PROBLEMA DE APRENDIZAJE DE PESOS EN CARACTERÍSTICAS
COYOTE OPTIMIZATION ALGORITHM
MEMORIA PRÁCTICA FINAL METAHEURÍSTICAS

Ignacio Garach Vélez MH DG

28 de junio de 2022

Índice

1. Introducción

En esta práctica se va a abordar el estudio de una metaheurística de optimización basada en la adaptación social y mejora generacional de manadas de coyotes. En los últimos años el comportamiento de diversos fenómenos naturales ha despertado un gran interés en el diseño de algoritmos de optimización. Dada la gran diversidad de problemas del campo, las técnicas clásicas como puede ser el descenso de gradiente se han quedado algo estancadas en la consecución de óptimos globales y han comenzado a surgir cientos de estos algoritmos con excelentes resultados y otros no tan buenos. Podría pensarse por tanto que el campo está sobrecargado, sin embargo, los teoremas de No Free Lunch establecen que no existe un algoritmo que pueda adaptarse y obtener buenos resultados en todo problema de optimización, por ello la investigación en este área es aún intensa.

El algoritmo de optimización global que vamos a estudiar combina muchas de las técnicas que se han estudiado a lo largo de las prácticas y de las clases de teoría. Podríamos definirlo en pocas palabras como una combinación de un algoritmo poblacional con técnicas de adaptación social. Además introduce nuevos mecanismos para balancear el compromiso exploración-explotación. Está inspirado en el comportamiento de la especie Canis Latrans, autóctona principalmente de Norteamérica.

En las siguientes secciones se analizará el funcionamiento del algoritmo con especial atención a lo que aporta cada técnica usada al equilibrio entre intensificación y diversificación. Además se adaptará al problema de aprendizaje de pesos de características, del cual se tienen variedad de resultados para comparar debido a la realización de las prácticas de la asignatura. Adelantamos que tendrá un gran comportamiento y se estudiará el porqué.

Finalmente se propondrán y analizarán varias mejoras posibles del algoritmo, basadas en hibridación con búsqueda local (memético) y variación de los métodos de inicialización, cruce e intercambio cultural.

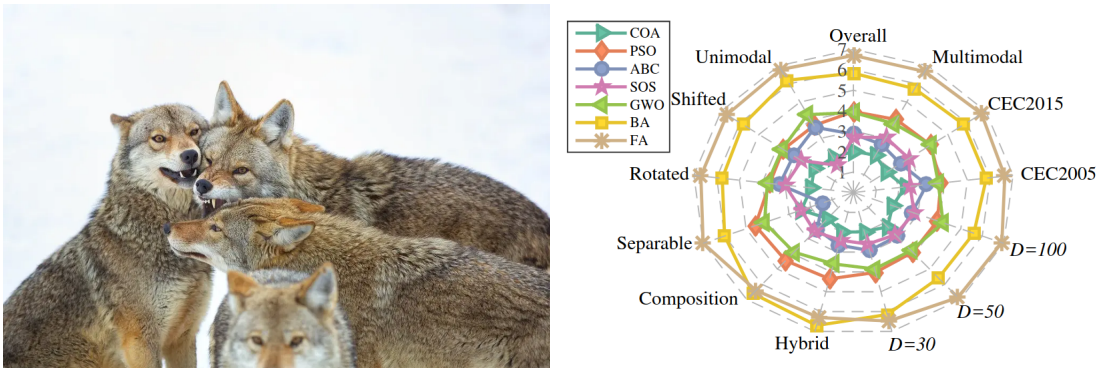


Figura 1: A la derecha resultados del COA en la competición CEC2015 frente a algoritmos estándar como PSO. A la izquierda, artículo del New York Times:"Los coyotes deben de ser de los animales más adaptables del planeta". Ver *Los coyotes ya conquistaron Norteamérica*.

2. Descripción del algoritmo

El algoritmo COA puede clasificarse como un algoritmo evolutivo con inteligencia de colmena. Al contrario que otros algoritmos basados en lobos con importante dominancia y jerarquías marcadas, el algoritmo COA se caracteriza por centrarse en la estructura social de la población por manadas y los intercambios culturales tanto intrínsecos (en la manada), como extrínsecos(con el resto de manadas).

Se define la población de coyotes (soluciones del probema) mediante división en manadas. Se establece N_p como el número de manadas (packs) y N_c como el número de coyotes por manada. Por tanto el tamaño de la población es el producto de estas 2 cantidades.

De acuerdo con las investigaciones de los biólogos, los factores que afectan el comportamiento de los coyotes son diversos: sexo, estátus social, compañeros de manada, hábitat,etc... Para codificarlos se considera un vector de condiciones sociales *soc* que se identifican como las dimensiones del espacio de búsqueda de la optimización, por ello estan limitadas por el dominio del problema. La función objetivo o fitness se calcula a partir de dichas condiciones sociales y se denomina en el contexto del algoritmo como la adaptación del coyote a su entorno.

2.1. Inicialización de la población

El primer paso del algoritmo es la inicialización de los coyotes y de sus condiciones sociales. Para una buena exploración del espacio de búsqueda la inicialización es estocástica pura, se inicializa cada condición social con un valor aleatorio dado por una distribución de probabilidad uniforme en el rango posible de dicha característica.

A continuación se calcula el fitness de cada coyote y se les asigna de forma aleatoria en manadas dadas por los valores N_p y N_c.

2.2. Cálculo de tendencias culturales y coyotes alfa

El algoritmo dados los signos de inteligencia de colmena que ha mostrado la especie, asume que los coyotes en cada manada estan suficientemente organizados como para compartir sus condiciones sociales con sus compañeros y así contribuir a la supervivencia de la manada.

Para calcular esta tendencia en cada manada, el algoritmo utiliza la siguiente fórmula que no hace más que calcular la mediana de cada condición social entre las de todos los coyotes de la manada, O representa a las condiciones condiciones sociales ordenadas. Para facilitar la implementación se tomarán tamaños de manada impares.

$$cult_j^{p,t} = \begin{array}{l} O^{p,t}_{\frac{(N_c+1)}{2}j} \end{array} \quad \text{si } N_c \text{ es impar} \quad . \tag{1}$$

Por otro lado, en esta especie las manadas suelen tener dos coyotes alfa líderes, sin embargo el algoritmo solo tendrá en cuenta uno y este será aquel que este mejor adaptado al entorno, es decir, el que mejor función objetivo tenga.

$$\text{alpha}^{p,t} = \left\{ \text{soc}_c^{p,t} \mid \arg_{c=\{1,2,\ldots,N_c\}} \text{máx}\, f\left(\text{soc}_c^{p,t}\right) \right\} \tag{2}$$

2.3. Intercambio cultural intrínseco

Para representar la interacción cultural entre los elementos de una misma manada, se asume que todo individuo está bajo la influencia del alfa (δ_1) y de la tendencia cultural de la manada (δ_2). Estas se calculan como la diferencia de las condiciones sociales de dos coyotes aleatorios del pack con la del alfa y la tendencia cultural de la población respectivamente.

Por tanto, en cada iteración del algoritmo, las nuevas condiciones sociales de cada coyote de una

manada se calculan en función de las anteriores y de los productos de δ_1 y δ_2 por un número aleatorio entre 0 y 1 cada uno, en resumen mediante la siguiente expresión en la que r_1 y r_2 son los números aleatorios.

$$\begin{aligned} \delta_1 &= \text{alpha}^{p,t} - \text{soc}_{cr_1}^{p,t} \text{ y} \\ \delta_2 &= cult^{p,t} - \text{soc}_{cr_2}^{p,t} \end{aligned} \tag{3}$$

$$new\_\,\text{soc}_c^{p,t} = \text{soc}_c^{p,t} + r_1 \cdot \delta_1 + r_2 \cdot \delta_2 \tag{4}$$

Al finalizar esta actualización es necesario recalcular el fitness de cada coyote y verificar si han mejorado con respecto a la iteración anterior, en caso contrario se vuelven a reestablecer sus antiguas condiciones sociales.

2.4. Nacimiento y muerte

Vamos a definir como funcionarán estos 2 acontecimientos básicos de la vida de los individuos, estamos realmente definiendo el operador de cruce. Será necesario que el algoritmo vaya actualizando la edad de cada coyote según el número de iteraciones para poder realizar esto.

El nacimiento de un nuevo coyote se realiza como la combinación de las condiciones sociales de 2 padres elegidos aleatoriamente en la manada, más una influencia ruidosa del entorno o mutación. Para garantizar cierta herencia siempre se mantiene al menos una condición social aleatoria de cada padre. Para el resto, se define una probabilidad de dispersión y otra de asociación con respecto a la dimensión del problema como:

$$P_s = 1/D$$

$$P_a = \frac{1 - P_s}{2}$$

y se toman la de un padre u otro o una mutación aleatoria en función de la siguiente expresión donde R_j es un valor aleatorio en el rango de la componente en cuestión:

$$pup_j^{p,t} = \begin{cases} \text{soc}_{r_{1,j}j,}^{p,t} & rnd_j < P_a \text{ or } j = j_1 \\ \text{soc}_{r_{2,j}}^{p,j}, & rnd_j > 1 - P_a \text{ or } j = j_2 \\ R_j, & \text{en otro caso} \end{cases} \tag{5}$$

Para mantener estáticos los tamaños de la manada se sincronizará el nacimiento con la muerte. Para hacerlo, en cada nacimiento se calculan los coyotes con peor adaptación que el neonato, si sólo hay uno, el que tiene peor adaptación muere y el cachorro sobrevive. En caso de que haya más de uno peores, se muere el más viejo y sobrevive el cachorro. Finalmente en caso de que el cachorro sea el peor, no sobrevivirá.

2.5. Intercambio cultural extrínseco

Pese a que inicialmente los coyotes se asignan aleatoriamente a una manada, puede ocurrir que a veces un coyote la abandone y se vaya a otra. Para mantener estático el número de coyotes por manada, vamos a considerar que se intercambia con un coyote de otra manada favoreciendo así el intercambio cultural entre las mismas y mejorando la explotación del espacio de búsqueda.

Esto ocurrirá con probabilidad $P_e = 0.005\dot{N}_c^2$, por tanto se observa que si N_c es grande, se hará casi siempre, por tanto vamos a limitar el tamaño de las manadas 14 para no perder progresos de explotación en cada manada. Realmente en nuestra implementación usaremos tamaño 5.

3. Pseudocódigo del algoritmo COA

Atendiendo al diseño del algoritmo descrito en la sección anterior, tendríamos los siguientes pseudocódigos para el algoritmo general, la inicialización, el nacimiento de coyotes y el cálculo de la tendencia cultural:

Coyote Optimization Algorithm(TrainingSet, Pesos)

```
Evaluaciones = 0
Generación = 0
Poblacion = vector<vector<Coyote> >
launchPopulation(Población, NPACKS, NCOYOTES, Pesos.size(), TrainingSet)
Evaluaciones += NPACKS * NCOYOTES

Mientras Evaluaciones < 15000:
    Para cada manada i de la población:
        Calcular coyote alfa de la manada. Buscar el mayor fitness.
        TendenciaCultural = computeCulturalTendency(poblacion[i])

        Para cada coyote j en la manada i:
            RC1 y RC2 = Coyotes aleatorios distintos de j de la manada
            delta1 = Alfa.Pesos - RC1.Pesos
            delta2 = TendenciaCultural.Pesos - RC2.Pesos

            Actualizar los pesos del coyote j:
            R1 y R2 = Nºs aleatorios en [0, 1]
            Poblacion[i][j].Pesos += R1*delta1.Pesos + R2*delta2.Pesos
            Truncar a 0 o 1 si salieron del rango
            Recalcular fitness del Coyote j de la manada i
            Adaptación. Si el fitness no ha mejorado:
                Restaurar pesos antiguos.

            Puppy = Birth(Población[i], Training, Pesos.size(), P_a)
            Calcular coyotes con peor fitness que Puppy.
            Si hay uno:
                Muere y el Puppy lo sustituye.
            Sino y hay más:
                Muere el mayor y el Puppy lo sustituye.
            Sino:
                El Puppy no sobrevive
        Fin-Para cada
    Fin-Para cada
    Para todo Coyote aumentar edad en 1.

    Con Probabilidad 0.005*Nc*Nc:
        Cambiar un coyote aleatorio de una población por otro de otra.
Fin-Mientras

Calcular Alfa de cada Manada.
Calcular Alfa con mejor fitness.
Almacenar en Pesos los del mejor alfa.
return Generación
```

LaunchPopulation(Población[][], NP, NC, tamaño, TrainimgSet)

```
Para i menor que NP:
    vector<Coyote> Manada
    Para j menor que NC:
        Inicializar Coyote c:
            c.Edad = 0
            c.Pesos = Aleatorios uniformes en el espacio de búsqueda
            Calcular Fitness de c
        Añadir c a Manada
    Fin-Para cada
    Añadir Manada a Población
Fin-Para cada
```

Birth(Coyotes[] pack)

```
Tamaño = Pack[0].Pesos.size()
Para toda Condición Social de los Coyotes (0, Tamaño):
    Ordenar las de la manada en orden ascendente en un vector cada una.
    Medianas[i] = SocialConditions[i][(SocialConditions[i].size()/2)]
Fin-Para cada
Return vector de medianas.
```

Birth(Coyotes[] pack, Training, Tamanio, P_a)

```
RC1 y RC2 = Coyotes aleatorios distintos de la manada
D1 y D2 = Dimensiones aleatorias distintas del vector de pesos

Generar cachorro de coyote:
    Puppy.Edad = 0
    Puppy.Pesos[D1] = RC1.Pesos[D1]
    Puppy.Pesos[D2] = RC2.Pesos[D2]
    Para cada dimensión d distinta de D1 y D2:
        aleatorio = Nº Aleatorio en [0,1]
        Si aleatorio < Pa:
            Puppy.Pesos[d] = RC1.Pesos[d]
        Sino:
            Si aleatorio > (1 - Pa):
                Puppy.Pesos[d] = RC2.Pesos[d]
            Sino:
                Puppy.Pesos[d] = Número aleatorio en espacio de búsqueda.
    Fin-Para cada
    Calcular Puppy.Fitness
Return Puppy
```

4. Descripción del problema APC

Como preliminares introducimos brevemente el problema de clasificación. Se trata de construir un modelo que a partir de las características (datos descritos numéricamente) de un objeto sea capaz de predecir su clase (tipo de objeto entre un número finito de posibilidades). Para ello se entrena al modelo con un conjunto de datos correctamente clasificados (aprendizaje supervisado) y despues se prueba su valía con un conjunto independiente de test. Existen numerosos enfoques para resolver este problema con gran éxito, nosotros nos vamos a centrar en probablemente el más sencillo, se trata del algoritmo $1 - NN$ conocido popularmente como el del vecino más cercano.

Este algoritmo, simplemente almacena los datos de entrenamiento en memoria junto con su clase asociada, matemáticamente esto se puede representar como un conjunto de puntos de $\mathbf{R}^n$ junto con un entero representando su clase asociada. Y para cada punto que quiera clasificar, calcula la distancia euclídea con todos los puntos de entrenamiento y escoge la clase del punto para el cuál es mínima, su vecino más cercano.
Nuestro trabajo va a consistir en conseguir pesos que ponderen la importancia en la distancia final de cada característica en cada caso concreto de entrenamiento, es decir si la distancia euclídea usual es:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

nosotros consideramos el cálculo de los w_i tal que sea:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}$$

Nuestro objetivo es dual, por un lado queremos conseguir buenas tasas de acierto en el problema de clasificación y por otro obtener pesos bajos en las características menos influyentes para la clasificación. El problema de aprendizaje de pesos en características es determinar dichos pesos a partir del conjunto de datos de entrenamiento.
En la primera práctica se implementaron un algoritmo de búsqueda local con trayectorias simples y un algoritmo greedy (voraz) y se comparó su desempeño con el algoritmo 1-NN habitual.

En la segunda se implementaron varias versiones de algoritmos genéticos y meméticos y en la tercera una serie de algoritmos basados en búsqueda por trayectorias.

En esta última práctica vamos a implementar el algoritmo COA descrito, varios algoritmos meméticos basados en el mismo y dos versiones mejoradas, una basada en números caóticos y otra con una estrategia de adaptación social diferente. Veremos que el gran potencial del algoritmo original deja a todas estas variantes con resultados similares, aunque algunos algo mejores sobre los datasets considerados.

5. Datasets considerados

Para poder probar los algoritmos desarrollados se utilizarán los siguientes conjuntos de datos, todos ellos con clase binaria y algunos de ellos con distribución de clases no homogénea.

5.1. Ionosphere

Conjunto de datos recogidos en Goose Bay por un sistema con 16 antenas de alta frecuencia, el objetivo era captar electrones libres en la ionosfera, una devolución positiva indicaba la existencia de algún tipo de estructura. Los datos son las partes reales e imaginarias de 17 señales electromagnéticas complejas.

- 352 instancias
- 34 características
- 2 clases

5.2. Parkinsons

Conjunto de datos recogidos por el Centro Nacional de estudio de la voz de Oxford. Se trata de medidas asociadas a 195 grabaciones de voz de personas, algunas de ellas con enfermedad de Parkinson. La clase sería la presencia de dicha enfermedad.

- 195 instancias
- 22 características
- 2 clases

5.3. Spectf-Heart

Conjunto de datos de medidas asociadas a imágenes tomadas por tomografía computerizada cardíaca tomadas por el colegio de médicos de Ohio. Cada paciente se clasifica entre normal o con anomalías.

- 267 instancias
- 44 características
- 2 clases

6. Elementos comunes de prácticas anteriores

En este apartado vamos a describir las funciones que han sido comunes durante todo el desarrollo de las prácticas de la asignatura, incluida esta práctica final:

- Para representar nuestros conjuntos de entrenamiento y test utilizaremos un vector de la STL de una nueva clase a la que llamaremos Instance que representará una instancia del problema, es decir su vector de características y su clase.
- Los datos proporcionados se encuentran en formato ARFF, un conjunto de cabeceras indicando las características y a continuación cada instancia es una fila con espacios como separadores y con la clase al final de la línea. Esto facilita su lectura para su posterior postprocesado. Hemos desarrollado para ello la funcion *read_arff* que recibe como parámetro el nombre del archivo y un par de vectores donde se almacenan las características y su clase. A posteriori en la función principal se genera el *vector de Instance* y se rellena con dichos datos.
- Los datos requieren ser normalizados para que no influyan las distintas escalas en nuestros algoritmos y no distorsionen los datos. Para esto se ha implementado la función *normalize* que modifica el vector de instancias, mediante escalado y translaciones hasta el intervalo $[0, 1]$, para ello se hace uso de la siguiente fórmula en cada característica:

$$x_i = \frac{x_i - mín(x_i)}{máx(x_i) - mín(x_i)}$$

- Para el cálculo de las distancias se implementa la función *weightedDistance* que recibe un vector de pesos y 2 vectores de características y devuelve la distancia ponderando por sus pesos.
- Para nuestro problema de aprendizaje de pesos, utilizaremos una versión del clasificador 1-NN que permitirá su generalización con la utilización de pesos. Recibe un valor booleano indicando si debe utilizar un vector de pesos al calcular las distancias. Atendiendo a estas características tendríamos el siguiente pseudocódigo:

Clasificador 1-NN (Instance, TrainingSet, Weighted?, Weights[])

Si no weighted
Entonces: Inicializar a 1 todos los pesos W
Para toda instancia de TrainingSet distinta de Instance (Leave One Out)
Hacer:

Calcular la distancia con respecto a Instance.
Si la distancia disminuye respecto al anterior guardamos su posición en pos.

Fin - Para todo

Devolver la clase de TrainingSet[pos]

6.1. Métricas utilizadas

Como medidas de la bondad de funcionamiento de los algoritmos se van a tomar las siguientes 4, relacionadas con los parámetros que se quieren optimizar, a saber, exactitud de clasificación, simplicidad y tiempo:

%**class** : Porcentaje de acierto en la labor de clasificación. Se utiliza el método de prueba basado en entrenamiento y test de 5 particiones cruzadas (se crean mediante la función *makeKFolds* inspirada en las prácticas de la asignatura de aprendizaje automático y que divide de forma homogénea y equilibrada las clases) y se realiza la media.

$$\%_{class} = 100\frac{\text{nº de instancias clasificadas correctamente en Test}}{\text{nº de instancias en Test}}$$

%**red** : Porcentaje de características que podrían obviarse, vamos a considerar aquellas cuyo peso asociado sea menor a 0.1. En efecto, para el algoritmo base no se reduce nada, porque todos los pesos son 1.

$$\%_{red} = 100\frac{\text{nº de características con peso menor que 0.1}}{\text{nº de características}}$$

Agregado : Valor de la función objetivo de nuestro problema. Se trata del promedio ponderado de las tasas de acierto y reducción, en nuestro caso consideramos una ponderación equiparada luego es simplemente la media de ambos valores

Tiempo : Tiempo de ejecución del algoritmo. En nuestro caso la media de 5 ejecuciones.

Para la implementación del cálculo de la tasa de clasificación se utilizará la función *score*, simplemente recorre el vector de predicciones del clasificador y cuenta los aciertos, devolviendo la división entre el número de instancias.
Para el cálculo de la tasa de reducción se hace algo análogo en *reduction* con el vector de pesos obtenido contando todos aquellos menores que 0.1 y devolviendo la división entre el número de características.
La función objetivo agregada se implementa de forma génerica con un parámetro α que pondera la importancia de precisión y simplicidad, aunque en nuestro caso usaremos $\alpha = 0.5$ luego coincidirá con la media de *score* y *reduction*, esto se realiza en *funcionObjetivo*.
Por último para el cálculo de tiempos se usará la librería *chrono* de C++ tomando el tiempo actual antes y después de las ejecuciones con *high_resolution_clock* y calculando su diferencia. Lo mediremos en segundos.

7. Algoritmos meméticos. Hibridación.

Los algoritmos meméticos son técnicas de optimización que combinan de forma sinérgica elementos tomados de otras metaheurísticas junto con técnicas de búsqueda local. Son un avance en la mejora de calidad de soluciones pues combinan a los algoritmos de búsqueda local que son buenos explotadores y malos exploradores (recorren muy bien una zona del espacio de soluciones pero pueden estancarse en óptimos locales) con los algoritmos evolutivos que son buenos exploradores pero malos explotadores por su factor aleatorio. De esta manera conseguimos mezclar lo mejor de ambos mundos obteniendo una exploración muy homogénea del espacio de búsqueda.

En nuestro caso vamos a hibridar la búsqueda local con el algoritmo COA.

Se realizará un estudio de tres versiones en las cuáles se aplicará la búsqueda local cada 30 generaciones y variando desde que coyotes se inicia la misma:

- CM1: Se lanza la búsqueda desde los coyotes alfa de cada manada.
- CM2: Se lanza la búsqueda desde los coyotes alfa y los coyotes omega (los que tienen peor fitness) de cada manada.
- CM3: Se lanza la búsqueda sólo al finalizar las generaciones desde todos los alfas, se considera un refinamiento de soluciones más que un algoritmo memético.

8. Propuestas de mejora

Se van a analizar 2 opciones para mejorar el algoritmo, por un lado, el uso de números caóticos en lugar de pseudoaleatorios en el cálculo de la inicialización y la adaptación de los coyotes y por otro un nuevo método de adaptación social que la autoregule de alguna forma.

8.1. Uso de funciones caóticas

La teoría del caos es una rama de las matemáticas que intenta estudiar algunos tipos de sistemas dinámicos no lineales que son muy sensibles a las variaciones en las condiciones iniciales, en el sentido de que pequeñas variaciones en ellas pueden provocar grandes diferencias en el comportamiento futuro, haciendo que sea casi aleatorio, caótico. Pero estos sistemas son totalmente deterministas, es decir; su comportamiento puede ser determinado conociendo sus condiciones iniciales, aunque se necesitarían cálculos con altísima precisión.

Tienen propiedades matemáticas muy interesantes como la ergodicidad y la generación de objetos fractales en su imagen.

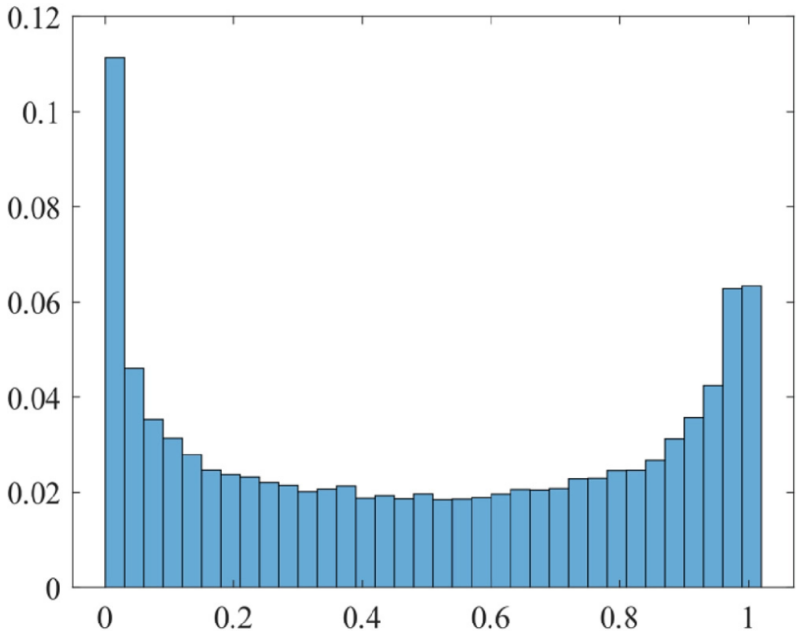
Nosotros vamos a utilizar una función que describe un sistema de este tipo para sustituir a los tradicionales números pseudoaleatorios. En numerosos artículos publicados se ha demostrado empíricamente que se mejoran los resultados de los algoritmos con esta sustitución, aunque es complicado establecer una demostración teórica debido al Teorema de No Free Lunch y a la dificultad de realizar un buen test de significancia estadística.

Usaremos la función logística siguiente con $\lambda = 4$:

$$x_{n+1} = f(x_n)$$

$$f(x) = \lambda x(1 - x)$$

que se distribuye aproximadamente de la siguiente forma:



8.2. Nuevo método de adaptación social

Ante los resultados tan similares del algoritmo original y sus versiones meméticas podemos plantearnos si el algoritmo converge prematuramente a una solución óptima local, para tratar de resolver esto, podemos hacer intentos como utilizar funciones caóticas en lugar de la aleatoriedad o podemos cambiar el modo de la adaptación social para que no mejore tanto a los individuos y sobre todo que no los haga muy homogéneos al principio de las generaciones estropeando la capacidad exploratoria, la cual es muy necesaria en etapas tempranas.

En lugar de los números aleatorios de la expresión siguiente, donde recordemos los deltas representan al alfa y a la tendencia cultural de la mandad:

$$new\_\,\mathrm{soc}_c^{p,t} = \mathrm{soc}_c^{p,t} + r_1 \cdot \delta_1 + r_2 \cdot \delta_2 \tag{6}$$

Proponemos la siguiente forma de adaptación social, utilizando los fitness de cada manada para calcular r_1 y r_2:

$$r1 = \frac{\text{meanfit}}{\text{bestfit}} \qquad\qquad r2 = \frac{\text{worstfit}}{\text{meanfit}}$$

Esta idea trata de detener la convergencia prematura y aumentar la diversificación exploratoria mediante una adaptación mas sostenida y menos brusca de cada manada.

Si en una manada hay un individuo alfa que es demasiado bueno con respecto al resto de individuos, la constante r_1 será más pequeña haciendo que las condiciones sociales del alfa no afecten tanto a los individuos y alejandolos de posibles óptimos locales en instancias prematuras de la ejecución.

De una forma análoga, si hay un individuo omega (consideremoslo como aquel que tiene peor fitness de la manada) muy malo respecto al resto, la constante r_2 se hace pequeña provocando 2 efectos muy beneficiosos. Por un lado, el omega va a mejorar, pero no tanto como parecerse mucho al resto de individuos de la manada, proporcionando exploración de nuevas zonas y por otro, la mala calidad del omega que se ha utilizado en el cálculo de la tendencia cultural va a afectar en menor medida al resto de individuos.

9. Implementación y guía de usuario

Para la implementación se ha utilizado el lenguaje C++, las librerías habituales de entrada y salida y lectura de fichero, toma de tiempos y funciones auxiliares. Se usó un generador de números aleatorios inicializado con la semilla 2022.
La adaptación en este caso del algoritmo al problema de aprendizaje de pesos es tremendamente sencilla, se identifican las condiciones sociales con los pesos y el cálculo del fitness se hace con las tasas de reducción y clasificación idénticas a las demás prácticas.
Se ha especificado por secciones la estructura en funciones del código, por simplicidad se ha optado por implementar todo en un mismo archivo sin usar cabeceras.
Para la implementación de los Coyotes se ha aprovechado el struct Cromosoma de la práctica 2 cambiandole el nombre, mantenemos los pesos y el fitness y se añade un campo edad para la estrategia de nacimiento y muerte.
Para la implementación de la función logística se ha usado la variable global valor para ir almacenando el estado anterior del sistema y poder seguir iterando. Se utiliza optimización en la compilación con g++, concretamente se usa la siguiente orden para compilar:

```
g++ -std=c++11 -O3 main.cpp
```

Para ejecutar se llama en terminal a ./a.out seguido de la semilla que se desee.

10. Análisis y presentación de resultados

En la tabla general se han incluido además de los resultados de las 6 variantes de COA implementadas, las mejores variantes de los algoritmos genéticos de la P2 y los algoritmos de la P1 así como los de la práctica 3 de búsqueda por trayectorias. Es reseñable que la labor de aprendizaje de pesos en Parkinsons parece la más fácil de los datasets ya que obtiene muy buenos resultados en todos los algoritmos con cierta complejidad.

En general, resulta que los algoritmos de esta práctica final en los primeros 2 datasets mejoran ligeramente a los meméticos y genéticos basados en poblaciones y igualan y superan por la mínima al enfriamiento simulado. En el tercer dataset de Spectf-Heart la mejora es muy grande, consigue obtener tasas de clasificación tan buenas como los métodos genéticos y de trayectorias pero la gran tara de los mismos que era la tasa de reducción se ve solventada aumentando enormemente con todas las variantes, esto provoca que cuando hasta ahora lo mejor obtenido era un fitness de 76, ahora todas las variantes no bajan de 82 y la mejor obtiene casi 86, una mejora de más de un 10 por ciento. Los resultados que hemos obtenido globalmente indican que en dos datasets el mejor resultado lo ha tenido la variante caótica y en el otro la mejorada por nueva adaptación social, esto no hace más que recordarnos que no existe un algoritmo universal y debemos hacer un estudio en cada problema dependiente de su dominio y características.

Una metaheurística es buena, no por si misma sino por su adaptabilidad a distintos datasets y superficies de optimización, debe encontrarse por tanto en cada caso el correcto compromiso entre exploración y explotación.

Cuadro 1: Cuadro resumen de resultados generales

	Ionosphere				**Parkinsons**				**Spectf-heart**			
	%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
1-NN	86.26	0.00	43.13	0.002	96.00	0.00	48.00	0.002	65.90	0.00	32.95	0.002
RELIEF	86.54	2.94	44.74	0.01	96.00	0.00	48.00	0.002	73.35	21.82	47.58	0.01
BL	88.88	86.47	87.68	18.79	92.93	82.73	87.83	33.50	72.77	70.91	71.84	23.16
AGG1	85.50	69.41	77.46	119.55	89.86	81.82	85.84	31.34	73.64	60.45	67.05	126.69
AGE2	89.39	98.82	94.11	120.38	92.50	99.09	95.80	31.52	73.64	77.27	75.46	127.11
AM1	92.26	89.41	90.84	123.70	92.43	92.73	92.58	31.15	76.51	72.73	74.62	130.72
BMB	89.67	87.06	88.37	120.21	92.43	95.45	93.94	29.43	71.90	74.09	73.00	124.13
ILS	89.44	92.35	90.90	95.08	94.00	97.27	95.64	16.88	77.93	75.45	76.69	114.01
ES	87.11	97.65	92.38	118.53	95.50	100.00	97.75	30.78	72.48	61.82	67.15	15.10
ILS-ES	90.27	100.00	95.14	118.39	94.00	97.27	95.64	10.16	77.95	66.36	72.15	38.22
COA	88.91	99.41	94.16	118.16	94.50	100.00	97.25	30.95	74.51	93.64	84.07	124.75
CCOA	90.85	100.00	**95.43**	126.92	94.50	100.00	97.25	33.95	79.36	92.27	**85.82**	134.64
COAM	88.81	99.41	94.11	128.42	95.50	100.00	**97.75**	33.76	73.64	90.91	82.27	137.34
CM1	87.12	100.00	93.56	117.06	92.93	100.00	96.46	30.88	77.94	91.82	84.88	125.57
CM2	89.41	100.00	94.70	118.43	93.36	100.00	96.68	31.18	73.93	92.27	83.10	131.46
CM3	87.98	100.00	93.99	134.53	95.00	100.00	97.50	35.62	75.06	90.45	82.76	143.50

Las siglas de esta práctica hacen referencia en orden al algoritmo COA básico, a su versión caótica, a su versión mejorada mediante nueva adaptación social y a sus 3 versiones meméticas. Vamos ahora a ir desgranando ahora los resultados de todas las versiones.

10.1. Algoritmo COA

Los algoritmos se han ejecutado sobre las particiones de Cross Validation con un número de manadas y de coyotes por manada igual a 5, además para ser comparables con los genéticos se ha utilizado el mismo criterio de parada por 15000 evaluciones

El algoritmo obtiene tasas de reducción muy altas sobre todos los datasets y tasas de clasificación muy buenas, los resultados estan al nivel de los genéticos y los superan en muchos casos. Se obtienen resultados muy parecidos a enfriamiento simulado que era de los mejores algoritmos utilizados, luego parece que este algoritmo consigue adecuarse muy bien a los datasets explotando bastante bien muy buenas zonas de búsqueda. Además los resultados sobre Spectf-Heart son sorprendentes, son muy buenos y especificamente las tasas de reducción se disparan en comparación todos los demás algoritmos, esto puede deberse a que una mayor diversificación consigue llegar de mejor forma a soluciones prometedoras.

Cuadro 2: Cuadro de resultados para COA

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
90.14	100.00	95.07	116.64	92.50	100.00	96.25	30.42	77.14	97.73	87.44	124.13
88.73	100.00	94.37	116.75	92.50	100.00	96.25	30.68	77.14	93.18	85.16	124.58
90.14	100.00	95.07	117.29	92.50	100.00	96.25	30.75	72.86	90.91	81.88	125.14
84.51	100.00	92.25	117.95	95.00	100.00	97.50	30.53	68.57	93.18	80.88	124.65
91.04	97.06	94.05	122.15	100.00	100.00	100.00	32.36	76.81	93.18	85.00	125.25
88.91	99.41	94.16	118.16	94.50	100.00	97.25	30.95	74.51	93.64	84.07	124.75

10.2. Variante caótica CCOA

La resultados de la variante caótica muestran el brillante efecto de la utilización de números caóticos en lugar de pseudoaleatorios. Este algoritmo consigue los mejores resultados absolutos en Ionosphere y Spectf-Heart, por tanto parece que el efecto del caos, permite evitar caer en óptimos locales de forma temprana y lo hace mejor que los números aleatorios explorando zonas más prometedoras antes. Sin embargo, esto podría haber sido casualidad para estos datasets, habría que realizar un test riguroso de significancia estadística a larga escala.

Cuadro 3: Cuadro de resultados para Chaotic COA

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
92.96	100.00	96.48	126.13	95.00	100.00	97.50	32.82	72.86	93.18	83.02	134.72
91.55	100.00	95.77	125.49	90.00	100.00	95.00	33.47	84.29	97.73	91.01	134.47
91.55	100.00	95.77	126.21	100.00	100.00	100.00	33.82	81.43	84.09	82.76	133.74
90.14	100.00	95.07	125.93	87.50	100.00	93.75	33.84	81.43	88.64	85.03	134.82
88.06	100.00	94.03	130.86	100.00	100.00	100.00	35.80	76.81	97.73	87.27	135.43
90.85	100.00	95.43	126.92	94.50	100.00	97.25	33.95	79.36	92.27	85.82	134.64

10.3. Variante mejorada COAM

La variante mejorada mantiene resultados similarmente buenos a los del algoritmo original por lo que podemos considerarlo una variante prometedora y que podría usarse en otras situaciones. Además consigue que en el dataset Parkinsons se obtengan los mejores resultados absolutos, algo que ni siquiera COA conseguía. Por tanto podemos considerar que el método basado en los fitness de la manada consigue evitar convergencia prematura a zonas que contienen a óptimos mas globales.

Si en una manada hay individuos muy distintos al resto conseguimos que las condiciones sociales del alfa o del omega no afecten tanto a los individuos y nos alejamos de posibles óptimos locales en instancias prematuras de la ejecución.

Cuadro 4: Cuadro de resultados para COA Mejorado

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
88.73	97.06	92.90	127.76	95.00	100.00	97.50	33.55	72.86	95.45	84.16	134.75
92.96	100.00	96.48	128.11	95.00	100.00	97.50	33.16	74.29	93.18	83.73	136.80
90.14	100.00	95.07	127.32	97.50	100.00	98.75	33.28	68.57	90.91	79.74	138.49
90.14	100.00	95.07	127.21	90.00	100.00	95.00	33.36	78.57	86.36	82.47	137.93
82.09	100.00	91.04	131.70	100.00	100.00	100.00	35.43	73.91	88.64	81.27	138.75
88.81	99.41	94.11	128.42	95.50	100.00	97.75	33.76	73.64	90.91	82.27	137.34

10.4. Algoritmos meméticos

Finalmente se consideran los resultados de los algoritmos meméticos que proporcionan unas conclusiones muy interesantes sobre el algoritmo original. Los tres métodos, el de alfas, el de alfas y omegas y el del refinamiento final obtienen resultados muy similares entre sí y muy parecidos al algoritmo original, no lo consiguen superar prácticamente. Entre ellos no hay ninguno que se destaque, cada uno consigue mejores resultados en un dataset distinto.

Es muy reseñable que casi no consigan mejorar al algoritmo original, mucho menos de las variantes mejoradas. Esto nos muestra la increíble capacidad del algoritmo, sobre todo en cuanto a explotación porque añadirle una búsqueda local en los mejores alfas no consigue mejores resultados que él. Esto unido a la capacidad exploratoria demostrada en la mejora para todos los datasets que no conseguían sobrepasar un umbral claro, y a todos los métodos de exploración mediante intercambio cultural explicados, nos proporciona un algoritmo muy equilibrado que podría aplicarse a todo tipo de problemas con probables buenos resultados.

En un futuro podrían plantearse diversas mejoras como coyotes solitarios que exploren el espacio cambiando de manada continuamente, algo que ya anticipan los autores en el artículo y podría hacerse un estudio en profundidad de otras funciones caóticas distintas a la logística utilizada para mejorar aún más el algoritmo.

Cuadro 5: Cuadro de resultados para COA memético 1

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
85.92	100.00	92.96	117.00	90.00	100.00	95.00	30.45	71.43	93.18	82.31	125.45
88.73	100.00	94.37	116.04	92.50	100.00	96.25	30.40	75.71	90.91	83.31	125.70
88.73	100.00	94.37	115.42	92.50	100.00	96.25	30.41	81.43	93.18	87.31	125.25
90.14	100.00	95.07	117.17	92.50	100.00	96.25	30.53	81.43	90.91	86.17	125.87
82.09	100.00	91.04	119.65	97.14	100.00	98.57	32.60	79.71	90.91	85.31	125.60
87.12	100.00	93.56	117.06	92.93	100.00	96.46	30.88	77.94	91.82	84.88	125.57

Cuadro 6: Cuadro de resultados para COA memético 2

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
85.92	100.00	92.96	117.54	90.00	100.00	95.00	30.76	71.43	93.18	82.31	131.63
91.55	100.00	95.77	116.20	92.50	100.00	96.25	30.70	74.29	95.45	84.87	131.00
95.77	100.00	97.89	117.72	95.00	100.00	97.50	30.73	75.71	93.18	84.45	131.55
88.73	100.00	94.37	119.03	95.00	100.00	97.50	30.77	71.43	93.18	82.31	130.88
85.07	100.00	92.54	121.63	94.29	100.00	97.14	32.92	76.81	86.36	81.59	132.24
89.41	100.00	94.70	118.43	93.36	100.00	96.68	31.18	73.93	92.27	83.10	131.46

Cuadro 7: Cuadro de resultados para COA memético 3

Ionosphere				**Parkinsons**				**Spectf-heart**			
%cl	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**	***%cl***	***%red***	***Agr.***	**T**
90.14	100.00	95.07	132.70	92.50	100.00	96.25	35.13	77.14	97.73	87.44	143.12
88.73	100.00	94.37	132.64	95.00	100.00	97.50	35.45	74.29	86.36	80.32	143.47
87.32	100.00	93.66	133.32	95.00	100.00	97.50	35.07	75.71	88.64	82.18	143.33
90.14	100.00	95.07	134.89	92.50	100.00	96.25	35.16	75.71	86.36	81.04	143.49
83.58	100.00	91.79	139.12	100.00	100.00	100.00	37.28	72.46	93.18	82.82	144.11
87.98	100.00	93.99	134.53	95.00	100.00	97.50	35.62	75.06	90.45	82.76	143.50

11. Bibliografía

• J. Pierezan and L. Dos Santos Coelho. Coyote Optimization Algorithm: A New Metaheuristic for Global Optimization Problems

• Iannick Gagnon, Alain April and Alain Abran. An investigation of the effects of chaotic maps on the performance of metaheuristics

• Dixiong Yang, Zhenjun Liu, Jilei Zhou. Chaos optimization algorithms based on chaotic maps with different probability distribution and search speed for global optimization

• Teoría del caos Wikipedia

• Gawali, M.B., Gawali, S.S. Development of improved coyote optimization with deep neural network for intelligent skill knowledge transfer for human to robot interaction.