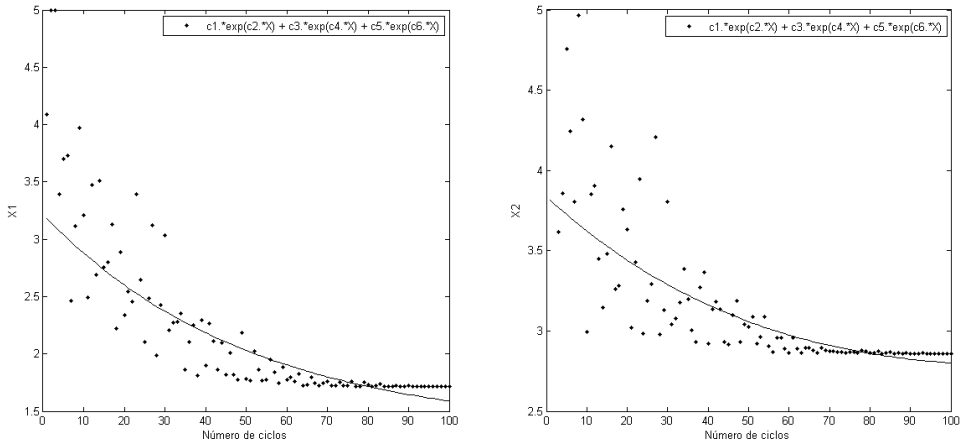

Fig. 3. Regresión exponencial de las velocidades de las partículas correspondientes a las variables x1 y x2 de la función a optimizar.

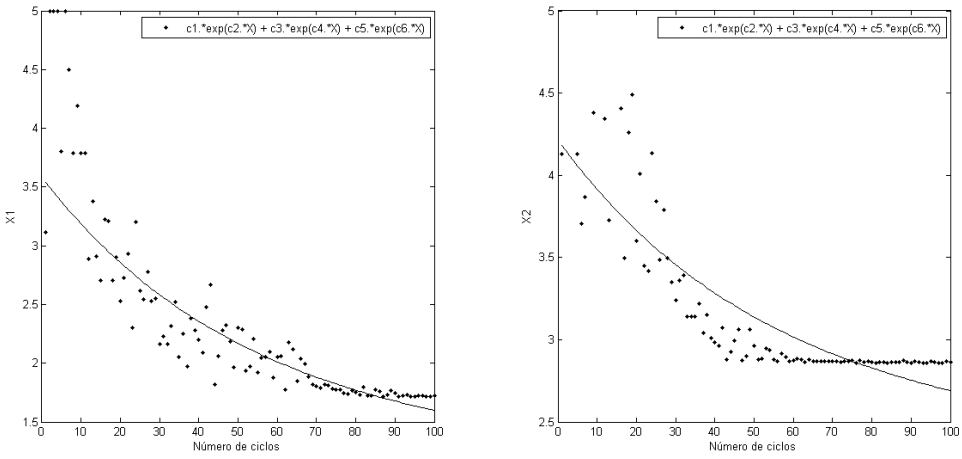

Fig. 4. Regresión exponencial de las velocidades de las partículas correspondientes a las variables x1 y x2 de la función a optimizar (ejecución 2).

El análisis realizado en diferentes pruebas da cuenta de que es posible tomar dicho punto de intersección como condición de parada del algoritmo PSO obteniendo una rápida convergencia con un error promedio inferior al 0.3% evitando llegar al ciclo 200 que se sugiere habitualmente para asegurar una mejor aproximación.

Cabe aclarar que el error promedio del 0.3% no corresponde a un valor de ajuste sino a la diferencia entre el valor óptimo buscado y el valor de la variable x1 (o x2 según corresponda) en el ciclo en donde se produce la intersección entre la curva de puntos y la curva de regresión.

4 Conclusiones

El empleo de un método de convergencia rápida como el presentado se encuentra en una etapa de análisis por lo que aún no es posible obtener conclusiones taxativas. Sin embargo, en esta primera fase se obtuvieron resultados satisfactorios que podrían ser extrapolados a funciones no lineales con un mayor número de variables.

La utilización de una función objetivo lineal permitió un acercamiento básico, a la reducción del costo computacional que implica el uso de algoritmos PSO, ante la incertidumbre que supone la condición de parada que, para asegurar una convergencia aproximada, debe estar en el orden de los 200 ciclos o más. Conseguir una convergencia aproximada en el ciclo 80 frente al clásico ciclo 200 mejora la respuesta

en cuanto a velocidad pero es necesario continuar desarrollando nuevas pruebas con funciones de referencia complejas no lineales (benchmark test), disponibles para evaluar este tipo de algoritmos, y así obtener conclusiones más significativas en cuanto a su aplicación.

Referencias

1. Bonabeau E., Dorigo M., Theraulaz G.: Swarm Intelligence: From Natural to Artificial System. Oxford University Press, New York (1999)
2. Kennedy J., Eberhart R. C., Shi Y.: Swarm Intelligence. Morgan Kaufmann, San Francisco, CA (2001)
3. P Panigrahi B.K., Shi Y., and Lim M., Handbook of Swarm Intelligence: Concepts Principles and Applications, Springer (2011)
4. Eiben A.E., Smith J.E., Introduction to Evolutionary Computing. Springer Natural Computing Series, 1st edition, ISBN: 3-540-40184-9 (2003)
5. Merkle D., Middendorf M., Swarm Intelligence, Introductory Tutorials in Optimization and Decision Support Techniques. Department of Computer Science. Chapter 14 University of Leipzig pp. 401-435, ISBN 978-0-387-23460-1 (2005)
6. Kennedy J., Eberhart R.C., Particle Swarm Optimization, in Proceedings IEEE International Conference of Neural Network, IEEE Service Center, Piscataway, pp. 1942-1948,NJ (1995)
7. Paz F.P., Azar M.A., Herrera Cognetta A., Pérez Otero N.M., Una alternativa para el mecanismo de manejo de restricciones en algoritmos PSO, in Proceedings Second Argentine Conference on Human-Computer Interaction, Telecomunications, Informatics and Scientific Information HCITISI, Córdoba, Argentina, ISBN 978.88.96.471.25.8, Noviembre 21-22 (2013)
8. Eiben A.E., Smith J.E.: Introduction to Evolutionary Computing. Springer Natural Computing Series, 1st edition, ISBN: 3-540-40184-9 (2003)
9. Shi Y., Eberhart R.C., Parameter Selection in Particle Swarm Optimization, in Proceeding of the Seventh Annual Conference on Evolutionary Programming, San Diego, California, pp. 591–600, USA, March 25–27 (1998)
10. Wu J., Liu W., Zhao W., Li Q., Exponential Type Adaptive Inertia Weighted Particle Swarm Optimization Algorithm. Second International Conference on Genetic and Evolutionary Computing, pp. 79-82. IEEE Press, New York (2008)
11. Chauhan P., Deep K., Pant M., Novel inertia weight strategies for particles warm optimization. Memetic computing, Vol. 5, pp. 229-25, ISSN: 1865-9284 (2013)
12. Ghali N.I., El-Desouki N., Mervat A.N., Bakrawi L. Exponential Particle Swarm Optimization Approach or Improving Data Clustering. World Academy of Science, Engineering and Technology 42, pp. 50-54. (2008)
13. Chen G., Huang X., Jia J., Ming Z. Natural Exponential Inertia Weight Strategy in Particle Swarm Optimization. Proceedings of the 6th Congress on Control and Automation, June 21-23, pp. 3672, Dalian, China. (2006)