UNIVERSIDAD
DE GRANADA

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial

Memoria de Prácticas Final de Metaheurísticas

Daniel Chico
DNI: 26508525J
Correo: dachival@correo.ugr.es

Práctica optativa: Implementación del GWO y comparativas aplicando el problema de la competición CEC2017

Subgrupo: 3

Horario: Miercoles (17.30/19.30)

Tutor: Daniel Molina

15 de junio de 2023

Algoritmos Implementados:

- Grey wolf Optimization
- Grey Wolf Optimization - Búsqueda local

Índice general

0.1. Resumen

0.1.1. Grey Wolf Optimization

Para el desarrollo de la práctica alternativa decidí implementar una metaheurística basada en el lobo gris, a partir de ahora será GWO de sus siglas en inglés. Esta metaheurísticas se basa en el comportamiento de las manadas de lobos. Se basa en dos premisas, la gerarquía social dentro de la manada y las técnicas de caza.

De observar la gerarquía de los lobos se pueden distinguir 4 roles entre los que se dividen los individuos de una manada:

- α: Los jefes de la manada, no son los mas fuertes sino los que tienen una mayor capacidad organizativa
- β: Segundos en el escalafón, realizan tareas organizativas también y apoyan al α
- δ: Tercer escalafón
- ω: últimos en la escala social de la manada, son unos mandados a efectos prácticos, también son los lobos más débiles de la manada.

A la hora de cazar los lobos persiguen a su presa hasta que consiguen rodearla y que esta se pare, a partir de ese momento empiezan a atacarlo poco a poco hasta que consigue su objetivo.

Formalización matemática

Por lo comentado anteriormente esta metaheurística se basa en modelos poblacionales en el que las peores soluciones de la población son consideradas ω_s y se modifican en función de un parametro aleatório y una "suma" de las distancias a las mejores soluciones. En este caso corresponderían a los lobos α,β y δ. Este proceso difiere de como cazan los lobos ya que, en problemas de este tipo, no se puede "divisar" a la presa (solución óptima) para perseguirla asi que se hace la asumpción de que las mejores soluciónes "saben" algo sobre la solución óptima e influyen en el comportamiento del resto de la población.

Durante el proceso de caza, los lobos tienden a rodear a su presa primero. matemáticamente se puede formular de la siguiente manera:

$$D = |C * X_p(t) - X(t)|$$
$$X(t+1) = X_p(t) - A * D$$
$$A = 2ar_1 - r_2$$
$$C = 2r_2$$

Donde:

- t $\rightarrow$ Iteración actual
- $X_p \rightarrow$ Vector posición de la presa
- X $\rightarrow$ Posicion de un lobo
- A $\rightarrow$ Vector con coeficientes
- D $\rightarrow$ Vector con coeficientes
- $r_1 \rightarrow$ vector aleatorio con coef [0,1]
- $r_2 \rightarrow$ vector aleatorio con coef [0,1]
- a $\rightarrow$ parametro que va de [2,0] y decrece linealmente con el paso de las iteraciones

Aunque el proceso de caza es guiado por α, β y δ, en un problema en un espacio de búsqueda abstracto, no sabemos la posición de la presa "presa" (solución, óptima o no), para simular la caza, se asume que la cuspide de la gerarquía estíman mejor la posición de la presa y por consiguiente guían al resto de la manada. Esto se puede formular de la siguiente manera:

- $D_\alpha = |C_1 * X_\alpha(t) - X(t)|$
- $D_\beta = |C_2 * X_\beta(t) - X(t)|$
- $D_\delta = |C_3 * X_\delta(t) - X(t)|$
- $X_1 = X_\alpha(t) - A_1 * D_\alpha$
- $X_2 = X_\beta(t) - A_2 * D_\beta$
- $X_3 = X_\delta(t) - A_3 * D_\delta$
- $X(t+1) = \frac{X_1+X_2+X_3}{3}$

Pseudocódigo

Algorithm 1 Grey Wolf Optimization

```
1: function GWO(n_sol)
2:     Inicializamos la población X_i(i = 1, ..., N_SOL)
3:     func_coste ← create_fitnes_func(datos)
4:     mejor_solucion ← GeneraSolucionAleatoria(datos.n)
5:     mejor_solucion ← BL(datos, mejor_solucion, 2000, k)   ▷ El parametro K controla el tamaño del entorno a explorar
6:     mejor_coste ← func_coste(mejor_solucion)
7:     k ← 1
8:     K_MAX ← datos.n
9:     for i ← 1 to N_SOL − 1 do
10:        sol ← mutacion_ils(mejor_solucion)
11:        sol ← BL(datos, sol, 2000, k)
12:        costo_nuevo ← func_coste(sol)
13:        if costo_nuevo < mejor_coste then
14:            mejor_solucion ← sol
15:            mejor_coste ← costo_nuevo
16:            k ← 1
17:        else
18:            k ← k + 1
19:        end if
20:        k ← (k > K_MAX)?1 : k
21:    end forreturn mejor_solucion
22: end function
```

0.2. Análisis de Rendimientos

0.2.1. 1 % de Evaluaciones

0.2.2. 50 % de Evaluaciones

0.2.3. 100 % de Evaluaciones

0.3. Hibridación y Análisis de Rendimiento

0.3.1. Proceso de Hibridación

0.3.2. Análisis del rendimiento

1 % de Evaluaciones

50 % de Evaluaciones

100 % de Evaluaciones

0.4. Procedimientos

Para el desarrollo de la práctica se ha usado el lenguaje de programación C++. Los motivos de esta decisión son: la familiaridad con el lenguaje de programación y el rendimiento superior a lenguajes de programación de alto nivel, conveniente para el tratamiento de grandes cantidades de datos, o la resolución de problemas computacionalmente intensivos.

Para la compilación del código se ha usado el compilador g++ de GNU. La estructura del proyecto se ha configurado usando CMAKE. Para la ejecución de los programas se ha usado el sistema operativo Linux. Aunque al usar CMAKE se podría generar un proyecto para Windows.

0.4.1. Estructura del proyecto

El proyecto se ha estructurado de la siguiente manera:

```
software
├── memoria.pdf
├── README.md ..... Archivo con indicaciones del proyecto
├── CMakeList.txt ..... Archivo de configuración CMAKE
├── README.md ..... Archivo con instrucciones de ejecución y dependencias
├── .vscode/ ..... Archivos de configuración de vscode
│   ├── c_cpp_properties.json
│   └── settings.json
├── Memoria/ ..... Carpeta con los archivos latex de la memoria de la práctica
├── bin/ ..... Carpeta con los ejecutables del proyecto.
├── build/ ... Carpeta donde se recomienda generar los archivos de compilación de CMAKE en caso de compilar
├── data/ ..... Carpeta con los archivos de datos relativos al proyecto
│   └── instancias/ ..... 1
└── source/ ..... Carpeta con el código del proyecto
    ├── src/ ..... Carpeta con los archivo *.cpp *.cc del proyecto
    ├── demo/ .... Archivos entregados por el profesor para testear la biblioteca RAndom y el uso del reloj interno
    ├── include/ ..... Archivos de cabecera de c++
    └── tools/ ..... Herramientas externas al proyecto
        ├── externo/ ..... Herrmamientas externas descargadas por el alumno
        │   └── fmt/ ..... Libreria de formateo de strings para la salida estandar de c++
        └── profesor/ ..... Librerias dadas por el profesor
```

1: Carpeta con los archivos de datos relativos a las entradas del problema, si se quieren ejecutar nuevos test se deben poner en esta carpeta

0.4.2. Set Up

Se le entregará un archivo .zip llamado software que contendrá la estructura de directorios anterior. La carpeta build estará vacía por cuestiones obvias y en la carpeta bin se hallarán distintos ejecutables, todos del mismo código, compilado para varias plataformas directamente.

Si su plataforma no está entre las pre-compiladas siga leyendo este apartado, sino, salte al apartado de ejecución.

Si no puede ejecutar ninguno de los ejecutables. Deberá tener un compilador de c++ instalado en su computador y el programa ***CMAKE*** y ***make*** para seguir con el tutorial.

Linux

1. Abra una terminal en la raíz del proyecto
2. Si no está creada la carpeta build (que no debería), créela, sino pase al paso 3

```
mkdir build
```

3. Acceda a la carpeta build e inicialice el proyecto

```
cd build
cmake ..
cd ..
cmake -DCMAKE_BUILD_TYPE=Debug -S . -B build
```

4. Compile el programa

```
cmake --build build --target main
```

Windows

El programa se ha escrito pensando en ejecutarse en Linux, si se tiene un sistema Windows lo que se recomienda es usar WSL, instalar el compilador de c++ GNU y CMAKE en esa máquina virtual y ejecutar desde ahí, siguiendo las instrucciones del apartado anterior.

0.4.3. Ejecución

Para la ejecución del programa se ha pensado en pasar los parámetros siempre por línea de comandos. El programa permite configurar que ejecutar y como de la siguiente manera:

Por orden aparecen primeros los obligatorios y después los flags. Si llevan un guion delante, es un flag y, por tanto, optativo

Ejemplos de ejecuciones

0.5. Analisis de Rendimiento

0.6. Bibliografía

Principalmente, documentos de la universidad apoyados del siguiente libro:

https://www.google.com/search?q=The+algorithm+Design+Manual&oq=The+algorithm+Design+Manual&aqs=chrome..69i57.8716j0j1&sourceid=chrome&ie=UTF-8#wptab=si:AMnBZoEG3b_8oGF0zZDE6xv96fMHXP7HJH_MnzBXKd6lQPUm1FQ6wfaQI4rwDvhX1e-GXS_G0MX7E6K2fWcdKuVs64FZpIqGagHPROC0mvbWvcSAQYL-QLu2yIPStw0Zslk%3D