UNIVERSIDAD DE GRANADA

Metaheurísticas: Práctica Alternativa - Whale Optimization Algorithm

---

Iván Rodríguez Chacón
77393480K
Grupo A2 curso 2024-2025

Índice

Introducción

Vamos a hablar sobre la metaheurística Whale Optimization Algorithm (WOA), la cual podemos enmarcar en el contexto de algoritmos que pretenden encontrar buenas soluciones en espacios de más de una dimensión. Dicha metaheurística está bioinspirada en la estrategia de caza de las ballenas jorobadas, se trata de una técnica única de esta especie, lo que la hace bastante curiosa. Dicha estrategia de caza consiste en rodear a las presas y descender varios metros, una vez han descendido liberan burbujas por sus espiráculos y comienzan a subir en círculos sobre este banco creando una “red de burbujas” alrededor de las presas que les impide escapar, se puede ver mejor en esta imagen:

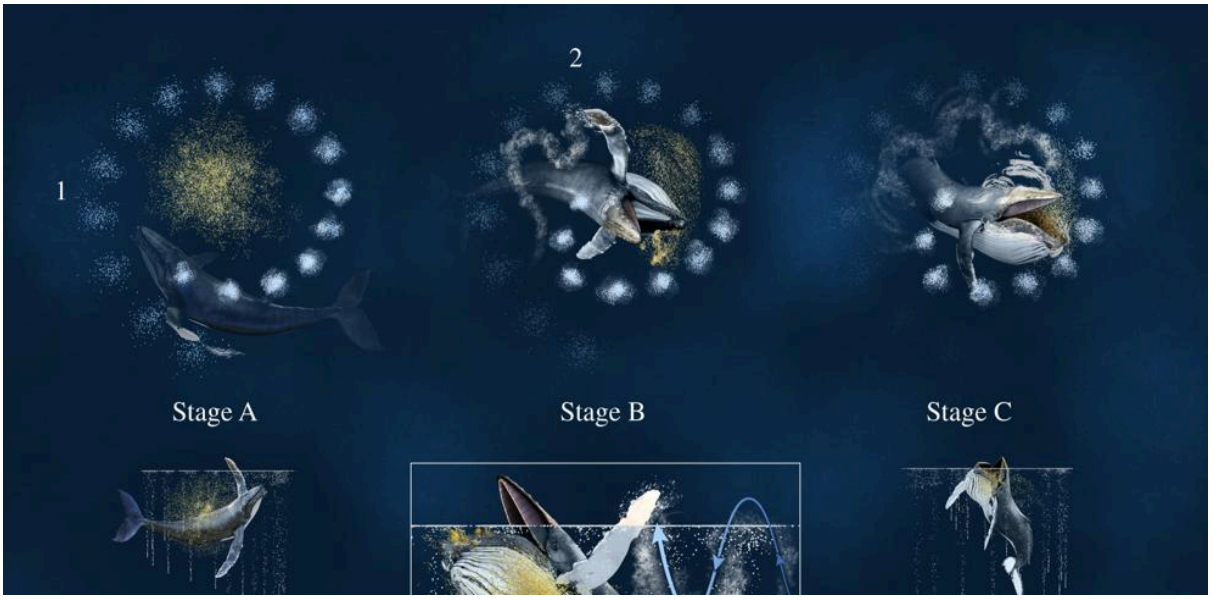

Imagen extraída de “La Vanguardia”

La filosofía principal que sigue esta metaheurística consiste en mantener un equilibrio adecuado entre exploración (generación de soluciones en regiones distintas del espacio) y explotación (mejora de soluciones en una zona concreta). En términos biológicos, primero se explora en busca de un buen banco de peces, una vez encontremos una buena presa comenzamos a explotar dicha zona realizando la técnica de la caza en espiral con burbujas.

Explicación del algoritmo

El algoritmo de optimización de ballenas funciona de la siguiente manera:

1. El algoritmo comienza con una población aleatoria de "ballenas" (agentes de búsqueda), donde cada ballena representa una solución candidata.
2. La mejor solución candidata (la mejor ballena) pasa a ser el objetivo.
3. Para simular el ataque de red de burbujas, el WOA considera dos mecanismos que ocurren simultáneamente en las ballenas jorobadas:
    a. Encogimiento del cerco: Cuando el valor absoluto de A es menor que 1 las ballenas se acercan a la mejor solución simulando el encogimiento del cerco aplicando las siguientes fórmulas:

    $$D = |C \cdot X'(t) - X(t)|$$

    $$X(t+1) = X'(t) - A \cdot D$$

    Dónde A y C son vectores de coeficientes, X'(t) es la mejor solución y X(t) la posición de la solución actual. El valor del vector A (Calculado como $2a \cdot r - a$) disminuye linealmente desde 2 a 0 a lo largo de las iteraciones. Esto permite la correcta transición entre exploración y explotación.

    b. Forma en Espiral: Se calcula la distancia entre la ballena actual y la mejor ballena, después se crea un movimiento en espiral que imita la trayectoria de las ballenas jorobadas al soplar burbujas alrededor de la presa. La ecuación para este movimiento es $X(t+1) = D' \cdot e^{bl} \cdot cos(2\pi l) \cdot + X'(t)$. Donde D' es la distancia a la mejor ballena, b es una constante para la forma de la espiral, y l es un número aleatorio.

    Para combinar estos dos comportamientos, el algoritmo utiliza una probabilidad del 50% para decidir si una ballena actualiza su posición usando el mecanismo de encogimiento del cerco o el modelo en espiral.

4. También se produce exploración cuando para una ballena el valor absoluto de A es mayor o igual a 1, en este caso la ballena se

acercará con la misma fórmula que utilizaría para acercarse al objetivo pero sustituyendo X’(t) por una ballena aleatoria.

5. Por último tras cada iteración se produce una evaluación para cada ballena actualizando, si procede, el valor X’(t), es decir, la mejor ballena.

El éxito de este algoritmo es propiciado por unos algoritmos de exploración y explotación muy potentes pero simples.

Implementación

La implementación realizada para ejecutar con el benchmark es la siguiente:

```
Entrada:
- func_id: identificador de la función CEC
- dim: dimensión del problema
- lim_inferior, lim_superior: límites inferiores y superiores de cada dimensión
- maxevals: número máximo de evaluaciones
- n_ballenas: número de ballenas (tamaño de la población)

Inicializar n_evals ← 0
Inicializar población de ballenas aleatoriamente dentro de los límites

Para cada ballena i en la población:
   Evaluar su aptitud
   n_evals ← n_evals + 1
   Si es la mejor hasta ahora:
      Guardar como mejor aptitud y mejor posición global

current_iteration ← 0

Mientras n_evals < maxevals:
   Calcular a ← 2 - (2 * current_iteration / (maxevals / n_ballenas))
   Limitar a entre 0 y 2

   Para cada ballena i:
      Si n_evals ≥ maxevals, salir del bucle

      Generar r1, r2, p, l aleatorios en [0,1] y [-1,1]
      Calcular A ← 2*a*r1 - a
      Calcular C ← 2*r2
      Inicializar nueva_posición

      Para cada dimensión j:
         Si p < 0.5:
```

```
            Si |A| ≥ 1:
               Escoger una ballena aleatoria X_rand
               D ← |C*X_rand - X_i|
               NuevaPos[j] ← X_rand[j] - A * D
            Sino:
               D ← |C*Mejor - X_i|
               NuevaPos[j] ← Mejor[j] - A * D
         Sino:
            D ← |Mejor[j] - X_i|
            NuevaPos[j] ← D * e^(b*l) * cos(2πl) + Mejor[j]

      Aplicar límites a nueva_posición

      Evaluar aptitud de nueva_posición
      n_evals ← n_evals + 1
      Actualizar ballena i con nueva_posición

      Si la nueva aptitud es mejor que la global:
         Actualizar mejor_aptitud_global y mejor_posición_global

   current_iteration ← current_iteration + 1

Devolver mejor_aptitud_global
```

En esta implementación podemos ver cómo se aplica todo lo comentado anteriormente.

Modificación

Este algoritmo es bastante malo ya que converge muy rápido y suele caer en óptimos locales con facilidad si nos regimos estrictamente por el paper, no obstante dicho algoritmo tiene funciones de exploración y explotación muy potentes por lo que pienso que puede mejorar si definimos mejor el valor “a” que es el que controla el cambio entre exploración y explotación. En mi caso para solucionar este problema de convergencia pensé en aplicar lo aprendido con el algoritmo de enfriamiento simulado. La forma en la que pensé para realizar esta modificación consiste es modificar el valor “a” por la temperatura normalizada entre 0 y 2, de esta forma el sistema seguiría funcionando de la misma forma, con valores de “a” entre 2 y 0, pero este descenso se hará mediante la temperatura. Aquí podemos ver el pseudocódigo de dicho algoritmo:

Entrada:

- func_id: identificador de la función CEC
- dim: dimensión del problema
- lim_inferior, lim_superior: límites inferiores y superiores de cada dimensión
- maxevals: número máximo de evaluaciones
- n_ballenas: número de ballenas (tamaño de la población)

Inicializar n_evals ← 0
Inicializar población de ballenas aleatoriamente dentro de los límites

mejor_aptitud_global ← ∞
mejor_posicion_global ← vector vacío

Para cada ballena i:
    Evaluar aptitud ← cec17_fitness(posición i)
    n_evals ← n_evals + 1
    Si aptitud < mejor_aptitud_global:
        mejor_aptitud_global ← aptitud
        mejor_posicion_global ← posición i

// Inicializar parámetros de enfriamiento simulado
Si mejor_aptitud_global <= 0:
    T0 ← 1.0
Sino:
    T0 ← (mu * mejor_aptitud_global) / -log(phi)
Tf ← 1e-6
current_T ← T0
beta ← (T0 - Tf) / (M * T0 * Tf)

current_iteration ← 0

Mientras n_evals < maxevals:
    Si current_iteration > 0:
        current_T ← current_T / (1 + beta * current_T)
    Si current_T < Tf:
        current_T ← Tf

    a ← 2 * (current_T / T0)
    Limitar a entre 0 y 2

    Para cada ballena i:
        Si n_evals ≥ maxevals:
            Salir del bucle

        r1, r2, p ← aleatorios en [0,1]
        l ← aleatorio en [-1,1]
        A ← 2 * a * r1 - a
        C ← 2 * r2

```
        Para cada dimensión j:
            Si p < 0.5:
                Si |A| ≥ 1:
                    Escoger ballena aleatoria X_rand
                    D ← |C * X_rand[j] - X_i[j]|
                    nueva_pos[j] ← X_rand[j] - A * D
                Sino:
                    D ← |C * mejor[j] - X_i[j]|
                    nueva_pos[j] ← mejor[j] - A * D
            Sino:
                D ← |mejor[j] - X_i[j]|
                nueva_pos[j] ← D * e^(b * l) * cos(2π * l) + mejor[j]

        Aplicar límites a nueva_pos

        Evaluar aptitud ← cec17_fitness(nueva_pos)
        n_evals ← n_evals + 1
        Reemplazar ballena i por nueva_pos

        Si aptitud < mejor_aptitud_global:
            mejor_aptitud_global ← aptitud
            mejor_posicion_global ← nueva_pos

    current_iteration ← current_iteration + 1

Devolver mejor_aptitud_global
```

Manual de usuario

Para utilizar ambos programas, tanto la versión puramente basada en el paper como la modificada por mi se debe de pasar un solo parámetro, que es el nombre de la carpeta donde almacenar los ficheros de resultados, dichas carpetas tienen que empezar por “results_” y después de dicho prefijo debe ir lo que se le pase como parámetro a los programas, es decir, en el caso de que se quiera guardar en “results_whale” debemos ejecutar el programa de la siguiente manera: “./woa whale”. Para compilar ambos programas se puede hacer mediante el fichero CMake, una vez compilados aparecerán dos ficheros: woa y woa-hybrid, el primero es el que respeta el paper y el segundo el modificado por mí.

Análisis de resultados y Benchmark

Resultados para dimensión 10:

Mean Ranking by algorithm

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	1.967	1.733	1.733	1.767	2.067	2.267	2.500	2.667	2.733	2.733	2.767	2.833	2.767	2.667
PSO	3.900	3.833	3.800	3.700	3.400	3.200	2.867	2.700	2.533	2.467	2.333	2.300	2.267	2.167
SSA	2.167	2.300	2.300	2.433	2.433	2.400	2.433	2.433	2.467	2.433	2.533	2.467	2.467	2.500
WOA	1.967	2.133	2.167	2.100	2.100	2.133	2.200	2.200	2.267	2.367	2.367	2.400	2.500	2.667

Mean Ranking by algorithm

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	1.900	1.633	1.633	1.733	2.100	2.300	2.567	2.700	2.767	2.767	2.800	2.900	2.867	2.833
PSO	3.933	3.867	3.833	3.700	3.433	3.167	2.967	2.633	2.433	2.467	2.400	2.300	2.233	2.267
SSA	2.100	2.300	2.267	2.367	2.333	2.433	2.400	2.400	2.467	2.467	2.533	2.533	2.600	2.567
WoA-HYBRID	2.067	2.200	2.267	2.200	2.133	2.100	2.067	2.267	2.333	2.300	2.267	2.267	2.300	2.333

Resultados para dimensión 30:

Mean Ranking by algorithm

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	2.067	2.367	2.500	2.633	2.800	2.933	2.967	3.000	2.967	2.900	2.900	2.800	2.733	2.733
PSO	3.833	3.567	3.500	3.433	3.233	3.067	2.900	2.867	2.600	2.533	2.533	2.433	2.367	2.267
SSA	1.900	1.833	1.800	1.800	2.000	1.967	1.967	1.967	2.067	2.067	2.067	2.167	2.200	2.233
WOA	2.200	2.233	2.200	2.133	1.967	2.033	2.167	2.167	2.367	2.500	2.500	2.600	2.700	2.767

Mean Ranking by algorithm

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	2.067	2.333	2.500	2.700	2.833	2.967	3.067	3.133	3.133	3.133	3.000	2.933	2.900	2.867
PSO	3.867	3.567	3.467	3.433	3.233	3.000	2.800	2.833	2.633	2.600	2.600	2.567	2.533	2.433
SSA	1.933	1.900	1.800	1.767	1.933	1.967	1.967	1.967	2.067	2.067	2.067	2.167	2.200	2.233
WoA-HYBRID	2.133	2.200	2.233	2.100	2.000	2.067	2.167	2.067	2.167	2.200	2.333	2.333	2.367	2.467

Podemos ver como los resultados son peores con una dimensión superior para ambos algoritmos y también podemos ver cómo el algoritmo modificado funciona mejor que del que partimos, por tanto mi hipótesis de que el algoritmo requería una alteración en el parámetro “a” serían ciertas, aunque viendo los resultados diría que seguramente habrá una forma mejor de hacer esto, no obstante el algoritmo obtiene resultados bastante buenos, pero en mi opinión se podría potenciar mucho más el algoritmo de optimización de ballenas a partir de modificaciones quizás en el umbral de “p” que actualmente es 0.5, o ajustando los hiperparámetros. Tras estos resultados pude investigar y efectivamente hay versiones de este algoritmo que dan resultados muchísimo mejores, como en este artículo donde además comparan su algoritmo modificado en el benchmark cec2019 donde obtiene resultados muy buenos. Por tanto el algoritmo base no es bueno, pero es una buena base para comenzar a desarrollar uno muy potente, tal y como se describe en dicho artículo.