UNIVERSIDAD
DE GRANADA

Metaheurísticas

grado en ingeniería informática

PRÁCTICA 3

Enfriamiento Simulado, Búsqueda Local Reiterada y Evolución Diferencial para el Problema del Aprendizaje de Pesos en Características

Autor
Vladislav Nikolov Vasilev

NIE
X8743846M

E-Mail
vladis890@gmail.com

Grupo de prácticas
MH3 Jueves 17:30-19:30

Rama
Computación y Sistemas Inteligentes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2018-2019

Índice

1. Descripción del problema

El problema que se aborda en esta práctica es el Aprendizaje de Pesos en Características (APC). Es un problema típico de *machine learning* en el cuál se pretende optimizar el rendimiento de un clasificador basado en vecinos más cercanos. Esto se consigue mediante la ponderación de las características de entrada con un vector de pesos W, el cuál utiliza codificación real (cada $w_i \in W$ es un número real), con el objetivo de modificar sus valores a la hora de calcular la distancia. Cada vector W se expresa como $W = \{w_1, w_2, \ldots, w_n\}$, siendo n el número de dimensiones del vector de características, y cumpliéndose además que $\forall w_i \in W,\ w_i \in [0, 1]$.

El clasificador considerado para este problema es el 1-NN (genéricamente, un clasificador k-NN, con k vecinos, siendo en este caso $k = 1$), es decir, aquél que clasifica un elemento según su primer vecino más cercano utilizando alguna medida de distancia (en este caso, utilizando la distancia Euclídea). Cabe destacar que no en todos los casos se usará el clasificador 1-NN ya que se pueden dar casos en los que el vecino más cercano de un elemento sea él mismo. Por ese motivo, en algunas técnicas/algoritmos se usará un 1-NN con el criterio de *leave-one-out*, es decir, que se busca el vecino más cercano pero excluyéndose a él mismo.

El objetivo propuesto es aprender el vector de pesos W mediante una serie de algoritmos, de tal forma que al optimizar el clasificador se mejore tanto la precisión de éste como su complejidad, es decir, que se considere un menor número de características. Estos dos parámetros, a los que llamaremos $tasa\_clas$ y $tasa\_red$, respectivamente, se pueden expresar de la siguiente forma:

$$tasa\_clas = 100 \cdot \frac{n^o\ instancias\ bien\ clasificadas\ en\ T}{n^o\ instancias\ en\ T}$$

$$tasa\_red = 100 \cdot \frac{n^o\ valores\ w_i < 0.2}{n^o\ caracteristicas}$$

siendo T el tamaño del conjunto de datos sobre que el que se evalúa el clasificador.

Por tanto, al combinarlos en una única función a la que llamaremos $F(W)$, la cuál será nuestra función objetivo a optimizar (maximizar), tenemos que:

$$F(W) = \alpha \cdot tasa\_clas(W) + (1 - \alpha) \cdot tasa\_red(W)$$

siendo α la importancia que se le asigna a la tasa de clasificación y a la de reducción, cumpliendo que $\alpha \in [0, 1]$. En este caso, se utiliza un $\alpha = 0.5$ para dar la misma importancia a ambos, con lo cuál se pretende que se reduzcan al máximo el número de características conservando una $tasa\_clas$ alta.

2. Descripción de los algoritmos

2.1. Consideraciones previas

Antes de empezar con la descripción formal de los algoritmos implementados, vamos a describir algunos aspectos comunes, como por ejemplo cómo se representan e inicializan las soluciones en todos los casos, cómo se representa la función objetivo o *fitness* y cómo se evalúan las soluciones. También vamos a comentar brevemente otros aspectos, como por ejemplo cómo mantener las soluciones factibles después de aplicarles alguna mutación o transformación (es decir, que se mantengan en el rango dado), entre otros. Cabe destacar que muchos de los pseudocódigos que aparecen a continuación no se han implementado exactamente igual o no aparecen en el código, ya que o bien son operaciones que se han vectorizado o bien ya hay funciones que hacen eso.

Como se dijo al principio, cada solución es un vector de valores reales W en el que $\forall w_i \in W,\ w_i \in [0, 1]$. En el caso de la **Evolución Diferencial** se tendrá un conjunto vectores que formarán la población, ya que sigue un esquema basado en poblaciones. Esto se volverá a comentar más adelante, cuando se hable con más detenimiento de esta técnica.

Para evitar que las soluciones se salgan de este intervalo, se ha implementado una función que se encarga de normalizar los valores de W en el rango. La función se ha usado, por ejemplo, en la búsqueda local, para hacer que al aplicar el operador de generación de un nuevo vecino la solución siguiese siendo válida. La implementación de esta función es la siguiente:

Algorithm 1 Función que normaliza un vector de pesos W

```
1: function NormalizarW(W)
2:     for each w_i ∈ W do
3:         if w_i < 0 then
4:             w_i ← 0
5:         else if w_1 > 1 then
6:             w_i ← 1
7:     return W
```

Para generar las soluciones iniciale se han seguido dos esquemas. En uno de ellos, el cuál es utilizado por el **Enfriamiento Simulado** y la **ILS**, se genera un único vector de pesos inicial con valores aleatorios dados por una distribución uniforme en el rango $[0, 1]$. En el otro esquema, el cuál es seguido por la **Evolución Diferencial** se lleva a cabo algo muy parecido, solo que en vez de generar una única solución inicial, se generan una serie de M soluciones iniciales, donde M es

el tamaño de la población. Los pseudocódigos de estos dos esquemas se pueden ver a continuación:

Algorithm 2 Inicialización de un vector de pesos W

```
1: function GenerarWAleatorio(N)
2:     W ← vector[N]
3:     for each w_i ∈ W do
4:         w_i ← ValorAleatorioUniformeRango0-1()
5:     return W
```

Algorithm 3 Generación de una población inicial en **Evolución Diferencial**

```
1: function GenerarPoblacionInicial(numCrom, numGenes)
2:     poblacion ← NuevaMatrizVacia(numCrom, numGenes)
3:     for i ← 0 to numCrom − 1 do
4:         for j ← 0 to numGenes − 1 do
5:             poblacion[i][j] ← ValorAleatorioUniformeRango0-1()
6:     return poblacion
```

Vamos a comentar ahora algunos detalles extra. Es importante saber como se calcula la distancia a un vecino, ya que esto juega un factor muy importante a la hora de encontrar cuál es el vecino más cercano a un elemento (o el vecino más cercano por el criterio *leave-one-out*). En la implementación de la práctica se ha utilizado un KDTree, que es una estructura de datos parecida a un árbol binario, solo que de K dimensiones. Por dentro, esta estructura utiliza la distancia Euclídea (distancia en línea recta entre dos elementos) para determinar cuál es el elemento más próximo a otro. No hace falta conocer como se implementa esta estructura de datos, pero sí es importante conocer cómo se realiza el cálculo de la distancia Euclídea. En el siguiente pseudocódigo se puede ver el cálculo:

Algorithm 4 Cálculo de la distancia Euclídea entre dos puntos

```
function DistanciaEuclidea(e_1, e_2)
    distancia ← sqrt(sum_{i=1}^{N} (e_1^i − e_2^i)^2)
    return distancia
```

Para la **Evolución Diferencial** sería interesante intentar mantener la población ordenada por valor *fitness*, ya que eso nos va a simplificar bastante el trabajo para uno de los procesos de mutación. Para hacer esto, se ha creado una función que recibe la lista de valores *fitness* y la poblacion, obtiene los índices que dan el orden de forma ascendente de la lista y con estos índices ordena la población y la lista de *fitness*. Aquí se puede ver como funciona:

Algorithm 5 Función para ordenar la población según su valor *fitness*

```
1: function OrdenarPoblacion(fitness , poblacion)
2:     indicesOrden ← ObtenerIndicesOrdenados(fitness)
3:     fitnessOrdenado ← NuevoVectorVacioMismaCapacidad(fitness)
4:     poblacionOrdenada ← NuevaMatrizVaciaMismaCapacidad(poblacion)
5:     for each indice ∈ indicesOrden do
6:         fitnessOrdenado ← fitness[indice]
7:         poblacionOrdenada ← poblacion[indice]
8:     return fitnessOrdenado, poblacionOrdenada
```

Pasemos a ver ahora la función objetivo, $F(W)$, que es lo que se pretende optimizar. Para evaluar la función objetivo, necesitamos calcular *tasa_clas* y *tasa_red*. Para calcular lo primero, podemos seguir la idea detrás del siguiente pseudocódigo:

Algorithm 6 Cálculo de la tasa de clasificación

```
1: function CalculoTasaClas(etiq, etiqPred, N)
2:     bienClasificados ← 0
3:     for i ← 1 to N do
4:         if etiq_i = etiqPred_i then
5:             bienClasificados ← bienCasificados + 1
6:     tasa_clas ← bienClasificados / N
7:     return tasa_clas
```

Para calcular *tasa_red*, suponiendo que queremos saber el número de características por debajo de 0.2 podemos seguir un esquema como el siguiente:

Algorithm 7 Cálculo de la tasa de reducción (I)

```
1: function CalculoTasaRed(W, N)
2:     caracRed ← 0
3:     for each w_i ∈ W do
4:         if w_i < 0.2 then
5:             caracRed ← caracRed + 1
6:     tasa_red ← caracRed / N
7:     return tasa_red
```

Y finalmente, para poder calcular la función a optimizar (nuestra función *fitness* u objetivo), teniendo en cuenta que usamos un $\alpha = 0.5$ para ponderar las dos tasas, y que anteriormente hemos calculado ambas tasas, podemos seguir el siguiente esquema:

Algorithm 8 Cálculo de la función objetivo o *fitness*

function CalculoFuncionFitness($tasa\_clas, tasa\_red, \alpha$)
    $fitness \leftarrow \alpha \cdot tasa\_clas + (1 - \alpha) \cdot tasa\_red$
    return $fitness$

Para acabar, y antes de pasar a ver la implementación de los algoritmos, veamos otra funcionalidad que se usa en todos los algoritmos, que es la forma en la que se evalúa la función objetivo. Podemos evaluar la función objetivo tanto para una solución como para una población de soluciones. Por tanto, vamos a tener dos funciones que nos permitan evaluar las soluciones: una para una única solución y una para toda una población de individuos (soluciones), la cuál por debajo utilizará la evaluación simple tantas veces como individuos tenga la población. Vamos a ver el funcionamiento de estas funciones comenzando primero por la que evalúa una única solución, y veamos luego la función que evalúa toda una población de soluciones:

Algorithm 9 Función para evaluar un vector de pesos W

function Evaluar(*datos*, *etiquetas*, W)
    $datosPesos \leftarrow$ aplicar $w_i \in W$ sobre los $x_i \in datos$ donde $w_i > 0.2$
    $arbolKD \leftarrow$ KDTree($datosPesos$)
    $vecinos \leftarrow arbolKD$.ObtenerVecinosMasCercanoL1O($datosPesos$)
    $pred \leftarrow etiquetas[vecinos]$
    $tasa\_clas \leftarrow$ CalcularTasaClas(*etiquetas*, *pred*, num. etiquetas)
    $tasa\_red \leftarrow$ CalcularTasaRed(W, num. caracteristicas)
    $fitness \leftarrow$ CalculoFuncionFitness($tasa\_clas$, $tasa\_red$)
    return $fitness$

Algorithm 10 Función para evaluar una población

function EvaluarPoblacion(*datos*, *etiquetas*, *poblacion*)
    $listaFitness \leftarrow$ NuevoVector()
    for each $W \in poblacion$ **do**
        $listaFitness$.Añadir(Evaluar(*datos*, *etiquetas*, W))
    return $listaFitness$

2.2. Algoritmos de comparación

2.2.1. Clasificador 1-NN

El primer algoritmo con el que compararemos es el 1-NN que utiliza todas las características, tal como hacíamos en la práctica anterior.

Este clasificador lo que hace es, dado un conjunto de valores X que pertenecen a una muestra y un elemento e, decir cuál es el $x \in X$ más cercano a e, y por tanto, decir que e pertenece a la misma clase x. Para determinar cuál es el elemento más cercano se puede usar alguna métrica de distancia, como por ejemplo la distancia Euclídea, descrita anteriormente. A la hora de implementarlo, para poder acelerar los cálculos, se puede usar un KDTree, ya que permite realizar una consulta rápida (en los casos más favorables su complejidad temporal es $\mathcal{O}(\log n)$, mientras que en el peor caso es $\mathcal{O}(n)$) utilizando la distancia Euclídea para determinar el vecino más cercano, con la penalización de que tarda un tiempo $\mathcal{O}(n)$ en ser construido. Para poder ver un esquema de su funcionamiento, se ofrece el siguiente pseudocódigo:

Algorithm 11 Clasificador 1-NN

```
1: function KNN(X, y, e)
2:     arbolKD ← KDTree(X)
3:     x ← arbolKD.VecinoMasCercano(e)
4:     etiqueta ← y[x]
5:     return etiqueta
```

2.2.2. Algoritmo greedy *RELIEF*

Otro algoritmo con el que vamos a comparar es el algoritmo para el cálculo de pesos *RELIEF*. Es un algoritmo greedy que, comenzando con un W cuyos pesos valen 0, actualiza W para cada $x_i \in X$, buscando para cada x_i cuál es su aliado más cercano (elemento que tiene la misma etiqueta que x_i con el criterio de *leave-one-out*, ya que él mismo podría ser su vecino más cercano) y su enemigo más cercano (elemento que tiene diferente etiqueta a la que tiene x_i).

A la hora de implementarlo, vamos a utilizar 2 KDTree en cada iteración que se van a construir sobre la marcha. En uno se encontrarán todos los aliados de e y en el otro estarán todos sus enemigos. Esto puede suponer una gran penalización por el tiempo de creación de los árvoles, pero es un tiempo insignificante ya que el algoritmo es muy rápido. Después de construir los árboles, se buscará en el caso del aliado más cercano, por el criterio de *leave-one-out*, cuál es el índice de este aliado. En el caso del enemigo más cercano, como este no puede ser él mismo, se buscará el índice del vecino más cercano en ese árbol. Una vez hecho eso, obtendremos los respectivos aliado y enemigo del conjunto de aliados y enemigos. Una vez teniéndolos, ya se puede actualizar el valor de W.

Cuando se ha terminado de iterar sobre todos los elementos de X, se normaliza W para que esté en el rango $[0, 1]$ eligiendo el $w_i \in W$ que sea más grande. Todos aquellos valores por debajo de 0 se truncan a 0, y el resto se normaliza dividiéndolos entre w_m (el w_i más grande).

Antes de ver su implementación, veamos como se inicializa una solución:

Algorithm 12 Inicialización de un vector de pesos W en *RELIEF*

```
1: function GenerarWRelief(N)
2:     W ← VectorVacioCapacidad(N)
3:     for each w_i ∈ W do
4:         w_i ← 0
5:     return W
```

Una vez dicho esto, veamos cómo sería la implementación:

Algorithm 13 Cálculo de los pesos mediante *RELIEF* (I)

```
1: function RELIEF(X, Y)
2:     N ← ObtenerNumElementos(Y)
3:     numCarac ← ObtenerNumCarac(X)
4:     W ← GenerarWRelief(numCarac)
```

Algorithm 14 Cálculo de los pesos mediante *RELIEF* (II)

```
5:    for i ← 0 to N − 1 do
6:        x, y ← X[i], Y[i]
7:        aliados ← e_a ⊂ X : etiqueta(e_a) = y
8:        enemigos ← e_e ⊂ X : etiqueta(e_e) ≠ y
9:        arbolAliados ← KDTree(aliados)
10:       arbolEnemigos ← KDTree(enemigos)
11:       aliadoCercano ← arbolAliados.ObtenerVecinoMasCercanoL1O(x)
12:       enemigoCercano ← arbolEnemigos.ObtenerVecinoMasCercano(x)
13:       aliado ← aliados[aliadoCercano]
14:       enemigo ← enemigos[enemigoCercano]
15:       W ← W + |x − enemigo| − |x − aliado|
16:   w_m ← max(W)
17:   for each w_i ∈ W do
18:       if w_i < 0 then
19:           w_i ← 0
20:       else
21:           w_i ← w_i / w_m
22:   return W
```

2.2.3. Búsqueda Local

En la primera práctica se implementó una búsqueda local (búsqueda basada en trayectorias), así que vamos a rescatarla para tener un algoritmo más de comparación, ademñas de que los AM la necesitarán luego, aunque ligeramente modificada. Esta búsqueda, como debemos recordar, está basada en el primer mejor (Simple Hill Climbing).

La búsqueda local parte de un vector W con valores aleatorios y busca optimizarlo mediante la exploración de los vecinos. Esta exploración se realiza modificando con un valor aleatorio generado a partir de una distribución normal con $\mu = 0$ y $\sigma = 0.3$ un $w_i \in W$, quedándose con el cambio en caso de mejorar la función objetivo, o descartándolo en otro caso.

Para realizar lo explicado anteriormente, se genera una permutación del conjunto $\{0, 1, \ldots, N-1\}$, donde N es el número de características, y se van escogiendo las características según el orden de la permutación, aplicándoles el cambio anteriormente descrito. Si no se produce una mejora, se descarta el cambio realizado. Si no se ha producido mejora en la función objetivo con la permutación, se escoge una nueva permutación y se repite el proceso, hasta un máximo de $20 \cdot N$ evaluaciones sin éxito seguidas de la función objetivo. Si se produce mejora, se acepta el cambio y se genera una nueva permutación, repitiendo el proceso. Todo esto se realiza hasta que se hayan realizado 15000 evaluaciones de la función objetivo, o hasta que se dé la condición anterior (demasiadas iteraciones sin mejora).

El pseudocódigo de la búsqueda local se puede ver aquí:

Algorithm 15 Cálculo de los pesos mediante la Búsqueda Local (I)

```
1:  function BusquedaLocal(datos, etiquetas)
2:      N ← ObtenerNumCaracteristicas(datos)
3:      W ← GenerarWAleatorio(N)
4:      evaluaciones ← 0
5:      evaluacionesMalas ← 0
6:      fitness ← Evaluar(datos, etiquetas, W)
7:      while evaluaciones < 15000 do
8:          Wactual ← W
9:          ordenCaracteristicas ← Permutacion(0 to N − 1)
10:         for each carac ∈ ordenCaracteristicas do
11:             W[carac] ← W[carac]+ GenerarValorDistribucionNormal(μ, σ)
12:             W ← NormalizarW(W)
13:             evaluaciones ← evaluaciones + 1
14:             nuevoFitness ← Evaluar(X, Y, W)
```

Algorithm 16 Cálculo de los pesos mediante la Búsqueda Local (II)

```
15:         if nuevoFitness > fitness then
16:             fitness ← nuevoFitness
17:             evaluacionesMalas ← 0
18:             break
19:         else
20:             evaluacionesMalas ← evaluacionesMalas + 1
21:             W[carac] ← Wactual[carac]
22:         if evaluaciones > 15000 or evaluacionesMalas > 20 · N then
23:             return W
24:     return W
```

2.3. Algoritmos implementados

2.3.1. Enfriamiento Simulado

La primera técnica que se ha implementado en esta práctica es el **Enfriamiento Simulado**. Esta es una búsqueda basada en trayectorias que se inspira en la termodinámica. Esta técnica, al igual que la Búsqueda Local, explora el entorno de la solución actual con el objetivo de intentar encontrar una solución mejor. Sin embargo, a diferencia de esta última, el **Enfriamiento Simulado** puede seleccionar soluciones peores que la actual con una cierta probabilidad, la cuál tendrá en cuenta a la temperatura. A mayor temperatura, mayor será la posibilidad de aceptar una solución peor, y a menor temperatura, menor probabilidad de hacerlo. Por tanto, es una técnica que permite explorar más al principio, ya que al aceptar soluciones peores se pueden salir por ejemplo de zonas de óptimos locales, mientras que al final permite explotar más, ya que habrá menos probabilidad de aceptar una solución que empeore a la actual.

El esquema de enfriamiento que sigue este algoritmo es el **esquema modificado de Cauchy**, el cuál permitirá realizar enfriamientos de forma proporcional a un parámetro β. Este esquema se puede ver en el siguiente pseudocódigo:

Algorithm 17 Esquema de enfriamiento de Cauchy modificaco

```
1: function EsquemaCauchyModificado(T_K, β)
2:     T_{K+1} ← T_K / (1 + β · T_K)
3:     return T_{K+1}
```

El valor de β depende de la temperatura inicial, de la temperatura final que se quiere alcanzar y de un valor M, que es el número de enfriamientos que se quieren hacer. Esto se puede ver en el siguiente pseudocódigo:

Algorithm 18 Cálculo del valor de β

```
1: function CalcularBeta(T_0, T_f, M)
2:     β ← (T_0 − T_f) / (M · T_0 · T_f)
3:     return β
```

La temperatura inicial se ha especificado que se calcule de la siguiente forma, donde C es el coste de la solución inicial y $\phi \in [0, 1]$ es la probabilidad de aceptar un μ por 1 peor que la inicial. Se ha especificado en este problema que $\phi = 0.3$ y que $\mu = 0.3$.

Algorithm 19 Cálculo de la temperatura inicial

```
1: function CalcularTempInicial(C, μ, φ)
2:     T_0 ← (μ · C) / −ln(φ)
3:     return T_0
```

También es importante saber como se generan los vecinos de una solución. Para ello, hace falta modificar una característica aleatoria del vector de pesos añadiéndole un valor aletorio generado mediante una distribución normal de $\mu = 0$ y $\sigma = 0.3$. Es importante destacar que este vecino tiene que normalizarse después de añadirle el valor aleatorio, ya que se tiene que dar que $\forall w_i \in W$, $w_i \in [0, 1]$. Esto se puede ver en el siguiente pseudocódigo:

Algorithm 20 Operador de generación de vecino

```
1: function OperadorVecindario(W, N)
2:     carac ← ValorEnteroAleatorio(N)
3:     vecino[carac] ← vecino[carac]+ GenerarValorDistribucionNormal(μ, σ)
4:     NormalizarW(vecino)
5:     return vecino
```

Antes de ver finalmente el pesudocódigo del Enfriamiento Simulado, hace falta comentar unos aspectos muy breves sobre él. Se ha especificado que el **número máximo de evaluaciones** sea 15000. Además, se ha modificado el bucle externo para que se hagan M iteraciones, siempre y cuando ha habido algún éxito, que es entrar dentro del bloque condicional, debido a que se ha encontrado una solución mejor o se ha aceptado una peor con una cierat probabilidad dada por un valor aleatorio uniforme en el rango $[0, 1]$. Y otra cosa importante es que, ya que estamos en un problema de maximización, el valor de Δf tiene que ser positivo, y al ser este valor positivo, dentro de la exponencial se tiene que eliminar el signo negativo que viene en la versión original.

Con esto ya dicho, pasemos a ver finalmente el pseudocódigo de esta técnica:

Algorithm 21 Función de cálculo de pesos mediante Enfriamiento Simulado (I)

```
1: function EnfriamientoSimulado(X, y, maxEvaluaciones)
2:     T_f ← 0.001
3:     N ← ObtenerNumCaracteristicas(X)
4:     W ← GenerarWAleatorio(N)
5:     mejorW ← W
6:     maxVecinos ← 10 · N
7:     maxExitos ← Redondear(0.1 · maxVecinos)
8:     M ← Redondear(maxEvaluaciones / maxVecinos)
```

Algorithm 22 Función de cálculo de pesos mediante Enfriamiento Simulado (II)

```
9:    numEvaluaciones ← 0
10:   numIteraciones ← 0
11:   numExitos ← 1
12:   C ← Evaluar(X, y, W)
13:   mejorFitness ← C
14:   fitness ← C
15:   numEvaluacions ← numEvaluaciones + 1
16:   T ← CalcularTempInicial(C, μ, φ)
17:   β ← CalcularBeta(T, T_f, M)
18:   while numIteraciones < M and numExitos > 0 do
19:       numExitos ← 0
20:       numVecinos ← 0
21:       while numVecinos < maxVecinos and numExitos < maxExitos do
22:           W' ← OperadorVecindario(W, N)
23:           fitness' ← Evaluar(X, y, W')
24:           numEvaluaciones ← numEvaluaciones + 1
25:           Δf ← fitness' − fitness
26:           if Δf > 0 or ValorAleatorioUniformeRango0-1() ≤ e^(Δf / T) then
27:               numExitos ← numExitos + 1
28:               W ← W'
29:               fitness ← fitness'
30:               if fitness > mejorFitness then
31:                   mejorW ← W
32:                   mejorFitness ← fitness
33:           numVecinos ← numVecinos + 1
34:       T ← EsquemaCauchyModificado(T, β)
35:       numIteraciones ← numIteraciones + 1
36:   return mejorW
```

2.3.2. ILS: Búsqueda Local Iterativa

La segunda técnica que se ha implementado es la **ILS**, o como se conoce en español, **Búsqueda Local Iterativa**. Esta es una búsqueda basada en trayectorias que, a diferencia de la búsqueda local, permite evitar óptimos locales, reiniciando la búsqueda con una nueva solución inicial. Se parte de una solución inicial y se aplica una búsqueda local sobre ella, y el restultado se guarda como la mejor solución hasta el momento. A continuación se aplica una mutación sobre la mejor solución actual, modificando una parte de los valores de la solución y se vuelve a aplicar una búsqueda local sobre esta modificación. aceptando no esta nueva solución como la mejor actual según algún criterio, y se repite este proceso hasta que se dé un cierto de parada.

En este problema, el operador de mutación que se aplica sobre la soluión modifica un 10 % de las características de forma aleatoria añadiéndoles un valor aleatorio generado por una distribución normal de $\mu = 0$ y $\sigma = 0.4$. Esto se puede ver en el siguiente pseudocódigo:

Algorithm 23 Operador de mutación en ILS

```
1: function MUTACION(W, N)
2:     caracteristicas ← GenerarSecuenciaAleatoriaEnteros(N, 0.1)
3:     for carac ∈ caracteristicas do
4:         W[carac] ← W[carac]+ GenerarValorDistribucionNormal(μ, σ)
5:     W ← NormalizarW(W)
6:     return W
```

En este problema, se piden hacer de nuevo 15000 evaluaciones de la función objetivo. Además, se utiliza la búsqueda local de la sección 2.2.3, modificándola para que acepte un número máximo de iteraciones de 1000 (que es el número de evaluaciones que se pide que haga la búsqueda local) y un vector de pesos de entrada, de forma que no tiene que generarlo ella, aunque sí que debe evaluarlo. Además, se elimina toda la parte que implica parar antes de tiempo por no mejorar. También se hace que devuelva el *fitness* de la solución a la que ha llegado, para así evitar tener que volver a calcularlo posteriormente. A continuación se puede ver un pseudocódigo del ILS:

Algorithm 24 Cálculo de los pesos mediante la ILS (I)

```
1: function ILS(X, y, maxEvaluaciones)
2:     N ← ObtenerNumCaracteristicas(X)
3:     numEvaluaciones ← 0
4:     evalsBL ← 1000
5:     W_0 ← GenerarWAleatorio(N)
```

Algorithm 25 Cálculo de los pesos mediante la ILS (II)

```
6:    mejorW, mejorFitness ← BusquedaLocal(X, y, W_0, evalsBL)
7:    numEvaluaciones ← numEvaluaciones + evalsBL
8:    while numEvaluaciones < maxEvaluaciones do
9:        W' ← mejorW
10:       W' ← Mutacion(W', N)
11:       W', fitness ← BusquedaLocal(X, y, W', evalsBL)
12:       if fitness > mejorFitness then
13:           mejorW ← W'
14:           mejorFitness ← fitness
15:   return mejorW
```

2.3.3. Evolución Diferencial

La tercera y última técnica que se ha pedido implementar es la **Evolución Diferencial**. Esta es una metaheurística basada en poblaciones, al igual que los Algoritmos Genéticos. Sin embargo, esta técnica está especialmente diseñada para problemas con variables reales, ya que las mutaciones y recombinaciones están especialmente pensados para éstos. Estos operadores permiten combinar tanto la capacidad explorativa de los AG como su capacidad explotativa con tal de conseguir unos mejores resultados.

En este problema se utilizan dos mutaciones. Una de ellas es el *rand/1*, donde se escogen 3 padres aleatorios y se combinan entre ellos. El otro es el *current-to-best/1*, donde se escogen dos padres aleatorios, el mejor elemento de la población y el elemento i-ésimo sobre el que se está haciendo la mutación, y se combinan entre ellos. Esto se puede ver en los siguientes pseudocódigos:

Algorithm 26 Esquema de mutación *rand/1*

```
1: function MutacionRand(p1, p2, p3, f gen)
2:     mutacion ← p1[gen] + f(p2 − p3)
3:     return mutacion
```

Algorithm 27 Esquema de mutación *current-to-best/1*

```
1: function MutacionCurrentToBest(mejor, actual, p1, p2, f gen)
2:     mutacion ← mejor[gen] + f(mejor[gen] − actual[gen]) + f(p1[gen] −
   p2[gen])
3:     return mutacion
```

Esta mutación, sin embargo, no se hace siempre, si no que viene por una tasa de cruce (CR). Se genera un número aleatorio y, en caso de obtener un valor más pequeño que CR, de realiza la mutación en el gen. En caso contrario, se copia la información genética del cromosoma sobre el que se están realizando las operaciones.

Una vez que se han generado todos los descendientes, éstos tienen que competir contra sus padres directos. Los descendientes solo pueden entrar en la población si su valor *fitness* es mejor que el de sus padres. En caso contrario, serán ignorados.

Para facilitar un poco el trabajo, la población estará ordeanda según su valor *fitness*, ya que uno de los modelos de mutación utiliza el mejor cromosoma. Por tanto, para tener un acceso más rápido a él, lo más adecuado sería tener toda la población ordenada.

Para este problema se tienen que realizar de nuevo 15000 evaluaciones a la función objetivo. Los valores de CR y f son ambos 0.5, con lo cuál hay bastante probabilidad de cruce.

Con todo esto dicho, vamos a ver un pseudocódigo de la Evolución Diferencial:

Algorithm 28 Evolución Diferencial

```
1: function AGG(X, Y, opMutacion)
2:     numGenes ← ObtenerNumCaracteristicas(datos)
3:     Determinar numPadres en función de opMutacion
4:     numEvaluaciones ← 0
5:     poblacion ← GenerarPoblacionInicial(numCrom, numGenes)
6:     fitness ← EvaluarPoblacion(datos, etiquetas, poblacion)
7:     fitness, poblacion ← OrdenarPoblacion(fitness, poblacion)
8:     numEvaluaciones ← numEvaluaciones + tamPob
9:     while numEvaluaciones < 15000 do
10:        descendientes ← NuevoVectorVacio()
11:        for i ← 1 to tamPob do
12:            indicesPadres ← GenerarIndicesExcluyendoValorI(N, i)
13:            Escoger numPadres de indicesPadres de forma aleatoria
14:            descendiente ← NuevoVectorVacio()
15:            for gen ← 1 to numGenes do
16:                if GenerarValorAleatorioUniforme01() < CR then
17:                    descendiente[gen] ← Aplicar mutación según opMutacion
18:                else
19:                    descendiente[gen] ← poblacion[i][gen]
20:            descendiente ← NormalizarW(descendiente)
21:            descendientes.Insertar(descendiente)
22:        fitnessDesc ← EvaluarPoblacion(X, y, descendientes)
23:        numEvaluaciones ← numEvaluaciones + numCrom
24:        for i ← 1 to tamPob do
25:            if fitnessDesc[i] > fitness[i] then
26:                poblacion[i] ← descendientes[i]
27:                fitness[i] ← fitnessDesc[i]
28:        fitness, poblacion ← OrdenarPoblacion(fitness, poblacion)
29:    W ← poblacion[0]
30:    return W
```

3. Desarrollo de la práctica

La práctica se ha implementado en **Python3** y ha sido probada en la versión 3.7.1. Por tanto, se recomienda encarecidamente utilizar un intérprete de Python3 al ejecutar el código y no uno de la versión 2.X, debio a problemas de compatibilidad con ciertas funciones del lenguaje. Se ha probado el código sobre Linux Mint 19 y al estar basado en Ubuntu 18 no debería haber problemas de compatibilidad con otros sistemas, además de que Python es un lenguaje muy portable. No se ha probado en el entorno **conda**, pero si se consiguen instalar los módulos necesarios, no debería haber problemas.

A la hora de implementar el software, se han utilizado tanto módulos ya incluidos en Python, como el módulo **time** para la medición de tiempos, como módulos científicos y para *machine learning*, como por ejemplo **numpy** y **sklearn**. Este último se ha utlizado para poder dividir los datos para el **5 Fold Cross Validation** y para obtener un clasificador KNN que poder utilizar para poder probar los resultados obtenidos por cada uno de los algoritmos. Para la visualización de datos se ha utilizado **pandas**, ya que permite conseguir una visualización rápida de estos gracias a los DataFrames.

Adicionalmente, la estructura de **KDTree** utilizada ha sido sacada de un módulo externo llamado **pykdtree**[1]. Este módulo está implementado en **Cython** y **C** y también utiliza **OMP**, con lo cuál su rendimiento va a ser muy superior a otras implementaciones como por ejemplo el **cKDTree** de **scipy**[1]. En cuanto a su uso, las funciones y la forma de construirlo son las mismas que las de cKDTree, con lo cuál se puede consultar su documentación[2] para obtener más información sobre su uso.

Siendo ahora más concretos en cuanto a la implementación, se ha creado un módulo que contiene tanto código reutilizado de la práctica anterior (creación de particiones, búsqueda local, función de normalización de los datos, función para generar una solución inicial aleatoria y funciones objetivo y de evaluación de los datos) como código referente a esta práctica (es decir, la implementación del Enfriamiento Simulado, la de la ILS y la de la Evolución Diferencial). Se ha implementado un algoritmo para las dos versiones de la Evolución Diferencial. Además, se ha implementado una función a la que se ha llamado clasificador, la cuál se se encargan de recorrer las particiones creadas, de ejecutar los respectivos algoritmos pasándoles los datos, entrenar luego un clasificador 1-NN con los pesos calculados y predecir las clases, además de recopilar información estadística para mostrarla luego por pantalla.

[1] De hecho, pykdtree está basado en cKDTree y libANN, cogiendo lo mejor de cada implementación y paralelizando el código con OMP para conseguir unos rendimientos muy superiores a ambos, tanto a la hora de crear el árbol como para hacer consultas.

Se han utilizado dos semillas aleatorias las cuáles están fijas en el código: una para dividir los datos, y otra para los algoritmos implementados, que se fija al justo antes de llamar a la función que le pasa los datos al algoritmo que se vaya a ejecutar. Los ficheros ARFF proporcionados se han convertido al formato CSV con un script propio, con el objetivo de facilitar la lectura de los datos. Estos archivos también se proporcionan junto con el código fuente implementado.

4. Manual de usuario

Para poder ejecutar el programa, se necesita un intérprete de **Python3**, como se ha mencionado anteriormente. Además, para poder satisfacer las dependencias se necesita el gestor de paquetes **pip** (preferiblemente **pip3**).

Se recomienda instalar las dependencias, las cuáles vienen en el archivo **requirements.txt**, ya que sin ellas, el programa no podrá funcionar. Se recomienda utilizar el script de bash incluido para realizar la instalación, ya que se encarga de instalarlo en un entorno virtual para no causar problemas de versiones con paquetes que ya se tengan instaladas en el equipo o para no instalar paquetes no deseados. Una vez instalados[2], para poder utilizar el entorno creado se debe ejecutar el siguiente comando:

```
$ source ./env/bin/activate
```

Para desactivar el entorno virtual, simplemente basta con ejecutar:

```
(env) $ deactivate
```

Para ejecutar el programa basta con ejecutar el siguiente comando:

```
$ python3 practica3.py [archivo] [algoritmo]
```

Los argumentos **archivo** y **algoritmo** son obligatorios, y sin ellos el programa lanzará una excepción. En cuanto a sus posibles valores:

- **archivo** puede ser: **colposcopy**, **ionosphere** o **texture**.
- **algoritmo** puede ser:
  - Para Enfriamiento Simulado: **sa**.
  - Para ILS: **ils**.
  - Para Evolución Diferencial: **der** para la mutación *Rand* y **de** para la mutación *current-to-best/1*.

A continuación, para ilustrar mejor lo explicado hasta el momento, se ofrece una captura de un ejemplo de ejecución del programa. En la imagen se puede ver la siguiente información:

[2]Si se produce algún error durante la instalación de los paquetes, puede ser debido a pykdtree, ya que al necesitar un compilador que soporte OMP puede fallar en los sistemas OSX. Para evitar estos problemas, el programa puede utilizar un cKDTree de scipy en caso de que a la hora de importar pykdtree se produzca un error, suponiendo a cambio una penalización en el tiempo de ejecución.

- Se muestra primero el conjunto de datos sobre el que se va a ejecutar y el clasificador que se va a ejecutar.
- Se puede ver una tabla en la que aparecen los datos referentes a cada partición (tasa de clasificación, tasa de reducción, agrupación y tiempo).
- Se muestran valores estadísticos para cada variable (valores máximo, mínimo, medio, mediana y desviación típica).

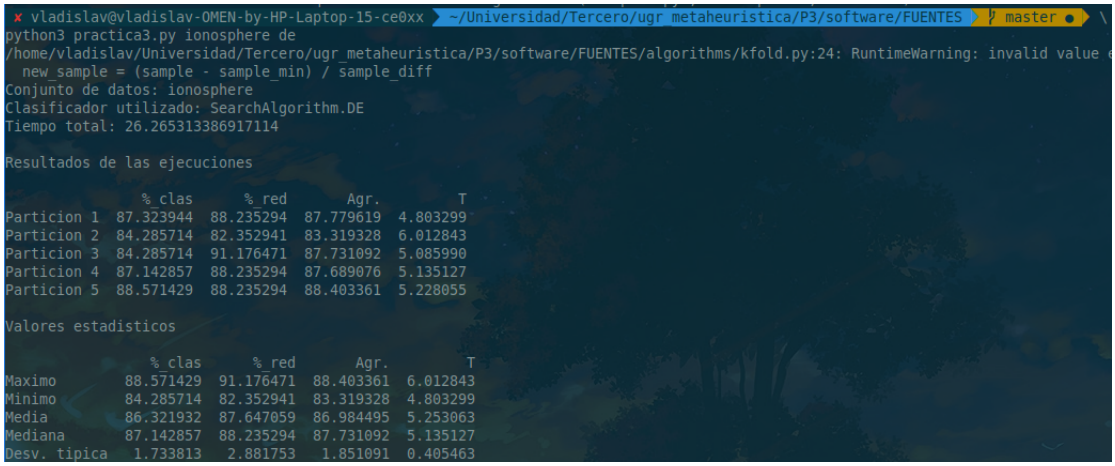

Figura 1: Ejemplo de salida de la ejecución con los datos **ionosphere** y la Evolución Diferencial con *current-to-best/1*.

5. Análisis de resultados y experimentación

5.1. Descripción de los casos del problema

Para analizar el rendimiento de los algoritmos, se han realizado pruebas sobre 3 conjuntos de datos:

- **Colposcopy**: Conjunto de datos de colposcopias adquirido y anotado por médicos profesionales del Hospital Universitario de Caracas. Las imágenes fueron tomadas al azar de las secuencias colposcópicas. 287 ejemplos con 62 características que deben ser clasificados en 2 clases.
- **Ionosphere**: Conjunto de datos de radar que fueron recogidos por un sistema en *Goose Bay*, Labrador. 352 ejemplos con 34 características que deben ser clasificados en 2 clases.
- **Texture**: Conjunto de datos de extracciones de imágenes para distinguir entre 11 texturas diferentes (césped, piel de becerro prensada, papel hecho a mano, rafia en bucle a una pila alta, lienzo de algodón,...). 550 ejemplos con 40 características que deben ser clasificados en 11 clases.

5.2. Análisis de los resultados

Se han realizado distintas ejecuciones con los datos para cada uno de los algoritmos. Cada conjunto de datos se ha dividido con una función de **sklearn**, y se han mezclado con la **semialla aleatoria** 40. Antes de ejecutar cada uno de los algoritmos con la primera partición de datos se ha fijado como **semilla** el valor 8912374 (la misma que la práctica anterior para realizar un análisis más justo, ya que si no podría pasar que alguno de los nuevos algoritmos es mejor). Ambas están fijadas en el código y se pueden comprobar. Es muy importante destacar que todas las gráficas que se vean proceden de la primera partición de cada conjunto de datos, con el objetivo de el análisis sea sobre los mismos datos, en vez de sobre diferentes particiones.

A continuación se muestran los resultados obtenidos para cada algoritmo y para cada uno de los conjuntos de datos, rescatando las tablas de la práctica anterior. Se han medido los valores de tasa de clasificación, tasa de reducción, agrupación (función objetivo) y tiempo que ha tardado (en segundos). Además se ofrece información extra sobre los valores máximo, mínimo, medio, mediana y desviación típica de cada uno de los datos, para poder comparar los datos más fácilmente. Y, como extra, se ha añadido una tabla en la que aparecen los valores medios de cada algoritmo, para facilitar la comparación. Estas tablas se pueden ver a continuación:

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	79,66	0	39,83	0,00049	85,92	0	42,96	0,00045	91,82	0	45,91	0,00056
Partición 2	66,67	0	33,33	0,00037	85,71	0	42,86	0,00037	91,82	0	45,91	0,00048
Partición 3	71,93	0	35,96	0,00035	88,57	0	44,29	0,00036	93,64	0	46,82	0,00044
Partición 4	82,46	0	41,23	0,00035	87,14	0	43,57	0,00039	93,64	0	46,82	0,00046
Partición 5	73,68	0	36,84	0,00035	84,29	0	42,14	0,00039	90,91	0	45,45	0,00049
Media	74,88	0	37,44	0,00038	86,33	0	43,16	0,00039	92,36	0	46,18	0,00049
Máximo	82,46	0	41,23	0,00049	88,57	0	44,29	0,00048	93,64	0	46,82	0,00056
Mínimo	66,67	0	33,33	0,00035	84,29	0	42,14	0,00036	90,91	0	45,45	0,00044
Mediana	73,68	0	36,84	0,00035	85,92	0	42,96	0,00039	91,82	0	45,91	0,00048
Desv. Típica	5,62	0	2,81	0,000056	1,44	0	0,72	0,000033	1,09	0	0,55	0,00004

Cuadro 1: Resultados obtenidos por el algoritmo 1-NN en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	72,88	29,03	50,96	0,04223180	87,32	2,94	45,13	0,04617286	93,64	7,50	50,57	0,09011912
Partición 2	70,18	25,81	47,99	0,03592849	91,43	2,94	47,18	0,04701781	94,55	15,00	54,77	0,08130646
Partición 3	68,42	48,39	58,40	0,03983450	91,43	2,94	47,18	0,03659153	95,45	7,50	51,48	0,08078218
Partición 4	77,19	64,52	70,85	0,03201556	90,00	2,94	46,47	0,03484225	97,27	5,00	51,14	0,08749294
Partición 5	82,46	29,03	55,74	0,03293490	84,29	2,94	43,61	0,03472829	93,64	2,50	48,07	0,09723902
Media	74,23	39,35	56,79	0,03658905	88,89	2,94	45,92	0,03987055	94,91	7,50	51,20	0,08738794
Máximo	82,46	64,52	70,85	0,04223180	91,43	2,94	47,18	0,04701781	97,27	15,00	54,77	0,09723902
Mínimo	68,42	25,81	47,99	0,03201556	84,29	2,94	43,61	0,03472829	93,64	2,50	48,07	0,08078218
Mediana	72,88	29,03	55,74	0,03592849	90,00	2,94	46,47	0,03659153	94,55	7,50	51,14	0,08749294
Desv. Típica	5,07	14,91	7,91	0,00392631	2,75	0,00	1,37	0,00553680	1,36	4,18	2,15	0,00608498

Cuadro 2: Resultados obtenidos por el algoritmo *RELIEF* en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	81,36	83,87	82,61	2,47677159	88,73	85,29	87,01	0,74606109	92,73	77,50	85,11	0,81571221
Partición 2	71,93	82,26	77,09	2,88534594	88,57	79,41	83,99	0,83731246	89,09	85,00	87,05	0,86508775
Partición 3	70,18	82,26	76,22	2,59485793	80,00	94,12	87,06	0,42965746	95,45	82,50	88,98	1,02096820
Partición 4	71,93	82,26	77,09	2,90626860	88,57	91,18	89,87	0,75856829	90,91	85,00	87,95	1,56289935
Partición 5	64,91	75,81	70,36	2,81053543	85,71	91,18	88,45	0,62555575	87,27	82,50	84,89	1,22837043
Media	72,06	81,29	76,68	2,73475590	86,32	88,24	87,28	0,67943101	91,09	82,50	86,80	1,09860759
Máximo	81,36	83,87	82,61	2,90626860	88,73	94,12	89,87	0,83731246	95,45	85,00	88,98	1,56289935
Mínimo	64,91	75,81	70,36	2,47677159	80,00	79,41	83,99	0,42965746	87,27	77,50	84,89	0,81571221
Mediana	71,93	82,26	77,09	2,81053543	88,57	91,18	87,06	0,74606109	90,91	82,50	87,05	1,02096820
Desv. Típica	5,31	2,81	3,89	0,16968432	3,35	5,26	1,95	0,14206914	2,84	2,74	1,59	0,27312796

Cuadro 3: Resultados obtenidos por el algoritmo BL en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	77.97	72.58	75.27	10.26	83.1	85.29	84.2	5.0	90.91	85.0	87.95	6.29
Partición 2	66.67	72.58	69.62	10.79	82.86	88.24	85.55	4.36	86.36	82.5	84.43	6.63
Partición 3	78.95	59.68	69.31	13.95	88.57	85.29	86.93	4.69	93.64	77.5	85.57	6.88
Partición 4	77.19	74.19	75.69	9.55	91.43	91.18	91.3	3.88	89.09	85.0	87.05	6.26
Partición 5	71.93	72.58	72.26	10.97	81.43	76.47	78.95	5.98	89.09	82.5	85.8	6.28
Media	74.54	70.32	72.43	11.1	85.48	85.29	85.39	4.78	89.82	82.5	86.16	6.47
Máximo	78.95	74.19	75.69	13.95	91.43	91.18	91.3	5.98	93.64	85.0	87.95	6.88
Mínimo	66.67	59.68	69.31	9.55	81.43	76.47	78.95	3.88	86.36	77.5	84.43	6.26
Mediana	77.19	72.58	72.26	10.79	83.1	85.29	85.55	4.69	89.09	82.5	85.8	6.29
Desv. Típica	4.63	5.36	2.7	1.51	3.84	4.92	4.01	0.7	2.4	2.74	1.22	0.25

Cuadro 4: Resultados obtenidos por el AGG con cruce BLX-α en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	81.36	58.06	69.71	18.7	87.32	61.76	74.54	9.83	93.64	75.0	84.32	8.72
Partición 2	73.68	51.61	62.65	17.91	85.71	70.59	78.15	9.98	90.91	77.5	84.2	8.04
Partición 3	66.67	54.84	60.75	14.76	90.0	64.71	77.35	9.03	92.73	62.5	77.61	10.16
Partición 4	78.95	54.84	66.89	16.96	91.43	64.71	78.07	8.22	90.91	67.5	79.2	9.85
Partición 5	71.93	48.39	60.16	18.4	81.43	61.76	71.6	10.36	90.91	55.0	72.95	12.62
Media	74.52	53.55	64.03	17.35	87.18	64.71	75.94	9.48	91.82	67.5	79.66	9.88
Máximo	81.36	58.06	69.71	18.7	91.43	70.59	78.15	10.36	93.64	77.5	84.32	12.62
Mínimo	66.67	48.39	60.16	14.76	81.43	61.76	71.6	8.22	90.91	55.0	72.95	8.04
Mediana	73.68	54.84	62.65	17.91	87.32	64.71	77.35	9.83	90.91	67.5	79.2	9.85
Desv. Típica	5.2	3.29	3.69	1.42	3.5	3.22	2.54	0.76	1.15	8.22	4.28	1.57

Cuadro 5: Resultados obtenidos por el AGG con cruce aritmético en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	69.49	72.58	71.04	11.01	88.73	91.18	89.95	5.58	88.18	80.0	84.09	8.03
Partición 2	71.93	70.97	71.45	11.7	85.71	88.24	86.97	6.71	85.45	75.0	80.23	7.94
Partición 3	68.42	64.52	66.47	13.99	81.43	82.35	81.89	8.23	94.55	85.0	89.77	6.83
Partición 4	77.19	80.65	78.92	10.92	80.0	85.29	82.65	5.73	94.55	67.5	81.02	10.15
Partición 5	71.93	70.97	71.45	11.41	84.29	79.41	81.85	6.69	91.82	77.5	84.66	8.06
Media	71.79	71.94	71.86	11.81	84.03	85.29	84.66	6.59	90.91	77.0	83.95	8.2
Máximo	77.19	80.65	78.92	13.99	88.73	91.18	89.95	8.23	94.55	85.0	89.77	10.15
Mínimo	68.42	64.52	66.47	10.92	80.0	79.41	81.85	5.58	85.45	67.5	80.23	6.83
Mediana	71.93	70.97	71.45	11.41	84.29	85.29	82.65	6.69	91.82	77.5	84.09	8.03
Desv. Típica	3.03	5.16	4.0	1.13	3.1	4.16	3.26	0.94	3.59	5.79	3.37	1.07

Cuadro 6: Resultados obtenidos por el AGE con cruce BLX-α en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	74.58	59.68	67.13	15.52	85.92	61.76	73.84	10.09	95.45	72.5	83.98	9.33
Partición 2	68.42	66.13	67.28	13.6	87.14	64.71	75.92	11.17	90.0	75.0	82.5	9.81
Partición 3	68.42	67.74	68.08	14.62	90.0	73.53	81.76	8.93	89.09	70.0	79.55	9.6
Partición 4	77.19	69.35	73.27	12.62	85.71	64.71	75.21	11.07	82.73	70.0	76.36	10.9
Partición 5	70.18	69.35	69.77	11.34	84.29	70.59	77.44	11.03	91.82	72.5	82.16	10.52
Media	71.76	66.45	69.1	13.54	86.61	67.06	76.84	10.46	89.82	72.0	80.91	10.03
Máximo	77.19	69.35	73.27	15.52	90.0	73.53	81.76	11.17	95.45	75.0	83.98	10.9
Mínimo	68.42	59.68	67.13	11.34	84.29	61.76	73.84	8.93	82.73	70.0	76.36	9.33
Mediana	70.18	67.74	68.08	13.6	85.92	64.71	75.92	11.03	90.0	72.5	82.16	9.81
Desv. Típica	3.53	3.59	2.29	1.47	1.92	4.32	2.72	0.86	4.16	1.87	2.68	0.59

Cuadro 7: Resultados obtenidos por el AGE con cruce aritmético en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	69.49	85.48	77.49	7.15	80.28	88.24	84.26	3.91	88.18	85.0	86.59	4.74
Partición 2	68.42	80.65	74.53	8.19	87.14	88.24	87.69	3.57	87.27	85.0	86.14	5.23
Partición 3	80.7	83.87	82.29	7.56	75.71	88.24	81.97	3.82	93.64	82.5	88.07	5.51
Partición 4	84.21	83.87	84.04	7.8	85.71	94.12	89.92	3.43	91.82	85.0	88.41	5.39
Partición 5	78.95	83.87	81.41	7.79	84.29	91.18	87.73	3.68	92.73	82.5	87.61	5.75
Media	76.35	83.55	79.95	7.7	82.63	90.0	86.31	3.68	90.73	84.0	87.36	5.32
Máximo	84.21	85.48	84.04	8.19	87.14	94.12	89.92	3.91	93.64	85.0	88.41	5.75
Mínimo	68.42	80.65	74.53	7.15	75.71	88.24	81.97	3.43	87.27	82.5	86.14	4.74
Mediana	78.95	83.87	81.41	7.79	84.29	88.24	87.69	3.68	91.82	85.0	87.61	5.39
Desv. Típica	6.28	1.58	3.46	0.34	4.15	2.35	2.83	0.17	2.53	1.22	0.87	0.34

Cuadro 8: Resultados obtenidos por el AM con cruce BLX-α y Búsqueda Local sobre toda la población en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	72.88	83.87	78.38	5.88	90.14	91.18	90.66	2.94	87.27	85.0	86.14	4.31
Partición 2	71.93	83.87	77.9	6.73	90.0	91.18	90.59	2.97	90.91	87.5	89.2	4.22
Partición 3	70.18	83.87	77.02	6.62	94.29	82.35	88.32	3.85	95.45	87.5	91.48	4.35
Partición 4	75.44	80.65	78.04	8.95	90.0	82.35	86.18	4.7	85.45	87.5	86.48	4.48
Partición 5	59.65	82.26	70.95	7.66	87.14	91.18	89.16	2.75	81.82	85.0	83.41	4.58
Media	70.01	82.9	76.46	7.17	90.31	87.65	88.98	3.44	88.18	86.5	87.34	4.39
Máximo	75.44	83.87	78.38	8.95	94.29	91.18	90.66	4.7	95.45	87.5	91.48	4.58
Mínimo	59.65	80.65	70.95	5.88	87.14	82.35	86.18	2.75	81.82	85.0	83.41	4.22
Mediana	71.93	83.87	77.9	6.73	90.0	91.18	89.16	2.97	87.27	87.5	86.48	4.35
Desv. Típica	5.45	1.29	2.79	1.06	2.28	4.32	1.66	0.73	4.67	1.22	2.77	0.13

Cuadro 9: Resultados obtenidos por el AM con cruce BLX-α y Búsqueda Local sobre el mejor de la población en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	72.88	83.87	78.38	5.88	90.14	91.18	90.66	2.94	87.27	85.0	86.14	4.31
Partición 2	71.93	83.87	77.9	6.73	90.0	91.18	90.59	2.97	90.91	87.5	89.2	4.22
Partición 3	70.18	83.87	77.02	6.62	94.29	82.35	88.32	3.85	95.45	87.5	91.48	4.35
Partición 4	75.44	80.65	78.04	8.95	90.0	82.35	86.18	4.7	85.45	87.5	86.48	4.48
Partición 5	59.65	82.26	70.95	7.66	87.14	91.18	89.16	2.75	81.82	85.0	83.41	4.58
Media	70.01	82.9	76.46	7.17	90.31	87.65	88.98	3.44	88.18	86.5	87.34	4.39
Máximo	75.44	83.87	78.38	8.95	94.29	91.18	90.66	4.7	95.45	87.5	91.48	4.58
Mínimo	59.65	80.65	70.95	5.88	87.14	82.35	86.18	2.75	81.82	85.0	83.41	4.22
Mediana	71.93	83.87	77.9	6.73	90.0	91.18	89.16	2.97	87.27	87.5	86.48	4.35
Desv. Típica	5.45	1.29	2.79	1.06	2.28	4.32	1.66	0.73	4.67	1.22	2.77	0.13

Cuadro 10: Resultados obtenidos por el AM con cruce BLX-α y Búsqueda Local sobre un cromosoma aleatorio de la población en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	69.49	83.87	76.68	1.27	87.32	88.24	87.78	0.65	91.82	85.00	88.41	0.89
Partición 2	73.68	83.87	78.78	1.32	87.14	91.18	89.16	0.60	90.00	82.50	86.25	0.86
Partición 3	66.67	85.48	76.08	1.22	80.00	88.24	84.12	0.71	90.00	87.50	88.75	0.91
Partición 4	70.18	83.87	77.02	1.40	88.57	91.18	89.87	0.62	88.18	82.50	85.34	0.87
Partición 5	70.18	80.65	75.41	1.46	87.14	88.24	87.69	0.67	87.27	85.00	86.14	0.85
Media	70.04	83.55	76.79	1.34	86.04	89.41	87.72	0.65	89.45	84.50	86.98	0.88
Máximo	73.68	85.48	78.78	1.46	88.57	91.18	89.87	0.71	91.82	87.50	88.75	0.91
Mínimo	66.67	80.65	75.41	1.22	80.00	88.24	84.12	0.60	87.27	82.50	85.34	0.85
Mediana	70.18	83.87	76.68	1.32	87.14	88.24	87.78	0.65	90.00	85.00	86.25	0.87
Desv. Típica	2.24	1.58	1.13	0.08	3.06	1.44	1.98	0.04	1.59	1.87	1.35	0.02

Cuadro 11: Resultados obtenidos por el Enfriamiento Simulado en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	72.88	88.71	80.80	4.74	85.92	91.18	88.55	2.79	90.00	87.50	88.75	4.20
Partición 2	75.44	85.48	80.46	6.00	91.43	88.24	89.83	2.96	85.45	85.00	85.23	4.98
Partición 3	77.19	90.32	83.76	6.17	88.57	88.24	88.40	2.79	92.73	87.50	90.11	4.89
Partición 4	80.70	88.71	84.71	6.22	90.00	88.24	89.12	3.12	93.64	85.00	89.32	5.15
Partición 5	75.44	87.10	81.27	6.52	85.71	94.12	89.92	3.12	85.45	85.00	85.23	4.64
Media	76.33	88.06	82.20	5.93	88.33	90.00	89.16	2.95	89.45	86.00	87.73	4.77
Máximo	80.70	90.32	84.71	6.52	91.43	94.12	89.92	3.12	93.64	87.50	90.11	5.15
Mínimo	72.88	85.48	80.46	4.74	85.71	88.24	88.40	2.79	85.45	85.00	85.23	4.20
Mediana	75.44	88.71	81.27	6.17	88.57	88.24	89.12	2.96	90.00	85.00	88.75	4.89
Desv. Típica	2.58	1.64	1.71	0.62	2.24	2.35	0.63	0.15	3.48	1.22	2.09	0.33

Cuadro 12: Resultados obtenidos por la ILS en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	66.10	93.55	79.83	7.03	85.92	91.18	88.55	4.88	88.18	87.50	87.84	5.97
Partición 2	73.68	91.94	82.81	7.28	85.71	88.24	86.97	4.93	91.82	87.50	89.66	6.05
Partición 3	68.42	93.55	80.98	6.93	78.57	94.12	86.34	4.78	90.91	87.50	89.20	6.17
Partición 4	84.21	93.55	88.88	6.42	85.71	91.18	88.45	5.03	92.73	87.50	90.11	6.23
Partición 5	71.93	93.55	82.74	6.29	85.71	94.12	89.92	4.81	85.45	87.50	86.48	6.92
Media	72.87	93.23	83.05	6.79	84.33	91.76	88.05	4.88	89.82	87.50	88.66	6.27
Máximo	84.21	93.55	88.88	7.28	85.92	94.12	89.92	5.03	92.73	87.50	90.11	6.92
Mínimo	66.10	91.94	79.83	6.29	78.57	88.24	86.34	4.78	85.45	87.50	86.48	5.97
Mediana	71.93	93.55	82.74	6.93	85.71	91.18	88.45	4.88	90.91	87.50	89.20	6.17
Desv. Típica	6.26	0.65	3.12	0.37	2.88	2.20	1.26	0.09	2.66	0.00	1.33	0.34

Cuadro 13: Resultados obtenidos por la Evolución Diferencial con mutación *rand/1* en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
Partición 1	72.88	87.10	79.99	6.19	87.32	88.24	87.78	4.59	82.73	85.00	83.86	6.14
Partición 2	70.18	83.87	77.02	6.61	84.29	82.35	83.32	5.78	91.82	87.50	89.66	5.79
Partición 3	68.42	80.65	74.53	7.77	84.29	91.18	87.73	4.49	91.82	85.00	88.41	6.30
Partición 4	78.95	85.48	82.22	6.49	87.14	88.24	87.69	4.69	83.64	87.50	85.57	6.09
Partición 5	75.44	83.87	79.65	7.02	88.57	88.24	88.40	4.86	87.27	85.00	86.14	6.18
Media	73.17	84.19	78.68	6.82	86.32	87.65	86.98	4.88	87.45	86.00	86.73	6.10
Máximo	78.95	87.10	82.22	7.77	88.57	91.18	88.40	5.78	91.82	87.50	89.66	6.30
Mínimo	68.42	80.65	74.53	6.19	84.29	82.35	83.32	4.49	82.73	85.00	83.86	5.79
Mediana	72.88	83.87	79.65	6.61	87.14	88.24	87.73	4.69	87.27	85.00	86.14	6.14
Desv. Típica	3.75	2.14	2.65	0.55	1.73	2.88	1.85	0.47	3.87	1.22	2.06	0.17

Cuadro 14: Resultados obtenidos por la Evolución Diferencial con mutación *current-to-best/1* en el problema del APC.

	Colposcopy				Ionosphere				Texture			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
1-NN	74.88	0	37.44	0.00039	86.33	0	43.16	0.00039	92.36	0	46.18	0.00049
RELIEF	74.23	39.35	56.79	0.03659	88.89	2.94	45.92	0.03987	94.91	7.50	51.20	0.08739
BL	72.06	81.29	76.68	2.73476	86.32	88.24	87.28	0.67943	91.09	82.50	86.80	1.09861
AGG-BLX	74.54	70.32	72.43	11.1	85.48	85.29	85.39	4.78	89.82	82.5	86.16	6.47
AGG-CA	74.52	53.55	64.03	17.35	87.18	64.71	75.94	9.48	91.82	67.5	79.66	9.88
AGE-BLX	71.79	71.94	71.86	11.81	84.03	85.29	84.66	6.59	90.91	77.0	83.95	8.2
AGE-CA	71.76	66.45	69.1	13.54	86.61	67.06	76.84	10.46	89.82	72.0	80.91	10.03
AM-(10,1.0)	76.35	83.55	79.95	7.7	82.63	90.0	86.31	3.68	90.73	84.0	87.36	5.32
AM-(10,0.1)	73.87	83.23	78.55	6.14	86.03	88.82	87.43	3.22	89.27	86.0	87.64	4.52
AM-(10,0.1mej)	70.01	82.9	76.46	7.17	90.31	87.65	88.98	3.44	88.18	86.5	87.34	4.39
ES	70.04	83.55	76.79	1.34	86.04	89.41	87.72	0.65	89.45	84.50	86.98	0.88
ILS	76.33	88.06	82.20	5.93	88.33	90.00	89.16	2.95	89.45	86.00	87.73	4.77
DE/rand/1	72.87	93.23	83.05	6.79	84.33	91.76	88.05	4.88	89.82	87.50	88.66	6.27
DE/current-to-best/1	73.17	84.19	78.68	6.82	86.32	87.65	86.98	4.88	87.45	86.00	86.73	6.10

Cuadro 15: Resultados medios obtenidos en el problema del APC.

Con las tablas de resultados ya vistas, solo nos queda analizar los resultados obtenidos para ver cómo se comportan las nuevas técnicas respecto a las anteriores. Para realizar ese análisis vamos a ayudarnos de la tabla de resultados medios, ya que es donde se pueden ver todos los resultados en conjunto. También nos vamos a ayudar de gráficas, las cuáles han sido **todas** obtenidas a partir de la **Partición 1** de los distintos conjuntos de datos, de forma que sea fácil compararlos todos entre ellos para ver cómo se comportan. Además, es esperable que los resultados obtenidos en esta partición sean similares a los que se pueden conseguir en las otras, y por tanto, con analizar una de ellas sabríamos lo que sucede de forma aproximada en las otras.

Es importante destacar que también se han incluido las tablas de los resultados obtenidos por los Algoritmos Genéticos y los Algoritmos Meméticos, ya que también queremos comparar con ellos.

Para comparar las distintas técnicas vamos a fijarnos principalmente en la **tasa de clasificación**, la **tasa de reducción** y la **agrupación**, dándole especial

importancia a ésta última. El **tiempo** también es un factor importante. No obstante, dos de los algoritmos están basados en trayectorias, mientras que el otro está basado en poblaciones. Por tanto, intentar comparar los tiempos para decidir cuál de ellos es el mejor no parece la mejor idea, ya que sería una comparación injusta. Además, aunque haya dos técnicas que siguen el esquema de las búsquedas basadas en trayectorias, una va a realizar un mayor número de evaluaciones a la función objetivo que la otra, con lo cuál, como es obvio, terminará antes. Así que, para intentar ser justos, no vamos a considerar el tiempo como un factor decisivo, pero en caso de que exista mucha similaridad entre los resultados obtenidos por dos técnicas, vamos a preferir quedarnos con aquella que tarde menos.

Hay que tener en cuenta que existen técnicas, como por ejemplo el *RELIEF* o el 1-NN que no ofrecen reducción, ya que no tienen esa capacidad. Por tanto, para comparar con ellos, nos centraremos solamente en la tasa de clasificación. A pesar de eso, en caso de que exista similaridad entre los resultados, preferiremos aquella técnica que **obtenga una buena tasa de clasificación reduciendo además el número de características utilizadas**.

Una vez dicho esto, vamos a explicar brevemente la metodología que vamos a seguir para compara los resultados. Vamos a ir para cada conjunto de datos comparando primero los resultados obtenidos por el Enfriamiento Simulado y la ILS, y vamos a ver cuál de ellas ofrece unos mejores resultados. A continuación observaremos cuál de las dos estrategias de mutación de la Evolución Diferencial y nos quedaremos con la mejor. Posteriormente compararemos las mejores técnicas de cada categoría y escogeremos la mejor. Finalmente, compararemos la mejor nueva metaheurística con el resto, y nos quedaremos con la mejor de ellas. Con todo esto comentado, comencemos por el primer conjunto de datos.

El primer conjunto que vamos a analizar es el **Colposcopy**. Podemos ver claramente que el Enfriamiento Simulado ha conseguido una tasa de clasificación muy baja, inferior al resto de técnicas. No obstante, ha conseguido una reducción decente, superior a algunas técnicas como por ejemplo los AG. La agrupación ha salido bastante alta, situándolo más o menos en un punto intermedio entre los mejores resultados de agrupación y los peores. Esto se debe a que la tasa de clasificación ha tirado por lo bajo, haciendo que la agrupación sea un tanto peor. A pesar de todo esto, ha resultado ser la metaheurística más rápida de todas (sin contar al 1-NN y al *RELIEF*), siendo incluso más rápida que la Búsqueda Local. Por tanto, para el poco tiempo que ha tardado, ha demostrado ofrecer unos resultados bastante buenos, lo cuál nos indica que no es una técnica que deba ser despreciada.

Si analizamos por ejemplo como desciende la temperatura en este algorimo y cómo van difiriendo los *fitness* de la mejor solución y de la actual, podemos ver lo siguiente:

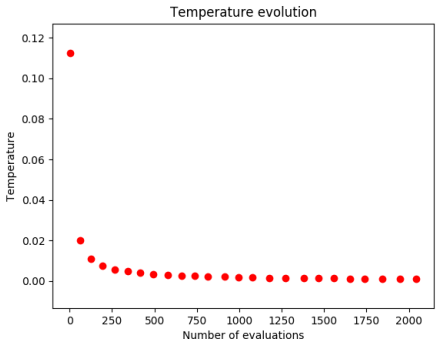

Figura 2: Evolución de la temperatura.

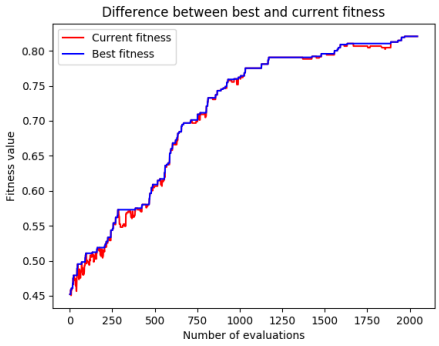

Figura 3: Diferencia entre la mejor solución y la actual.

Vemos como claramente esta técnica realiza poco más de 2000 evaluaciones de la función objetivo, cosa que se queda muy lejos de las 15000 evaluaciones máximas. Esto se debe a que la temperatura desciende bastante rápido debido a que M tiene un valor pequeño (entre 23 y 24 enfriamientos) aún siguiendo el esquema modificado de Cauchy. Por otra parte, vemos que al principio el algoritmo tiene bastante capacidad explorativa, ya que puede irse a soluciones peores. Sin embargo, a mediad que la temperatura va decreciendo, el número de movimiento de empeoramiento que se realizan son mucho menos, y por tanto tiene un comportamiento más de intensificación, ya que solo irá a soluciones mejores.

Observando la ILS, vemos que mejora en todos los aspectos al Enfriamiento Simulado menos en tiempo. Tiene unas tasas de clasificación, reducción y agrupación muy altas, mucho más que otras técnicas vistas anteriormente. El teimpo que tarda también es bastante bueno, aunque es peor que el del Enfriamiento Simulado. Veamos como es su comportamiento:

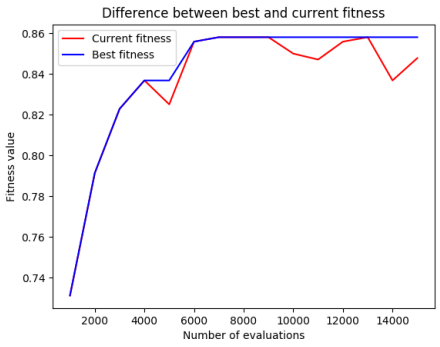

Figura 4: Diferencia entre el mejor *fitness* y el actual.

Podemos ver que, a diferencia del ES, esta técnica realiza las 15000 evaluaciones, lo cuál explica la diferencia de tiempos. Además, como permite reiniciar la búsqueda con otra solución, tiene un componente explorativo bastante fuerte, ya que permite visitar más zonas del espacio de búsqueda. Además, vemos que hay casos en los que el *fitness* actual empeora al mejor, y por tanto, no es escogido para seguir sobre él, si no que simplemente se aplica una mutación sobre el mejor de nuevo y se comienza otra Búsqueda Local. Podemos ver también que llega un punto en el que la mejor solución converge y no mejora, lo cuál nos podría indicar que podríamos haber detenido la búsqueda en ese momento, ya que no ha habido mejora (esto sucede aproximadamente en las 6000 evaluaciones).

Por tanto, de entre ES e ILS, para esta partición escogeríamos la ILS, ya que ha ofrecido los mejores resultados.

Pasando ahora a analizar los distintos modelos de Evolución Diferencial, vemos como el esquema de mutación *rand/1* ha permitido conseguir unos mejores resultados. Excepto en tasa de clasificación, donde gana el *current-to-best/1*, en el resto de aspectos el otro esquema se muestra superior, ya que permite reducir mucho más, y por tanto, nos permite conseguir una agrupación muy alta, la **más alta** de todas las metaheurísticas y técnicas en general. Los tiempos medios de las dos técnicas son muy parecidos, con lo cuál no existe mucha diferencia entre ellas en ese aspecto. Por tanto, de entre estos dos esquemas, nos quedaríamos con el *rand/1*, ya que nos ha ofrecido los mejores resultados. Veamos el motivo por el cuál se puede deber esto:

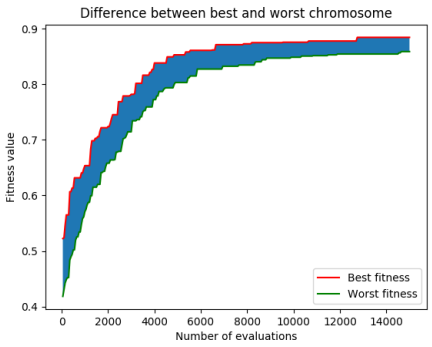

Figura 5: DE con esquema de mutación *rand/1*.

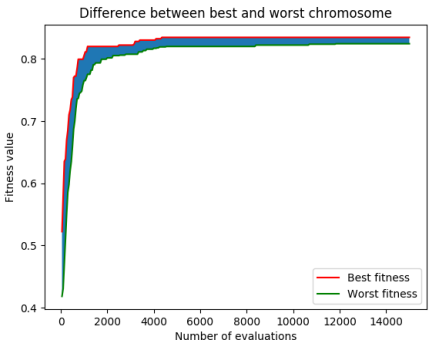

Figura 6: DE con esquema de mutación *current-to-best/1*.

Podemos ver claramente cómo en este problema el esquema *current-to-best/1* converge a la mejor solución muy rápidamente y llega al máximo posible aproximadamente a las 4000 evaluaciones, es decir, después de unas 80 épocas (en cada época se evalúan 50 nuevos cromosomas). Además, el pico del mejor *fitness* es mucho más

bajo que en el esquema *rand/1*. En este segundo observamos como no existe una convergencia tan drástica, ya que se ve que existe cierta diferencia siempre entre el mejor y el peor cromosoma. Además, el máximo al que llega el mejor *fitness* es a un valor cercano al 0.9, mucho mejor que en el otro esquema. Por tanto, podemos concluir de aquí que, al utilizar siempre el mejor, se va a tener una tendencia de que la población converja hacia él, y por tanto se va a perder diversidad. En cambio, en el esquema *rand/1* se va a conservar cierta diversidad, ya que los padres que se usarán en la mutación serán siempre aletorios, y al ser 3 los participantes en el proceso de mutación y recombinación (sin contar al elemento base), existen muchas más posibles combinaciones de valores.

Si tenemos que escoger una metaheurística de las que hemos implementado en esta práctica, deberíamos de escoger sin ningún tipo de duda la DE con el esquema de mutación *rand/1*, ya que es la que mejores resultados ofrece, además de que el tiempo que tarda ésta y el ILS son muy parecidos.

Si ahora comparamos el DE con esquema de mutación *rand/1* con todo el resto de técnicas, vemos que por ejemplo los AM que aplicaban la BL sobre toda la población consiguen una mayor tasa de clasificación que nuestra nueva metaheurística. Sin embargo, en el resto de aspectos la DE se muestra muy superior, ya que tarda un poco menos, reduce más y la agrupación es más alta. Si la comparamos además con la BL, el *RELIEF* y el 1-NN, podemos ver que gana a la BL en todo menos en tiempo, lo cuál es normal, y que para las otras dos técnicas, se ve que es un poco inferior a estas en cuanto a clasificación, pero que a pesar de eso, teniendo la capacidad de reducir, ha conseguido una tasa de aciertos media bastante respetable, reduciendo además una muy gran parte de las características. Por tanto, para este conjunto de datos, nuestro claro ganador sería el **DE con esquema de mutación *rand/1***, ya que nos ha ofrecido unos resultados de clasificación medios bastante buenos y de media ha reducido muchísimo más que cualquiera de las otras técnicas.

Pasemos ahora al conjunto de datos **Ionosphere**. Si analizamos el ES, podemos ver que sucede algo parecido a la última vez, ya que obtiene una tasa de clasificación no muy buena, sin embargo reduce bastante más que muchas de las otras metaheurísticas y la tasa de agrupación es muy buena debido a que reduce mucho, situándolo por tanto entre las técnicas con mayor agrupación. Además, sus tiempos son muy buenos, situándolo también entre los más rápidos. Tal y como hicimos la última vez, vamos a ver cómo es su comportamiento con algunas gráficas:

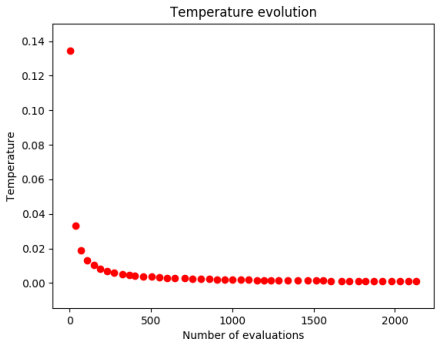

Figura 7: Evolución de la temperatura.

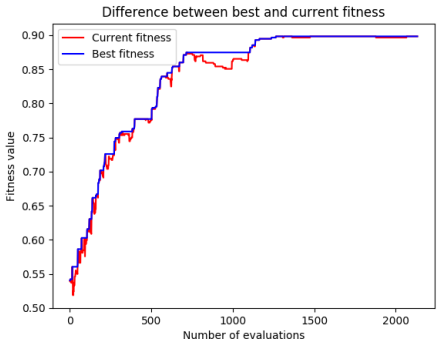

Figura 8: Diferencia entre la mejor solución y la actual.

Podemos ver que, en este caso, se producen muchos más enfriamientos que en el problema anterior, aunque el número de evaluaciones totales que se hacen parece ser muy similar, por no decir casi el mismo. En cuanto a las diferencias entre el *fitness* de la mejor solución y de la actual, podemos ver que, tal y como sucedía en el caso anterior, a medida que va disminuyendo la temperatura, se hace menos probable el aceptar soluciones peores. A partir de las 1000 evaluacoines parece que ya se van aceptando cada vez menos soluciones peores, y solo se pueden esperar soluciones que vayan mejorando a la actual.

Si ahora observamos la ILS, podemos ver que de nuevo sucede algo parecido a antes, ya que es una de las técnicas que consigue una mayor tasa de clasificación (viéndose superada solo por 3 técnicas), una tasa de reducción muy alta que la sitúa entre las mejores y una agrupación muy buena. Además, sus tiempos son también bastante buenos, situándola entre las más rápidas (en este aspecto ha sido superada, como es obvio, por el ES). Vamos a ver su comportamiento con una gráfica:

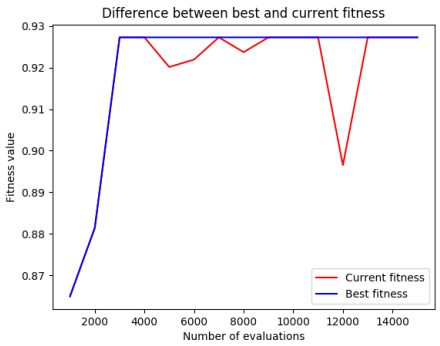

Figura 9: Diferencia entre el mejor *fitness* y el actual.

En este caso, a diferencia del problema anterior, podemos ver que, para esta partición, la BL mejora hasta cierto punto la mejor solución hasta que se estanca aproximadamente a las 3000 evaluaciones. Esto puede significar que, en este caso, la solución ha caído en un óptimo local del cuál por mucho que se mute, no puede salir en el número máximo de evaluaciones establecido. Por tanto, puede ser que aquí sí que hubiese sido buena idea darle algunas evaluaciones más a la ILS para ver si se conseguía salir de esa zona de óptimos locales.

De nuevo, tal y como sucedió la vez anterior, podemos afirmar que la ILS se comporta mejor que el ES por los buenos resultados que nos ha ofrecido la primera.

Pasando ahora a comparar los distintos esquemas de mutación de la DE, podemos ver que de nuevo sucede algo parecido a la última vez. Es decir, el modelo *current-to-best/1* ofrece una mejor tasa de clasificación, aunque se queda corto en cuanto a tasa de reducción y agrupación si lo comparamos con el *rand/1*. De nuevo, sus tiempos son similares, son lo cuál no esto no es un factor decisivo. Por tanto, de nuevo, el mejor esquema entre los dos es el *rand/1*. Tal y como hicimos la última vez, vamos a estudiar como se comportan estas dos técnicas según pasan el número de evaluaciones:

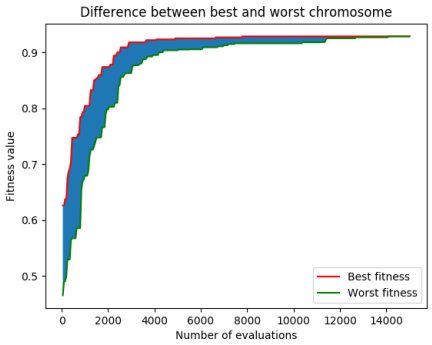

Figura 10: DE con esquema de mutación *rand/1*.

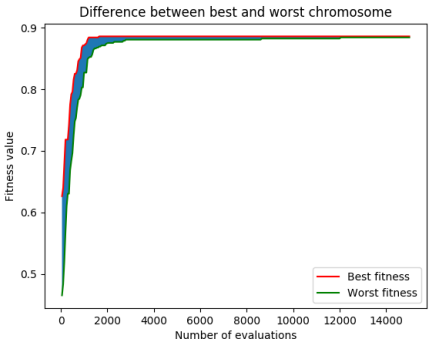

Figura 11: DE con esquema de mutación *current-to-best/1*.

Como podemos ver, en este caso existe una mayor convergencia en los dos casos. Antes, el *rand/1* no convergía tanto, pero ahora a medida que van pasando las evaluaciones, el mejor y el peor *fitness* se van acercando más y más. En el caso del *current-to-best/1* la convergencia es aún más exagerada que antes, ya que se puede ver que a partir de las 40 épocas (unas 2000 evaluaciones) el mejor cromosoma se estanca, y el peor de la población ya ha convergido a él, lo cuál hace que no se produzcan mejoras significativas en el mejor cromosoma de la población. Por tanto, notamos que este segundo enfoque tiene problemas de diversidad muy serios, ya que se pierde muy rápido. El *rand/1*, dependiendo del problema, conserva más o

menos diversidad al final, pero aún así sí que existe cierta diversidad.

Comparando ahora la ILS y el DE con *rand/1*, vemos que la ILS ofrece unos mejores resultados en casi todo menos en tasa de reducción, donde la DE gana por poco. Sin embargo, este no es motivo suficiente para escoger la DE sobre el ILS, así que por tanto, vamos a escoger la ILS, debido a que sus resultados medios son mejores y tarda menos, lo cuál en estos casos es deseable.

Si comparamos ahoa la ILS con el resto de metaheurísticas y técnicas, vemos que solo al *RELIEF* y el AM con BL sobre los 0.1 mejores elementos de la población le superan en cuanto a tasa de clasificación. Sin embargo, en cuanto a reducción solo lo supera el DE con *rand/1* y el AM con BL sobre toda la población se le acerca, pero es el que más agrupa de todos. Además, es uno de los algoritmos más rápidos de media. Por tanto, si de entre todos los modelos tuviésemos que escoger uno, nos quedaríamos con la **ILS**, ya que ha ofrecido unos resultados muy buenos en cuanto a clasificación reduciendo muchas características, muchas más que la mayoría de técnicas, y ha ofrecido una agrupación muy elevada, la más alta de todas, además de unos tiempos muy buenos.

Por último, analicemos los resultados para el conjunto de datos **Texture**. Tal y como lleva sucediendo hasta ahora, el ES ha conseguido una baja tasa de clasificación, consiguiendo sin embargo una mayor reducción que otras metahuerísticas y una tasa de agrupación bastante buena. Además, es una de las metahuerísticas más rápidas, lo cuál es un punto importante a su favor. Si pasamos a analizar sus gráficas, tal y como hicimos anteriormente, podemos ver lo siguiente:

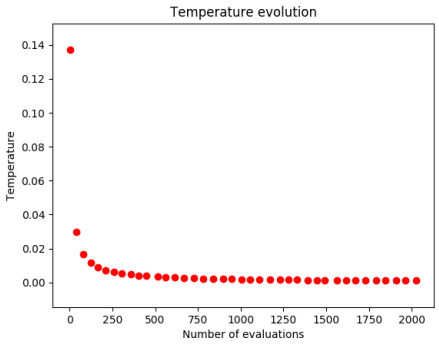

Figura 12: Evolución de la temperatura.

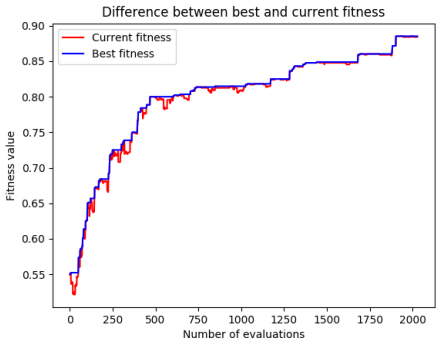

Figura 13: Diferencia entre la mejor solución y la actual.

Como pasó en el caso anterior, hay bastantes más enfriamientos que en el caso del Colposcopy, aunque el número de evaluaciones sigue siendo similar. Tal y como lleva pasando hasta ahora, al principio la metaheurística tiene una mayor capacidad

de exploración, y a medida que va disminuyendo la temperatura, se dejan de aceptar con tanta facilidad soluciones peores. Por tanto, al final es casi un proceso más de intensificación que de exploración.

Si observamos ahora a la ILS, podemos ver que, a diferencia de lo que pasba antes, la tasa de clasificación que ofrece es igual a la del ES, aunque sí que sigue siendo mejor en reducción, donde tiene una mayor reducción media, y en agrupación, donde también, tiene una mayor agrupación media que el ES. Los tiempos, al igual que antes, son mejores en el ES, pero esto no es algo que nos sorprenda a estas alturas, ya que sabemos que la ILS realiza más evaluaciones de la función objetivo. Vamos a ver cómo se ha comportado esta vez la metaheurística:

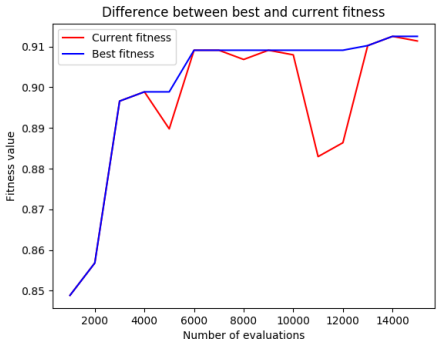

Figura 14: Diferencia entre el mejor *fitness* y el actual.

A diferencia del caso del Ionosphere, aquí podemos ver que no se produce una convergencia prematura de la mejor solución, si no que se va mejorando poco a poco hasta casi el final. Por tanto, podemos afirmar que la solución no había caído en una zona de óptimos locales, ya que en muchas ocasiones se iban aceptando las soluciones que han sido resultado de aplicar la BL sobre la solución mutada.

Tal y como ha sucedido hasta ahora, la ILS se comporta mejor que el ES, lo cuál nos haría decantarnos más por la primera que por la segunda, a menos que quisiéramos tener una solución en muy poco tiempo.

Observando ahora los resultados para la DE, el esquema *rand/1* es superior a *current-to-best/1* en todos los aspectos, es decir, tanto en tasa de clasificación, reducción y agrupación. Donde sí que ganaría el *current-to-best/1* sería en tiempo, aunque tampoco es que sea una diferencia demasiado signifiativa. Por tanto, para sorpresa de nadie a estas alturas, escogeremos la DE con *rand/1* al ser la que mejor ha funcionado. Tal y como hemos hecho hasta ahora, vamos a estudiar las diferencias entre el mejor y el pero cromosoma, para ver si existe una convergencia muy fuerte en cualquiera de las dos técnicas:

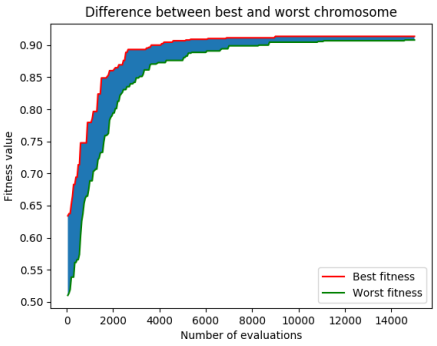

Figura 15: DE con esquema de mutación *rand/1*.

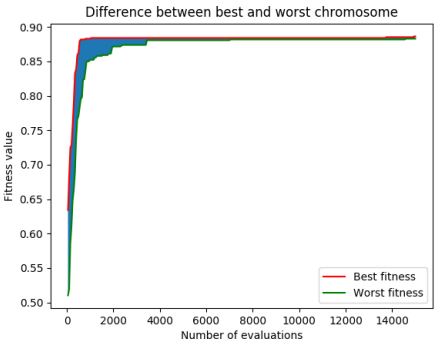

Figura 16: DE con esquema de mutación *current-to-best/1*.

Tal y como lleva pasando hasta ahora, existe una convergencia muy fuerte en el *current-to-best/1*, y esta vez es incluso más destacable, ya que sucede muy pronto. Además, el mejor cromosoma se estanca y no mejora casi nada desde el principio debido a una falta de diversidad. En cambio, *rand/1* sigue manteniendo cierta diversidad hasta casi el final, donde se va reduciendo. Además, en este caso, el mejor cromosoma si que va mejorando más, aunque sea poco a poco.

Comparando la ILS y el DE con *rand/1*, nos quedaríamos con el segundo, ya que, menos en cuanto a tiempo, en los demás aspectos ofrece unos mejores resultados que la ILS.

Si ahora comparamos esta versión del DE con el resto de técnicas, vemos que en agrupación y reducción no le supera nadie, pero en cuanto a tasa de clasificación media se ve superado por los AG, las técnicas básicas y el AM con BL sobre toda la población. A pesar de que en clasificación no haya conseguido unos resultados espectaculares, hay que reconocerle que no son nada malos teniendo en cuenta que ha reducido muchísimo más que cualquier otra técnica, y que por tanto, su agrupación sea muy superior a la del resto. No obstante, si solo nos fijamos en la clasificación, parece que para este conjunto no existe ninguna técnica que pueda ganarle al *RELIEF*, ya que es el que clasifica mejor de media. No obstante, podría darse el caso de que con más ejemplos de validación los resultados de *RELIEF* sean peores.

Aunque por los resultados en la tasa de clasificación parezca una técnica mala, no nos debemos dejar engañar, ya que los resultados siguen siendo buenos teniendo en cuenta que es la metaheurística que más ha conseguido reducir de todas, además de que es la que ha conseguido una mayor agrupación. Por tanto, podemos afirmar que la mejor metahuerística es el **DE con esquema de mutación *rand/1*** por todo lo que hemos dicho anteriormente.

5.3. Experimentación realizada

Después de todo lo que hemos comentado anteriormente, hemos visto que la DE con *rand/1* ha ofrecido, en general, los mejores resultados. También hemos visto que la DE con *current-to-best/1* no ha estado a la altura y ha ofrecido unos peores resultados. Por tanto, en esta sección pretendemos mejorar esta mutación para intentar conseguir unos mejores resultados.

Para ello, hemos modificado un poco la mutación *current-to-best/1*, permitiendo que, en vez de utilizar siempre el mejor, se escoja uno de los 5 mejores de forma aleatoria. La recombinación sigue siendo la misma; lo único que ha cambiado es que ahora en vez de utilizar siempre el mejor de la población y de forzar por tando la convergencia a éste, se utilizan los 5 mejores. Hemos llamado a esta mutación ***current-to-best5-rand/1***.

A continuación se ofrece una tabla con los resultados obtenidos para cada partición, además de la tabla de resultados medios:

	Colposcopy				**Ionosphere**				**Texture**			
	%_clas	***%red***	***Agr.***	**T**	***%_clas***	***%red***	***Agr.***	**T**	***%_clas***	***%red***	***Agr.***	**T**
Partición 1	74.58	91.94	83.26	6.04	85.92	91.18	88.55	4.82	88.18	87.50	87.84	5.92
Partición 2	68.42	91.94	80.18	5.61	87.14	91.18	89.16	4.73	92.73	87.50	90.11	6.07
Partición 3	59.65	93.55	76.60	5.76	77.14	94.12	85.63	4.51	95.45	87.50	91.48	5.94
Partición 4	73.68	93.55	83.62	5.70	85.71	91.18	88.45	4.60	91.82	87.50	89.66	6.12
Partición 5	78.95	90.32	84.63	5.73	88.57	91.18	89.87	4.81	86.36	90.00	88.18	5.89
Media	71.06	92.26	81.66	5.77	84.90	91.76	88.33	4.69	90.91	88.00	89.45	5.99
Máximo	78.95	93.55	84.63	6.04	88.57	94.12	89.87	4.82	95.45	90.00	91.48	6.12
Mínimo	59.65	90.32	76.60	5.61	77.14	91.18	85.63	4.51	86.36	87.50	87.84	5.89
Mediana	73.68	91.94	83.26	5.73	85.92	91.18	88.55	4.73	91.82	87.50	89.66	5.94
Desv. Típica	6.61	1.21	2.93	0.15	4.01	1.18	1.44	0.12	3.25	1.00	1.33	0.09

Cuadro 16: Resultados obtenidos por la Evolución Diferencial con mutación *current-to-best5-rand/1* en el problema del APC.

	Colposcopy				**Ionosphere**				**Texture**			
	%_clas	***%red***	***Agr.***	**T**	***%_clas***	***%red***	***Agr.***	**T**	***%_clas***	***%red***	***Agr.***	**T**
1-NN	74.88	0	37.44	0.00039	86.33	0	43.16	0.00039	92.36	0	46.18	0.00049
RELIEF	74.23	39.35	56.79	0.03659	88.89	2.94	45.92	0.03987	94.91	7.50	51.20	0.08739
BL	72.06	81.29	76.68	2.73476	86.32	88.24	87.28	0.67943	91.09	82.50	86.80	1.09861
AGG-BLX	74.54	70.32	72.43	11.1	85.48	85.29	85.39	4.78	89.82	82.5	86.16	6.47
AGG-CA	74.52	53.55	64.03	17.35	87.18	64.71	75.94	9.48	91.82	67.5	79.66	9.88
AGE-BLX	71.79	71.94	71.86	11.81	84.03	85.29	84.66	6.59	90.91	77.0	83.95	8.2
AGE-CA	71.76	66.45	69.1	13.54	86.61	67.06	76.84	10.46	89.82	72.0	80.91	10.03
AM-(10,1.0)	76.35	83.55	79.95	7.7	82.63	90.0	86.31	3.68	90.73	84.0	87.36	5.32
AM-(10,0.1)	73.87	83.23	78.55	6.14	86.03	88.82	87.43	3.22	89.27	86.0	87.64	4.52
AM-(10,0.1mej)	70.01	82.9	76.46	7.17	90.31	87.65	88.98	3.44	88.18	86.5	87.34	4.39
ES	70.04	83.55	76.79	1.34	86.04	89.41	87.72	0.65	89.45	84.50	86.98	0.88
ILS	76.33	88.06	82.20	5.93	88.33	90.00	89.16	2.95	89.45	86.00	87.73	4.77
DE/rand/1	72.87	93.23	83.05	6.79	84.33	91.76	88.05	4.88	89.82	87.50	88.66	6.27
DE/current-to-best/1	73.17	84.19	78.68	6.82	86.32	87.65	86.98	4.88	87.45	86.00	86.73	6.10
DE/current-to-best5-rand/1	71.06	92.26	81.66	5.77	84.90	91.76	88.33	4.69	90.91	88.00	89.45	5.99

Cuadro 17: Resultados medios obtenidos en el problema del APC.

Vamos ahora a hacer un pequeño análisis de esta mejora respecto a su versión original y ver cómo de buenos son sus resultados respecto a las otras metaheurísticas.

En **Colposcopy** consigue una tasa de clasificación peor que la nos ofrece *current-to-best/1*. No obstante, permite reducir más que la versión original, además de que consigue una tasa de agrupación muy alta, viéndose solo superado por la ILS y la DE con *rand/1*. En comparación con la mejor técnica, DE con *rand/1*, se queda bastante cerca de ésta, pero se ve superada en todas las tasas. Donde gana es en tiempo, donde es la más rápida de las variantes de la DE.

En **Ionosphere** sucede algu parecido que en el caso contrario. Clasifica peor que *current-to-best/1* aunque lo hace mejor que *rand/1*. Sin embargo, tiene la mayor reducción (al igual que el *rand/1* y una agrupación solo superada por la ILS y el AM con BL sobre los 0.1 mejores individuos. Por tanto, aunque sea la mejor versión de la Evolución Diferencial, sigue siendo superado por poco por la ILS, aunque ha ofrecido unos resultados muy buenos.

En el conjunto **Texture** se puede ver que tiene una tasa de clasificación más alta que todas las nuevas metahuerísticas y más que la mayoría de las anteriores, siendo superado solo por la BL, el 1-NN, el *RELIEF* y el AGG con cruce aritmético. Sin embargo, es la metaheurística con mayor tasa de reducción media y la agrupación más alta, haciendo por tanto que sea una opción a considerar en este problema, ya que es el que consigue una mayor agrupación. Además, los tiempos no son los peores, ya que hay otras muchas metaheurísticas basadas en poblaciones que son más lentas que esta.

Vamos a ver ahora mediante unas gráficas si existe o no convergencia muy fuerte:

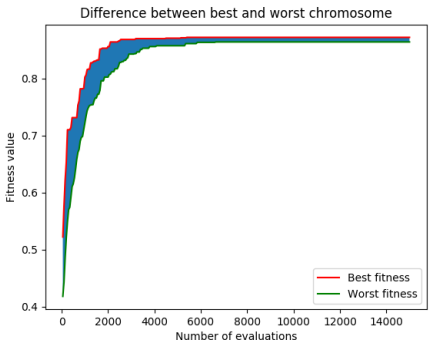

Figura 17: DE con esquema de mutación *current-to-best5-rand/1* en Colposcopy.

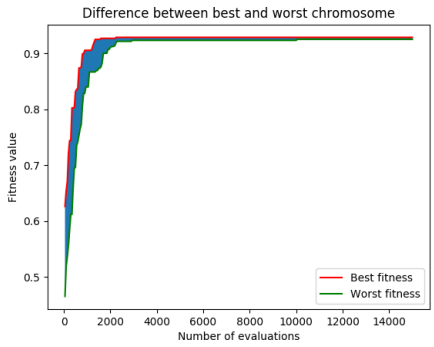

Figura 18: DE con esquema de mutación *current-to-best5-rand/1* en Ionosphere.

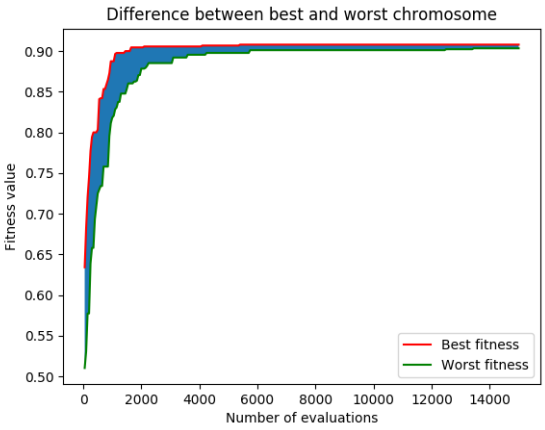

Figura 19: DE con esquema de mutación *current-to-best5-rand/1* en Texture.

Podemos ver que este enfoque ha mejorado un poco los problemas de convergencia que tenía la mutación *current-to-best/1*. Esta vez, se puede ver como la diversidad se mantiene durante un mayor tiempo, aunque, al igual que todas las técnicas basadas en población básicas, tiene problemas de convergencia en las últimas épocas, ya que se pierde diversidad. La diversidad que proporciona este esquema de mutación, sin embargo, no es tan buena como la del *rand/1*, el cuál tenía una diversidad mayor y la mantenía durante un poco más de tiempo. A pesar de eso, sí que se puede apreciar cierta mejora respecto a la versión original.

Para concluir este breve apartado, hemos visto que la mejora propuesta se ha acerado a los mejores algoritmos de cada conjunto de datos, llegando incluso a ser la mejor opción en el caso del conjunto **Texture**. Además, hemos visto que este esquema, al utilizar no solo al mejor, hace que la diversidad se vea aumentada un poco. A lo mejor, se podría intentar un enfoque que mezcle más la aleatoriedad del *rand/1* teniendo en cuenta cuáles son los mejores individuos de la población, con tal de conseguir una mayor diversidad, y por tanto, que no exista una convergencia tan fuerte. Pero, de momento, podemos decir que este expermiento ha ofrecido unos buenos resultados, aunque aún puede ser mejorado.

5.4. Conclusiones

Para concluir este análisis y el trabajo, vamos a ofrecer unas breves conclusiones. Hemos probado distintas técnicas que seguían distintas filosofías para ver cuáles de ellas eran capaces de ofrecernos los mejores resultados. Hemos visto que, ya que estamos en un problema con variables reales, la Evolución Diferencial con el esquema de mutación *rand/1* ha resultado ser, en la mayoría de casos, el mejor

enfoque a seguir, ya que nos ha permitido obtener unos muy buenos resultados. La ILS no debe ser despreciada, ya que sus resultados también han sido muy buenos y nos ha permitido llegar a una muy buena solución en uno de los conjuntos de datos. El Enfriamiento Simulado, a pesar de no haber conseguido los mejores resultados, ha demostrado que es una metaheurística muy eficiente, ya que ha ofrecido unos resultados relativamente buenos con muy poco tiempo de cómputo. También hemos probado una mejora sobre el *current-to-best/1* a la que hemos llamado *current-to-best5-rand/1*, la cuál, en general, nos ha permitido obener unos resultados muy buenos, y ha supuesto una mejora sobre la versión original, ya que funcionaba mejor que ésta en todos los casos.

Sin embargo, es muy importante recalcar, como llevamos haciendo durante todas las prácticas, que no existe una metaheurística ideal que nos resuelva el problema. Cuanta más información tengamos sobre el problema, mejor podremos elegir de entre las distintas metaheurísticas que existen y mejor podremos adaptarla al problema. Además, es muy importante no solo probar una técnica si no un grupo de ellas, y de entre éstas, escoger aquella que creamos que sea más adecuada para el problema a resolver. Como cada problema es un mundo, para cada uno habrá una metaheurística que funcione mejor que las otras. Así que, no hay que tener miedo a probar, ya que solo así podremos saber cuál es la más adecuada para el problema.

Referencias

[1] Repositroio de GitHub de pykdtree.
`https://github.com/storpipfugl/pykdtree`

[2] Documentación de cKDTree.
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html`