METAHEURÍSTICAS

Práctica alternativa a examen:

Coronavirus Optimization Algorithm: Un algoritmo bioinspirado basado en el modelo de propagación del Covid-19

UNIVERSIDAD DE GRANADA

Ignacio Yuste López, 54141533Q

ignacioyustel@correo.ugr.es

Grupo 2

Índice

1. Elección de una MHe.

1.1. Resumen.

Los modelos *bioinspirados* imitan comportamientos de la naturaleza y son conocidos por sus aplicaciones exitosas en las hibridaciones para encontrar parámetros en modelos de aprendizaje automático. Los virus pueden infectar a las personas y aquellas pueden morir, infectar a otras personas o simplemente recuperarse tras pasar la enfermedad. Este comportamiento se suele modelar a partir de un modelo *SIR* (S susceptible, I infectados, R recuperados).

Las metaheurísticas deben tratar con espacios de búsqueda enormes, incluso infinitos, y deben encontrar soluciones cercanas a las óptimas en el menos número de ejecuciones posible. La rápida propagación del coronavirus junto con su habilidad de infectar la mayoría de los países de una manera increíblemente rápida ha inspirado esta metaheurística, llamada *Coronavirus Optimization Algorithm* (CVOA).

Las principales ventajas de CVOA con respecto a otras propuestas similares se pueden resumir en los siguientes puntos:

1. Las estadísticas del coronavirus son conocidas por la comunidad científica. Los índices de propagación, de infección, la tasa de mortalidad o la probabilidad de reinfección son conocidas. CVOA está parametrizado con estos valores, de tal manera que los usuarios usen la mejor configuración para el algoritmo.
2. CVOA puede parar la exploración tras varias iteraciones, sin necesidad de configurarlo. Esto se debe a que el número de individuos infectados crece exponencialmente al principio, sin embargo, tras cierto número de iteraciones, el número de individuos infectados comienza a decrecer hasta que no haya ninguno.
3. La rápida tasa de infección del coronavirus es útil para explorar regiones prometedoras, haciendo uso de una estrategia de intensificación. Si se quisiera explorar todas las regiones se podría usar una estrategia de diversificación a través de paralelizar la búsqueda.

Una de las mayores limitaciones del modelo no incluye las estadísticas tras el desarrollo de la vacuna, ya que esta no existía aún. Esto hace que el contagio se produzca de manera exponencial al comienzo, lo cual se traduce en un aumento exponencial del tiempo de ejecución para estas iteraciones.

El trabajo original [1] hace una hibridación con LTSM (Long Short Term Memory networks). CVOA se ha usado para encontrar los valores óptimos de los hiperparámetros de la arquitectura LTSM, el cual es ampliamente usado por las redes neuronales recurrentes en el campo de *deep learning*. Usaron datos del consumo de electricidad de España, obteniendo unos resultados un 0.45% mejores que otros métodos ya establecidos, como *random forest, gradient-boost trees*, regresión linear o *deep-learning* optimizados con otras metaheurísticas.

Pasos de CVOA

1. **Generar la población inicial**. La población inicial consiste en una solución generada aleatoriamente, llamada Paciente Cero (PZ).
2. **Esparcir la enfermedad**. Dependiendo del individuo se dan varios casos:
   A. Algunos individuos mueren, con una probabilidad del 5%. Estos individuos no pueden infectar más.
   B. Los supervivientes podrán infectar a nuevos individuos. Hay dos tipos de infección, dependiendo de la probabilidad de ser un individuo súper contagioso (10%):
      i. Contagio normal. Los individuos normales pueden contagiar entre 0 y 5 nuevos individuos
      ii. Súper contagio. Los individuos súper contagiosos pueden contagiar entre 0 y 15 nuevos individuos.
   C. Para aumentar la diversificación se introduce el concepto de *individuo viajero.* Un individuo viajero infectará a nuevos individuos que sean más diferentes que el original, aumentando así la diversificación. Por tanto, hay una probabilidad de ser viajero (10%).
3. **Actualizar las poblaciones**. Se mantienen tres poblaciones distintas durante la ejecución:
   A. Muertos: Los individuos de esta población no pueden volver a usarse.
   B. Recuperados. Tras cada iteración, los individuos infectados pasan a la esta población (tras infectar a nuevos individuos). Sabemos que existe una probabilidad de volver a contagiarse, por tanto, un individuo perteneciente a esta población puede volver a introducirse en la población de infectados bajo una probabilidad (14%). También se considera que los individuos pueden aislarse haciendo cuarentena, evitando así el contagio de nuevos individuos. Por tanto, existe una probabilidad de que un individuo esté haciendo cuarentena (50%) y no pueda contagiar a nuevos individuos.
   C. Nuevos infectados. Agrupa todos los individuos infectados nuevos en cada iteración.
4. **Criterio de parada.** Este algoritmo tiene la capacidad de parar por sí solo, sin indicar un número máximo de iteraciones. Este se debe a que las poblaciones de recuperados y muertos están constantemente creciendo, por lo cual llega un punto donde no se generan nuevos infectados. Se espera que el número de nuevos infectados crezca durante un número de iteraciones, sin embargo, tras un número de iteraciones esta población irá decreciendo ya que los individuos estarán recuperados o muertos. Cuando no haya nuevos infectados el algoritmo para. Por otra parte, se puede añadir un número de iteraciones límite imitando la duración de la pandemia (*PANDEMIC_DURATION).*

2. Adaptación de CVOA al problema MDD.

Se ha implementado una clase CVOA con el fin de facilitar el paso de parámetros entre funciones del algoritmo. Esta contiene la definición de los distintos métodos y valores necesarios para esta metaheurística, además de algunas funciones extra para simplificar el código.

La definición de los parámetros de la solución es como sigue:

- *solución*: Conjunto solución. Es un subconjunto del espacio de búsqueda que cumple las condiciones de tener un tamaño *m* y no tener ningún elemento repetido, buscando tener la mínima dispersión posible entre ellos.
- *Poblaciones:* Una población consiste en una matriz de nxm donde cada fila es una posible solución. Para las poblaciones infectados y nuevos infectados existe una matriz paralela la cual contiene las distancias de cada elemento al resto de elementos de la solución.
- *Datos*: Matriz cuadrada que contiene las distancias entre todas las localizaciones. En mi implementación cada distancia está duplicada, ya que se trata de una matriz simétrica por comodidad de implementación.
- *m*: Número de elementos cuya dispersión debe ser lo mínima posible
- *Probabilidades*: Conjunto de constantes que almacenan la configuración de probabilidades previamente descritas
- *Variables auxiliares:* Variables como el número de iteraciones del algoritmo principal o el número de evaluaciones totales ejecutadas.

Se han implementado dos versiones híbridas del algoritmo para su posterior análisis. Se han usado los algoritmos BLM y ES implementados en las prácticas anteriores.

La hibridación ha consistido en ambos casos en aplicar la búsqueda local al mejor individuo de la población de nuevos infectados en cada iteración.

2.1. Pseudocódigos.

Función CVOA:

```
define infectedPopulation, newInfectedPopulation as set of Individual
define dead, recovered as list of Individual
define PZ, bestIndividual, currentBestIndividual, aux as Individual
define time as integer
define bestSolutionFitness, currentBestFitness as real

time <- 0
PZ <- InfectPatientZero()
infectedPopulation <- PZ
bestIndividual <- PZ

while time < PANDEMIC_DURATION and sizeof(infectedPopulation) > 0 do:
  dead <- die(infectedPopulation)
  for all i in infectedPopulatoin do:
    aux <- infect(i, recovered, dead)
```

```
    if notnull(aux) then
      newInfectedPopulation <- aux
    end if
  end for

  currentBestIndividual <- selectBestIndividual(newInfectedPopulation)
  if fitness(curentBestIndividual) > fitness(bestIndividual) then
    bestIndividual <- currentBestIndividual
  end if

  recovered <- infectedPopulation
  clear(infectedPopulation)
  infectedPopulation <- newInfectedPopulation
  time <- time + 1
end while

return bestIndividual
```

Función infect

```
Require: infected as of Individual; recovered, dead as list of Individual

define R1, R2 as real
define newInfected as list of Individual

R1, R2 <- RandomNumber()

if R1 < P_TRAVEL then
  if R2 < P_SUPERSPREADER then
    newInfected <- newInfection(infected, recovered, dead, SPREADER_RATE, ORDINARY_RATE)
  else
    newInfected <- newInfection(infected, recovered, dead, SUPERSPREADER_RATE, ORDINARY_RATE)
  enf if

else
  if R2 < P_SUPERSPREADER then
    newInfected <- newInfection(infected, recovered, dead, SPREADER_RATE, TRAVELER_RATE)
  else
    newInfected <- newInfection(infected, recovered, dead, SUPERSPREADER_RATE, TRAVELER_RATE)
  end if
end if

return newInfected
```

Función newInfection

```
Require: infected as of Individual; recovered, dead as list of Individual

define R3, R4 as real
define newInfected as list of Individual

R3, R4 <- RandomNumber()

aux <- replicate(infected, SPREAD_RATE, TRAVELER_RATE)

for all i in aux do
  if not i in dead do
    if not i in recovered do

      if R4 < P_ISOLATION then
        newInfected <- i
      else
```

```
        recovered <- i
      end if

    else if R3 < P_REINFECTION then
      newInfected <- i
      remove i from recovered
    end if
  end if
end for

return newInfected
```

Función replicate

Esta función no se define en el trabajo original ya que está pensado para trabajar con LSTM. Sin embargo, sí que lo definen en el código que dan como ejemplo, implementado en Java. En él la replicación consiste en cambiar un valor de la solución por otro aleatorio, por tanto en mi implementación he usado el algoritmo de mutación usado en la práctica 2, el cual hace un intercambio de un valor de la solución por otro que no esté en ella. Debido a que es un intercambio similar al que se produce en el *BLM*, he aprovechado para calcular el coste de las nuevas soluciones de manera factorizada para ahorrar tiempo de ejecución. Como el procedimiento de cómo hacerlo ya se explicó en la práctica 1 no lo incluyo en el pseudocódigo.

```
Require: infected as of Individual; spread_rate as Integer, traveler as Boolean

define replicates as list of Individuals
define traver_distance, interA, interB as Integer
define distance_v as Real



if traveler then
  travel_distance <- Random(infected)
else
  travel_distance <- 1



for i in spread_rate do
  aux <- infected

  for j in travel_distance do

    interA <- Random(infected)

    while interB in sel do
      interB <- Random(datos)
    end while

    aux(interA) <- interB
  end for

  replicates <- aux
end for

return replicates
```

2.2. Análisis de resultados experimentales.

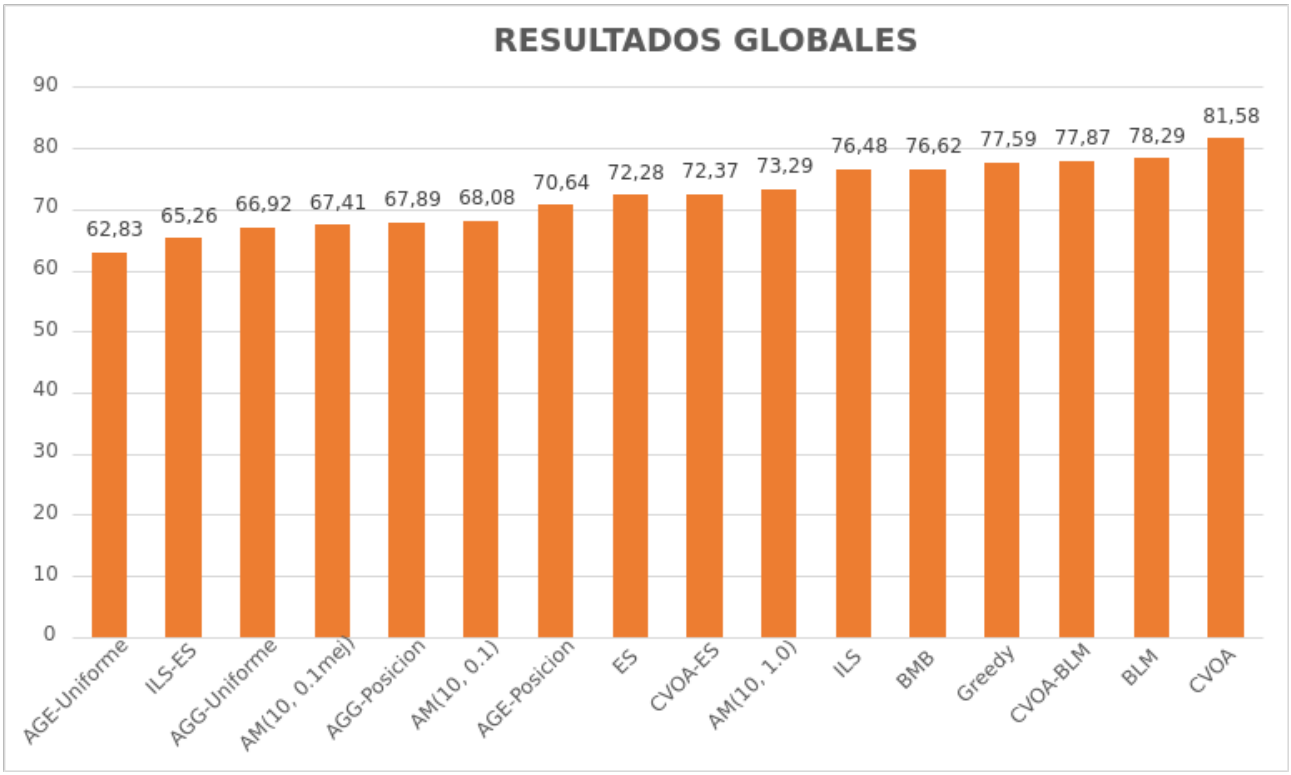
Como en el resto de prácticas de la asignatura, se ha usado el conjunto de datos GKD-b el cual consiste en 50 archivos de datos con diferentes "dificultades" y tamaños para probar los algoritmos. Se hará el análisis sobre la ejecución del algoritmo con estos datos, obteniendo tanto el coste de la solución como el tiempo empleado en su obtención.

Tabla de comparación.

Como he explicado en apartados anteriores, el experimento consiste en probar los distintos algoritmos para los 50 casos GDK-b de la MDPLIB. Para obtener los mejores datos posibles dentro de mis posibilidades el proyecto ha sido ejecutado en mi máquina en un entorno de sólo terminal por lo que eran muy pocos los procesos compitiendo por el uso de CPU (esto sólo afecta a los tiempos de ejecución ya que los resultados vienen dados por la semilla generadora de números aleatorios).

A continuación muestro una tabla con los resultados globales de todos los algoritmos implementados:

RESULTADOS GLOBALES		
Algoritmo	**Desviación**	**Tiempo**
Greedy	77,59	0,9
BLM	78,29	35,97
AGG-Uniforme	66,92	5748,18
AGG-Posicion	67,89	3322,3
AGE-Uniforme	62,83	1632,48
AGE-Posicion	70,64	1097,26
AM(10, 1.0)	73,29	783,26
AM(10, 0.1)	68,08	927,3
AM(10, 0.1mej)	67,41	970
ES	72,28	23,06
BMB	76,62	14,22
ILS	76,48	14,02
ILS-ES	65,26	204,38
CVOA	81,58	15219,4
CVOA-BLM	77,87	10647,5
CVOA-ES	72,37	11597,96



En un primer vistazo a los resultados obtenidos, CVOA obtiene los peores resultados tanto en coste como en tiempo de ejecución. Es el más alejado de los resultados de referencia del conjunto de datos.

Lo primero que llama la atención es el tiempo de ejecución. Esto se debe a que CVOA llega a tener miles de individuos en la lista de infectados y varias decenas de miles de individuos en la lista de recuperados. Por cada individuo de la lista de infectados se invoca el resto de funciones del algoritmo, dentro de las cuales, entre otros procesos, es necesario recorrer las listas de individuos recuperados y muertos. Ya que ni C++ ni Armadillo no proveen de un método rápido para comprobar si un individuo pertenece a una población es necesario hacer un bucle que recorra estas listas para hacer esta comprobación, de tal manera que cuantos más individuos recuperados/muertos haya mayor será el tiempo de ejecución.

Además, CVOA usa muchos valores generados aleatoriamente. La generación de estos valores pseudoaleatorios conlleva un tiempo de ejecución notable, el cual se hace aún más notable al aumentar el número de iteraciones.

Lo segundo es que, aun inviertiendo tanto tiempo de ejecución sus resultados dejan mucho que desear. Los resultados obtenidos son los más alejados de los resultados de referencia del conjunto de datos, con una diferencia notable con el algoritmo de búsqueda local, teniendo más de tres puntos de diferencia en la media de la desviación típica.

Estos resultados me resultan curiosos ya que por cada iteración del algoritmo principal se evalúan miles de individuos, llegando a evaluar varias decenas de miles de individuos distintos. Aún así los resultados obtenidos no consiguen una mejora. Mi teoría es que el método de replicación utilizado no consigue hacer una exploración de una zona muy amplia del espacio de búsqueda y se atasca en un óptimo local. Tras muchos experimentos he comprobado esto último, ya que dependiendo de la semilla de números aleatorios utilizada se pueden obtener o muy buenos o muy malos resultados para un mismo archivo de datos, denotando cómo el

algoritmo no hace una búsqueda muy amplia, es decir, la diversificación de este algoritmo es bastante pobre. Se supone que los infectados viajeros diversifican más la exploración ya que modifican más severamente los individuos (hacen una mutación de más valores de una solución).

Por otro lado, las versiones híbridas de este algoritmo consiguen unos resultados un poco mejores, colándose en mejores puestos de la lista. Comentaré estos resultados en los apartados siguientes.

CVOA *vs* algoritmos principales.

Algoritmo ILS-ES	
Media Desv:	65,26
Media Tiempo(ms):	204,38
Media Coste	217,283

ALGORITMO GREEDY	
Media Desv:	77,59
Media Tiempo(ms):	0,9
Media Coste	387,72593

ALGORITMO BLM	
Media Desv:	78,29
Media Tiempo(ms):	35,972
Media Coste	448,993354

ALGORITMO AGE-U	
Media Desv:	62,83
Media Tiempo(ms):	1632,48
Media Coste	195,895

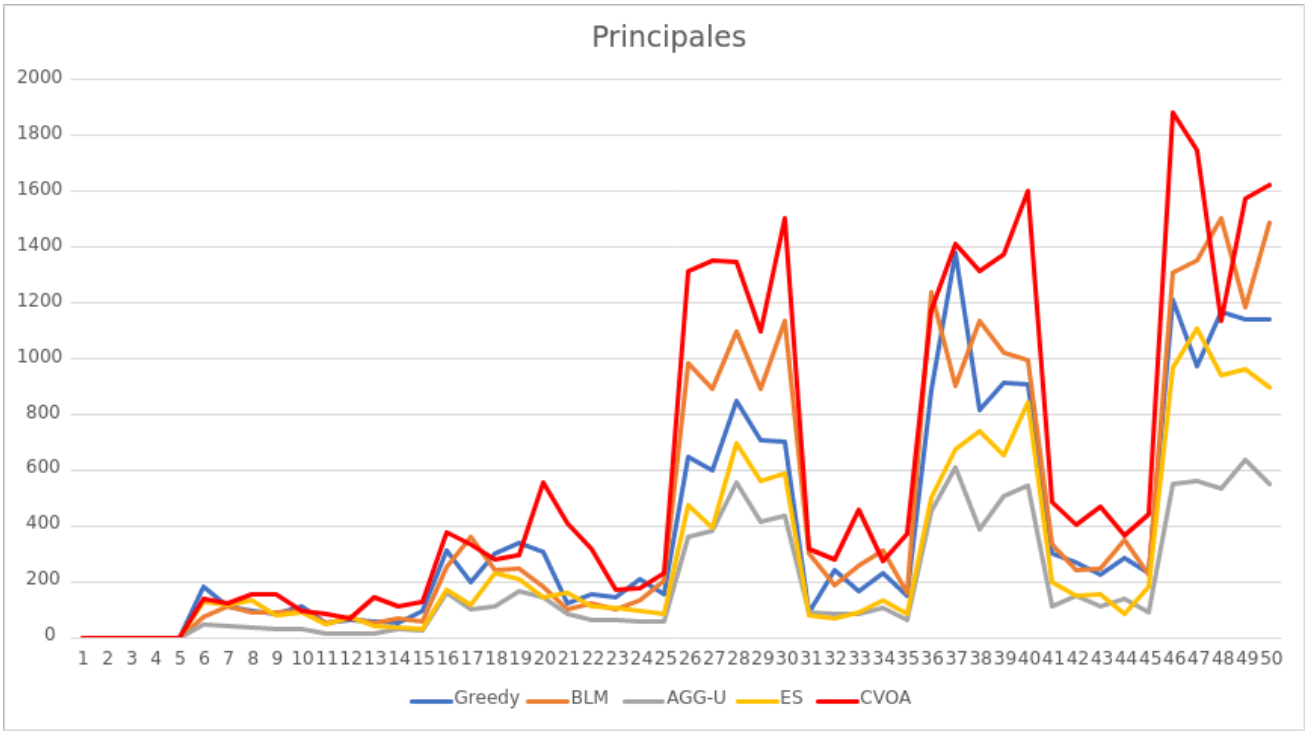
Algoritmo ES	
Media Desv:	72,28
Media Tiempo(ms):	23,06
Media Coste	288,336

Algoritmo CVOA	
Media Desv:	81,58
Media Tiempo(ms):	15219,4
Media Coste	593,450

En este apartado se comparará los resultados obtenidos con el algoritmo CVOA con una selección de algoritmos implementados en las prácticas de la asignatura, que a mi parecer son los más interesantes para esta comparación ya que son todos algoritmos sin hibridación.

Tal como dije en el apartado anterior, el tiempo de ejecución de CVOA es mucho superior que el resto de algoritmos. Por ejemplo, de media es 10 veces más lento que AGE-U, el cual hasta ahora era el más lento de todos los algoritmos implementados.

Para comparar los costes se muestra la siguiente tabla:



Se observa como la tendencia de coste de CVOA es siempre superior que la del resto, sin importar si se tratan de conjuntos de datos más “fáciles” o “difíciles” (el tamaño de m define la dificultad). Únicamente logra superar a BLM y Greedy en ciertos archivos.

Es curioso el comportamiento errático que muestra en ciertas ocasiones. Por ejemplo, en el archivo 48 la tendencia de todos los algoritmos es aumentar el coste obtenido, sin embargo, CVOA obtiene un coste mucho más bajo que en el anterior archivo. Esto se debe a la dependencia de la generación de valores aleatorios ya que puede coincidir que una ejecución con cierta semilla el algoritmo explore un espacio de búsqueda mucho mejor.

Hibridaciones de CVOA *vs* otras hibridaciones.

Algoritmo AM(10, 0.1mej)	
Media Desv:	67,41
Media Tiempo(ms):	970
Media Coste	318,407

Algoritmo ILS	
Media Desv:	76,48
Media Tiempo(ms):	14,02
Media Coste	360,713

Algoritmo ILS-ES	
Media Desv:	65,26
Media Tiempo(ms):	204,38
Media Coste	217,283

Algoritmo CVOA-BLM	
Media Desv:	77,87
Media Tiempo(ms):	10647,5
Media Coste	441,334

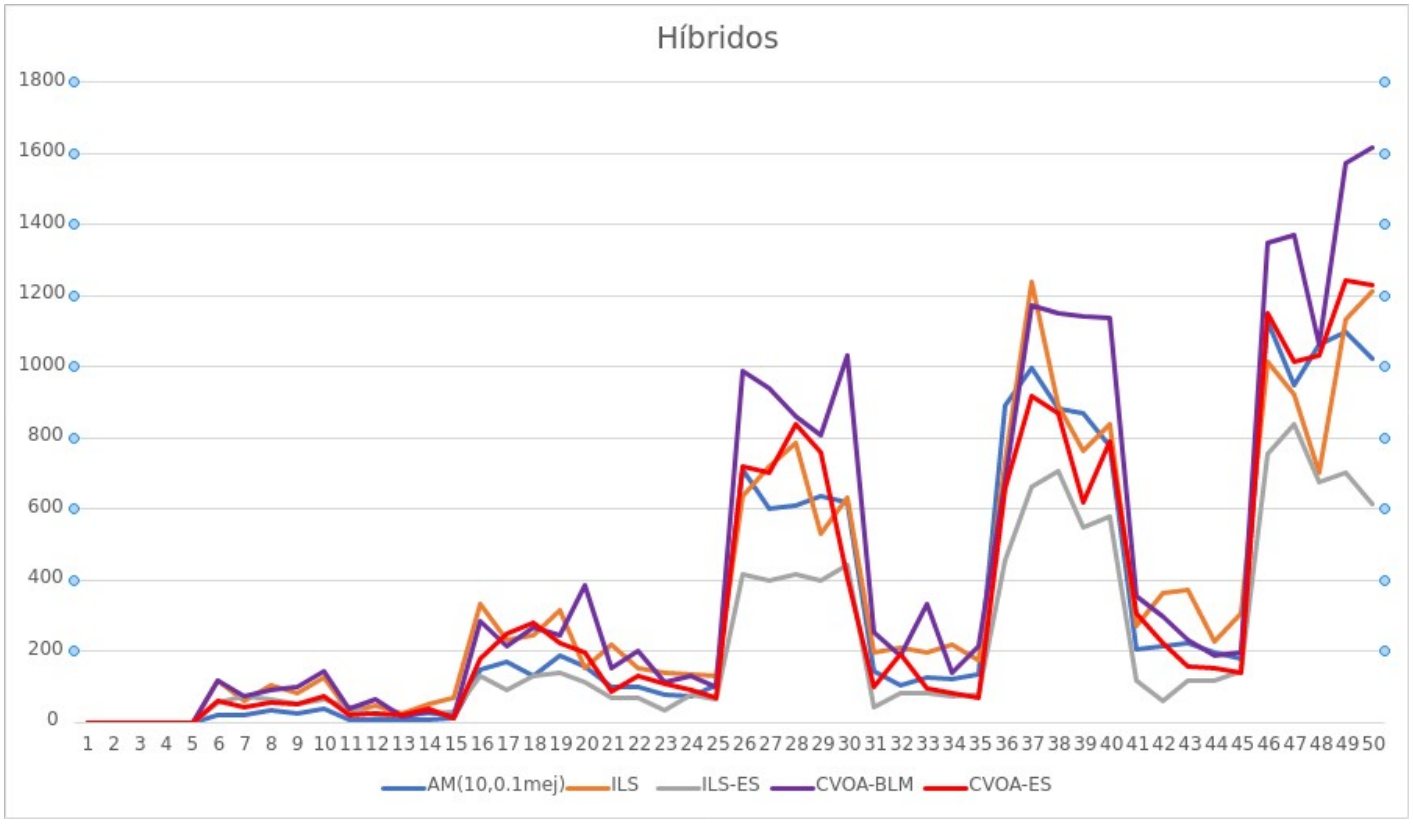
Algoritmo CVOA-ES	
Media Desv:	72,37
Media Tiempo(ms):	11597,96
Media Coste	329,149

En este apartado se compararán las versiones híbridas de CVOA (hibridado tanto con BLM como ES) con otros algoritmos de las prácticas que también usan hibridación, de una manera u otra.

Como dije en el apartado de pseudocódigos, para esta hibridación se ha optado por aplicar la búsqueda local una vez por cada iteración del algoritmo principal al mejor individuo de la lista de nuevos infectados.

En este caso observamos que, aunque los tiempos de ejecución sigan siendo altamente elevados, los costes obtenidos son mucho mejores que sin la hibridación. Tanto que, en el caso de CVOA-ES logran superar a ILS y acercarse a AM(10,0.1mej).

Para hacer una comparación con mayor profundidad se muestra una gráfica con todos los resultados obtenidos:



La hibridación con BLM, aunque mejora notablemente a CVOA, no logra ser suficiente. En la gráfica se ve claramente cómo esta implementación raramente supera al resto en los conjuntos de datos difíciles y sólo supera a ILS para los conjuntos de datos fáciles.

Sin embargo, CVOA-ES consigue unos resultados mucho más aceptables. Para la mayoría de archivos de datos consigue superar a AM(10,0.1mej), el cual no olvidemos que obtiene unos resultados bastante decentes. Desempeña un buen trabajo tanto en los conjuntos fáciles como en los difíciles, es más, para ciertos archivos de datos logra resultados muy parecidos a ILS-ES , aunque nunca mejores.

El uso de hibridación en este algoritmo resulta en datos decentes. Aunque el factor de aleatoriedad sigue ahí la búsqueda local aplicada al mejor individuo de cada nueva población de infectados hace que se encuentren valores fuera de ese óptimo local que encontraba CVOA.

3. Conclusiones.

El algoritmo CVOA me pareció muy interesante al leer el *paper* que contenía esta propuesta. Sin embargo, termino un poco decepcionado con los resultados obtenidos. Esperaba que un algoritmo con tal razonamiento detrás lograse encontrar mejores resultados que el resto de algoritmos.

Si bien es cierto que, como ya he dicho antes, creo que el principal problema de mi implementación yace en la obtención de nuevos individuos en la función *replicate*. En la propuesta original explican cómo obtener estos nuevos individuos por medio de una hibridación con *deep learning* (LTSM). De esta manera logran conseguir una serie de nuevos individuos mucho más interesantes que generan mejores resultados.

También he de decir que quizá este algoritmo no sea el mejor para este tipo de problema. Los creadores lo probaron a partir de un estudio estadístico del consumo eléctrico en España. La naturaleza de este problema es muy distinta al MDD, el cual es un problema lógico NP-Completo.

En definitiva, este algoritmo ha sido muy interesante de implementar y me ha propuesto una serie de retos muy interesantes, sin embargo, el trabajo realizado no se ha visto reflejado en los resultados obtenidos y ha resultado en una pequeña decepción.

Referencias.

1. https://arxiv.org/pdf/2003.13633.pdf