Universidad de Granada

Ingeniería Informática

Computación y Sistemas Inteligentes

Técnicas basadas en Trayectorias y Evolución Diferencial.

Autor: José Antonio Ruiz Millán
email: jantonioruiz@correo.ugr.es
Curso: 2017-2018
Problema: 1.b Aprendizaje de Pesos en Características (APC)
Algoritmos: Enfriamiento Simulado, Búsqueda Local Reiterada y Evolución Diferencial
Grupo: A3
Asignatura: Metaheurísticas
20 de julio de 2018

Índice

1. Descripción del problema **2**

2. Descripción de los algoritmos **3**
2.1. Componentes comunes . . . 3
2.2. Enfriamiento Simulado (ES) . . . 3
2.3. Búsqueda Local Reiterada (ILS) . . . 4
2.4. Evolución Diferencial (DE) . . . 5

3. Pseudocodigo algoritmos **6**
3.1. Búsqueda Local (BL) . . . 6
3.2. Algoritmo Enfriamiento Simulado (ES) . . . 7
3.3. Búsqueda Local Reiterada (ILS) . . . 10
3.4. Evolución Diferencial (DE) . . . 11

4. Descripción para ejecución **15**

5. Análisis de los resultados **16**
5.1. Descripción . . . 16
5.2. Resultados obtenidos . . . 16
5.2.1. 1-NN . . . 16
5.2.2. Greedy . . . 17
5.2.3. Enframiento Simulado (ES) . . . 17
5.2.4. Búsqueda Local Reiterada (ILS) . . . 18
5.2.5. Evolución Diferencial (*DE/Rand/1*) . . . 18
5.2.6. Evolución Diferencial (*DE/current-to-best/1*) . . . 18
5.2.7. Tabla resumen . . . 19
5.3. Análisis de resultados . . . 19

1. Descripción del problema

El problema del APC consiste en optimizar el rendimiento de un clasificador basado en vecinos más cercanos a partir de la inclusión de pesos asociados a las características del problema que modifican su valor en el momento de calcular las distancias entre ejemplos. En nuestro caso, el clasificador considerado será el 1-NN (k-NN, k vecinos más cercanos, con k=1 vecino). La variante del problema del APC que afrontaremos busca optimizar tanto la precisión como la complejidad del clasificador. Así, se puede formular como:

Maximizar $F(W) = \alpha \cdot \text{tasa\_clas(W)} + (1-\alpha) \cdot \text{tasa\_red(W)}$

con:

- $\text{tasa\_clas} = 100 \cdot \frac{n^{\text{o}} instancias \quad bien \quad clasificadas \quad en \quad T}{n^{\text{o}} instancias \quad en \quad T}$
- $\text{tasa\_red} = 100 \cdot \frac{n^{\text{o}} valores \quad w_i < 0{,}2}{n^{\text{o}} caracteristicas}$

sujeto a que :

- $w_i = [0, 1], 1 \leq i \leq n$

donde:

- $W = (w_1, ..., w_n)$ es una solución al problema que consiste en un vector de números reales $w_1 \in [0, 1]$ de tamaño n que define el peso que pondera o filtra a cada una de las características f_i.
- 1-NN es el clasificador k-NN con $k = 1$ vecino generado a partir del conjunto de datos inicial utilizando los pesos en W que se asocian a las n características.
- T es el conjunto de datos sobre el que se evalúa el clasificador, ya sea el conjunto de entrenamiento (usando la técnica de validación leave-one-out) o el de prueba.
- $\alpha \in [0, 1]$ pondera la importancia entre el acierto y la reducción de la solución encontrada.

Para la distribución de los datos de entrada utilizaremos la técnica de validación cruzada ***5-fold cross validation*** de la siguiente manera:

- El conjunto de datos se divide en 5 particiones disjuntas al 20 %, con la distribución de clases equilibrada.
- Aprenderemos un clasificador utilizando el 80 % de los datos disponibles (4 particiones de las 5) y validaremos con el 20 % restante (la partición restante) $\rightarrow$ 5 particiones posibles al 80-20 %
- Así obtendremos un total de 5 valores de porcentaje de clasificación en el conjunto de prueba, uno para cada partición empleada como conjunto de validación
- La calidad del método de clasificación se medirá con un único valor, correspondiente a la media de los 5 porcentajes de clasificación del conjunto de prueba

2. Descripción de los algoritmos

2.1. Componentes comunes

Para los algoritmos de esta práctica, vamos a definir primero los elementos comunes que comparten:

- **Esquema de representación**: Se seguirá la representación real basada en un vector W de tamaño n con valores en $[0, 1]$ que indican el peso asociado a cada característica y la capacidad para eliminarla si su peso es menor que 0.2.
- **Función objetivo**: Será la combinación con pesos de las medidas de precisión (tasa de acierto sobre el conjunto de entrenamiento) y la complejidad (la tasa de reducción de características con respecto al conjunto original) del clasificador 1-NN diseñado empleando el vector W. Para calcular la tasa de acierto será necesario emplear la técnica de validación leave-one-out. El valor de σ considerado será $\sigma = 0{,}5$, dándole la misma importancia a ambos criterios. El objetivo será maximizar esta función.
- **Generación de la solución inicial**: La solución inicial se generará de forma aleatoria utilizando una distribución uniforme en $[0, 1]$ en todos los casos. Para el caso de la DE se generarán todos los individuos de la población inicial de forma aleatoria.
- **Esquema de generación de vecinos en ES e ILS**: Se empleará el movimiento de cambio por mutación normal $Mov(W, \sigma)$ que altera el vector W sumándole otro vector Z generado a partir de una distribución normal de media 0 y varianza σ^2. Su aplicación concreta dependerá del algoritmo específico.
- **Algoritmo Búsqueda Local**: En ILS, se considerará la búsqueda local (BL) que sigue el enfoque del primer mejor vecino propuesta en la Práctica 1.b. Se detendrá la ejecución del algoritmo bien cuando no se encuentre mejora al generar un máximo número de vecinos o bien cuando se hayan evaluado 1000 soluciones distintas (en cualquier caso, en la BL, se parará también después de 1000 evaluaciones aunque siguiera habiendo soluciones mejores en el entorno).

2.2. Enfriamiento Simulado (ES)

Se ha de emplear un algoritmo ES con las siguientes componentes:

- **Esquema de enfriamiento**: Se empleará el esquema de Cauchy modificado:

  $$T_{k+1} = \frac{T_k}{1+\beta \cdot T_k} \quad ; \quad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

  donde M es el número de enfriamientos a realizar, T_0 es la temperatura inicial y T_f es la temperatura final que tendrá un valor cercano a cero.
- **Operador de Vecino y exploración del entorno para L(T)**: En cada iteración del bucle interno $L(T)$, se aplicará un único movimiento $Mov(W, \sigma)$ para generar una única solución vecina que será comparada con la solución actual. Se escogerá aleatoriamente la característica i a la que se le aplicará la perturbación.

- **Condición de enfriamiento L(T)**: Se enfriará la temperatura, finalizando la iteración actual, bien cuando se haya generado un número máximo de vecinos *máx_vecinos* (independientemente de si han sido o no aceptados) o bien cuando se haya aceptado un número máximo de los vecinos generados *máx_éxitos*.
- **Condición de parada**: El algoritmo finalizará bien cuando haya alcanzado el número máximo de evaluaciones prefijado o bien cuando el número de éxitos en el enfriamiento actual sea igual a 0.

Valores de parámetros y ejecuciones: La temperatura inicial se calculará en función de la siguiente fórmula:

$$T_0 = \frac{\mu \cdot C(S_0)}{-ln(\phi)}$$

donde T_0 es la temperatura inicial, $C(S_0)$ es el coste de la solución inicial y $\phi \in [0, 1]$ es la probabilidad de aceptar una solución un μ por 1 peor que la inicial. En las ejecuciones se considerará $\phi = \mu = 0{,}3$. La temperatura final T_f se fijará a 10^{-3} (¡comprobando siempre que sea menor que la inicial!).

Los parámetros que definen el bucle interno $L(T)$ tomarán valor *máx_vecinos*$=10 \cdot n$ (tamaño del caso del problema) y *máx_éxitos*$=0{,}1*$*máx_vecinos*. **El número máximo de evaluaciones será 15000**. Por lo tanto, el número de iteraciones (enfriamientos) M del algoritmo ES será igual a $15000/$*máx_vecinos*.

Comentar que aunque la práctica nos indicaba estos parámetros, pude comprobar que el algoritmo terminaba demasiado pronto por algunas condiciones. Lo que he realizado ha sido modificar la variable *max_exitos* de tal forma que **en mi implementación**, *máx_éxitos*$=0{,}4*$*máx_vecinos*.

2.3. Búsqueda Local Reiterada (ILS)

El algoritmo ILS consistirá en generar una solución inicial aleatoria y aplicar el algoritmo de BL sobre ella. Una vez obtenida la solución optimizada, se estudiará si es mejor que la mejor solución encontrada hasta el momento y se realizará una mutación sobre la mejor de estas dos, volviendo a aplicar el algoritmo de BL sobre esta solución mutada. Este proceso se repite un determinado número de veces, devolviéndose la mejor solución encontrada en todo el proceso. Por tanto, se sigue el criterio del mejor como criterio de aceptación de la ILS.

Tal y como se describe en las transparencias del Seminario 4, el operador de mutación de ILS estará basado en un operador de vecino que provoque un cambio más brusco en la solución actual que el considerado en la BL. Para ello, usaremos un operador que cambia conjuntamente el peso de t características escogidas aleatoriamente en la solución.

Valores de los parámetros y ejecuciones En cada ejecución del algoritmo se realizarán 15 iteraciones, es decir, se aplicará 15 veces el algoritmo de BL, la primera vez sobre una solución inicial aleatoria y las 14 restantes sobre soluciones mutadas. Se usará un valor $t = 0{,}1 \cdot n$ en el operador de mutación, es decir, se cambiará el valor del peso en un 10 % de las características. Además, para esta mutación se considerará un $\sigma = 0{,}4$. El número máximo de evaluaciones de cada BL será de 1000 evaluaciones.

2.4. Evolución Diferencial (DE)

La DE es un modelo evolutivo propuesto para optimización con parámetros reales que enfatiza la mutación y utiliza un operador de cruce/recombinación a posteriori. Consideraremos dos algoritmos de DE:

- **DE/Rand/1**: La fórmula para obtener el vector con mutación es la siguiente.

  $$V_{i,G} = X_{r1,G} + F \cdot (X_{r2,G} - X_{r3,G})$$

  Los índices r1, r2 y r3 de cada individuo i en cada generación G serán escogidos aleatoriamente de forma mutuamente excluyente (incluyendo el vector i-ésimo).

- **DE/current-to-best/1**: La fórmula para obtener el vector con mutación es la siguiente.

  $$V_{i,G} = X_{i,G} + F \cdot (X_{best,G} - X_{i,G}) + F \cdot (X_r1, G - Xr2, G)$$

  Los índices r1 y r2 de cada individuo i en cada generación G serán escogidos aleatoriamente de forma mutuamente excluyentes (incluyendo el vector i- ésimo). Xbest denota el mejor vector en la generación G.

La recombinación utilizada en ambos algoritmos de DE será la binominal (ver diapositiva 18 del Seminario 4). El reemplazamiento en ambos casos es one-to-one, es decir, selección del mejor entre padre e hijo. En esta versión, las componentes del vector solución se mantienen dentro de su rango permitido para no permitir soluciones inválidas.

Valores de los parámetros y ejecuciones El tamaño de la población será de 50 vectores en ambos modelos (*DE/Rand/1* y *DE/current-to-best/1*). La probabilidad de cruce CR será 0,5 y F valdrá 0,5. El criterio de parada en las dos versiones de DE consistirá en realizar **15000 evaluaciones de la función objetivo**.

3. Pseudocodigo algoritmos

3.1. Búsqueda Local (BL)

La BL utilizada es la misma que ne la práctica 1.b, no obstante, se ha cambiado la condición de parada, siendo ahora que el número de evaluaciones sea hasta 1000 en vez de 15000.

```
input  : datosTraining,datosTest,pesos,K
output: pesos[x, ..., n]
parada ← 20*pesos.size;
evaluaciones ← 0;
vecinosMal ← 0;
while evaluaciones < 1000 and vecinosMal < parada do
    mejorado ← false;
    for i ← 1 to pesos.size and not mejorado do
        nuevopeso ← calcularPesoNuevo(i,pesos);
        aciertos ← 0;
        for j ← 1 to datosTest.size do
            elemento ← datosTest[i];
            vecinos ← BusquedaVecinosCercanos(datosTraining,elemento,pesos,K);
            etiqueta ← Clasifica(vecinos);
            elemento.etiquetaPredicha ← etiqueta;
            if elemento.etiquetaPredicha = elemento.etiqueta then
                aciertos ← aciertos+1;
            end
        end
        tasa_clas ← CalcularTasaClas(aciertos);
        tasa_red ← CalcularTasaRed(pesos);
        valor ← F(tasa_clas,tasa_red);
        if valor > maxvalor then
            pesos ← nuevopeso;
            maxvalor ← valor;
            vecinosMal ← 0;
            mejorado ← true;
        else
            vecinosMal ← vecinosMal+1;
        end
        evaluaciones ← evaluaciones+1;
    end
end
return pesos;
```

Algorithm 1: BL

Para este algoritmo, utilizamos la siguiente forma de explorar el entorno donde al comprobar que

la distancia con el elemento a comparar no sea 0, estamos implementado el *leave-one-out*:

input : datosTraining,elemento,pesos,K
output: vecinos
vecinos ← [0, ..., 0];
while *vecinos.size* < *K* **do**
  datosTraining[i].distancia ← DistanciaEuclídea(datosTraining[i],elemento,pesos);
  if *datosTraining[i].distancia* ≠ 0,0 **then**
    vecinos.add(datosTraining[i]);
  end
end
for *i* ← *K* **to** *datosTraining.size* **do**
  datosTraining[i].distancia ← DistanciaEuclídea(datosTraining[i],elemento,pesos);
  peorVecino ← 0;
  for *j* ← 2 **to** *K* **do**
    if *vecino[j].distancia* > *vecino[peorVecino].distancia* **then**
      peorVecino ← j;
    end
  end
  if *vecino[peorVecino].distancia* > *datosTraining[i].distancia* **and** *datosTraining[i].distancia* ≠ 0,0 **then**
    vecinos[peorVecino] ← datosTraining[i];
  end
end

Algorithm 2: BusquedaVecinosCercanos

Y por último, para crear un nuevo vecino utilizo lo siguiente:

input : k,pesos
output: nuevopeso[$x, ..., n$]
nuevopeso ← pesos;
nuevopeso[k] ← pesos[k]+newGaussian();
nuevopeso[k] ← Truncar(pesos[k]) [0, 1];
return *nuevopeso*;

Algorithm 3: calcularPesoNuevo

3.2. Algoritmo Enfriamiento Simulado (ES)

Primero definiré el algoritmo en sí para después definir las propias funciones que utiliza.

Enfriamiento Simulado (ES) Comentar que las variables que no se definen en el propio algoritmo, son pasadas en el constructor del mismo y se utilizan en los distintos métodos al ser variables

globales.

```
input  :
output: pesos[x_i, ..., x_n]
max_vecinos ← 10*trainingSet.get(0).attributes.length;
max_exitos ← 0.4*max_vecinos;
vecinos ← -1;
exitos ← -1;
sol[] ← generarSolucionInicial();
bestSol[] ← sol;
knns ← new knn(trainingFile,testFile,K,metricType,trainingSet,testingSet,bestSol,0,true);
bestCost = knns.knn("ES", false);
f = bestCost;
TIni ← calcularTemperaturaInicial(0.3,0.3,bestCost);
TFin ← 0.001;
if TIni < TFin then
    return -1;
end
TAct ← TIni;
M ← 15000/(max_vecinos);
beta ← (TIni-TFin)/(M*TIni*TFin);
while TAct > TFin and exitos ≠ 0 do
    vecinos ← 0;
    exitos ← 0;
    while vecinos < max_vecinos and exitos < max_exitos do
        newSol[] = mutacion(sol);
        vecinos ← vecinos+1;
        knnNew ← new
          knn(trainingFile,testFile,K,metricType,trainingSet,testingSet,newSol,0,true);
        newF ← knnNew.knn("ES", false);
        dif ← f - newF;
        if (dif < 0) or (rnd.nextDouble() ≤ exp(-dif/(1*TAct))) then
            exitos ← exitos+1;
            sol ← newSol;
            f ← newF;
            if newF > bestCost then
                bestSol ← sol;
                bestCost ← newF;
            end
        end
    end
    TAct ← actualizarTemperatura(TAct, beta);
end
return bestSol
```

Algorithm 4: ES

Pasamos ahora a definir cada una de las funciones utilizadas en el algoritmo.

Generar solución inicial

Esta función nos creara una solución inicial aleatoria para poder comenzar a explorar las distintas soluciones del problema.

input :
output: pesos$[x_i, ..., x_n]$
tam $\leftarrow$ problema.size();
res $\leftarrow \emptyset$;
for $i \leftarrow 1$ **to** *tam* **do**
  res[i] $\leftarrow$ rnd.nextDouble();
end
return *res*;

Algorithm 5: generarSolucionInicial

Calcular temperatura inicial.

Dado un μ, un ϕ y el valor de la solución inicial ($C(S_0)$), devuelve la temperatura inicial utilizando la formula explicada en los apartados anteriores 2.2.

input : μ,ϕ,f
output: TIni
return $((\mu * f)/(-ln(\phi)))$;

Algorithm 6: calcularTemperaturaInicial

Actualizar temperatura.

Dada una temperatuca actual y un valor de β, calcula la nueva temperatura utilizando la fórmula explicada anteriormente 2.2.

input : TAct,β
output: TAct
return $(TAct/(1 + \beta * TAct))$;

Algorithm 7: actualizarTemperatura

3.3. Búsqueda Local Reiterada (ILS)

Búsqueda Local Reiterada (ILS)

input :
output: pesos$[x_i, ..., x_n]$
sol ← generarSolucionInicial();
bl ← new BL(trainingFile,K,metricType,trainingSet,sol,0,rnd);
sol ← bl.BL();
knns ← new knn(trainingFile,testFile,K,metricType,trainingSet,testingSet,sol,0,true);
f ← knns.knn("ils", false);
for *i* ←*1* **to** *14* **do**
  newSol ← mutacion(sol);
  bl ← new BL(trainingFile,K,metricType,trainingSet,newSol,0,rnd);
  newSol ← bl.BL();
  knnNew ← new knn(trainingFile,testFile,K,metricType,trainingSet,testingSet,newSol,0,true);
  newF ← knnNew.knn("ils", false);
  if *newF* > *f* **then**
    sol ← newSol;
    f ← newF;
  end
end
return *sol*;

Algorithm 8: ILS

Pasamos ahora a definir los distintos métodos que utiliza.

Generar solución inicial

Este método es exactamente igual que el del ES [5].

Mutación

En este caso, como necesitábamos una mutación algo mas fuerte de lo normal porque así lo pide

el propio algoritmo, he definido lo siguiente:

input : sol$[x_i, ..., x_n]$
output: pesos$[x_i, ..., x_n]$
res[] ← sol;
posiciones[] ← ∅;
t ← 0.1*sol.length;
for *i←1* **to** *t* **do**
    k ← rnd.nextInt(sol.length);
    while *posiciones.contains(k)* **do**
        k ← rnd.nextInt(sol.length);
    end
    posiciones.add(k);
    res[k] ← sol[k]+(rnd.nextGaussian()*0.4);
    res[k] ← Truncar(res[k]) [0,1];
end
return *res*;

Algorithm 9: mutacion

3.4. Evolución Diferencial (DE)

Para este algoritmo, tenemos que implemetar dos modelos distintos, por ello, en el mismo algoritmo se le pasa como parámetro un entero que determina el modelo a utilizar, siendo 0 el *DE/Rand/1* y 1 el *DE/current-to-best/1*.

Para este caso tenemos la variable bestSol, mejor, bestCost y mejorCost como **variables globales** donde bestSol y bestCost determinan la mejor solucion y su valor de todas las generaciones que llevamos, mientras que mejor y mejorCost nos indican el mejor de una poblacion determinada. La variable **evaluaciones** es una variable global, para poder modificarla en cualquier método de la clase. La variable **CR** se indica en el constructor, es una variable global e indica la probabilidad de cruce.

Evolución diferencial (DE)

```
input  : modelo
output: pesos[x_i, ..., x_n]
P[] ← ∅;
E[] ← ∅;
P ← inicializarPoblacion(50);
E ← evaluarPoblacion(P);
while this.evaluaciones < 15000 do
    newP[] ← ∅;
    newE[] ← ∅;
    hijo[] ← ∅;
    for i ← 1 to P.size() do
        padres[] ← seleccionarPadres(P,i);
        for k ← 1 to padres[0].length do
            if rnd.nextDouble() < this.CR then
                hijo[k] ← aplicarModelo(P,padres,modelo,k,i);
            else
                hijo[k] ← P.get(i)[k];
            end
        end
        knns ← new knn(trainingFile,testFile,K,metricType,trainingSet,testingSet,hijo,0,true);
        newP.add(hijo);
        newE.add(knns.knn("DE", false));
        this.evaluaciones ← this.evaluaciones+1;
    end
    actualizarPoblacion(P,E,newP,newE);
end
return bestSol;
```

Algorithm 10: Evolucion Diferencial

Ahora voy a definir los distintos métodos que utiliza éste algoritmo.

Inicializar poblacion

Indicando el número de elementos que va a tener la problación, nos crea una población del tamaño

indicando donde cada uno de los elementos están inicializados aleatoriamente.

input : tam
output: poblacion[$x_1[], x_2[], ..., x_n[]$]
res[][] $\leftarrow \emptyset$;
for $i \leftarrow 1$ **to** *tam* **do**
    aux[] $\leftarrow \emptyset$;
    for $j \leftarrow 1$ **to** *problema.size()* **do**
        aux[j] $\leftarrow$ rnd.nextDouble();
    end
    res.add(aux);
end
return *res*;

Algorithm 11: inicializarPoblacion

Seleccionar padres

Este método selecciona 3 padres para un determinado elemento de la población. La selección de estos padres es mutuamente excluyente como nos indican en el enunciado.

input : P[$x_1[], x_2[], ..., x_n[]$],pos
output: padres[$x_1, x_2, ..., x_n$]
res[] $\leftarrow \emptyset$;
padres[] $\leftarrow \emptyset$;
padres.add(pos);
for $i \leftarrow 1$ **to** *3* **do**
    padre $\leftarrow$ rnd.nextInt(P.size());
    while *padres.contains(padre)* **do**
        padre $\leftarrow$ rnd.nextInt(P.size());
    end
    padres.add(padre);
    res[i] $\leftarrow$ padre;
end
return *res*;

Algorithm 12: seleccionarPadres

Aplicar modelo

Este método aplica un modelo a un elemento para obtener el valor del hijo. En nuestro caso, al tener dos modelos, diferencia entre estos utilizando la variable modelo. Con esto conseguimos que

únicamente cambiando el modelo, podamos reutilizar código y que funcione correctamente.

input : P[x_1[], x_2[], ..., x_n[]],padres[x_1, x_2, x_3],modelo,k,i
output: valor
if *modelo = 0* **then**
    res $\leftarrow$ ($P.get(padres[0])[k] + this.F * (P.get(padres[1])[k] - P.get(padres[2])[k])$);
    res $\leftarrow$ Truncar(res) [0,1];
    return *res*;
else
    res $\leftarrow$ ($P.get(i)[k] + this.F * (this.mejor[k] - P.get(i)[k]) + this.F * (P.get(padres[0])[k] - P.get(padres[1])[k])$);
    res $\leftarrow$ Truncar(res) [0,1];
    return *res*;
end

Algorithm 13: aplicarModelo

Actualizar poblacion

Este método nos actualiza la población para obtener la nueva población. Actualiza tanto la población en sí como sus evaluaciones. Los parámetros se pasan por referencia y se actualizan en el propio método. Recordar que las variables **mejor,mejorCost,bestSol y bestCost** son variables

globales de la case.

input : P[x_1[], x_2[], ..., x_n[]],E[x_1, x_2, ..., x_n],newP[z_1[], z_2[], ..., z_n[]],newE[z_1, z_2, ..., z_n]
output:
res[][] ← ∅;
Eres[] ← ∅;
this.mejor ← ∅;
this.mejorCost ← 0;
for $i \leftarrow 1$ **to** *P.size()* **do**
  if *E.get(i) > newE.get(i)* **then**
    res.add(P.get(i));
    Eres.add(E.get(i));
  else
    res.add(newP.get(i));
    Eres.add(newE.get(i));
  end
  if *Eres.get(i) > this.mejorCost* **then**
    this.mejor ← res.get(i);
    this.mejorCost ← Eres.get(i);
  end
end
if *this.mejorCost > this.bestCost* **then**
  this.bestSol ← this.mejor;
  this.bestCost ← this.mejorCost;
end
P ← res;
E ← Eres;

Algorithm 14: actualizarPoblacion

4. Descripción para ejecución

En mi caso, he creado un proyecto con NetBeans para esta practica, por lo que adjuntado a este PDF irá un zip con el proyecto completo, con el .jar para poder ejecutarlo.

El proyecto esta compuesto por los siguientes elementos:

- **Carpeta src/practica3MH**: Donde se encuentran todos los ficheros implementados.
- **Carpeta src/input**: En esta capeta tenemos los 3 ficheros de entrada de la práctica.
- **Carpeta src/output**: Donde se generan los ficheros de salida del programa con la siguiente estructura.

  Se genera un fichero con el nombre de "ficheroEntrada-Algoritmo-train/test-Particion.txt" donde podemos ver por el nombre a qué fichero de entrada pertenecen, el algoritmo utilizado, train o test, donde train indica que son datos de entrenamiento, es decir, devuelve el tiempo y el vector resultado. Y test, que indica que que son datos de test por lo que tendremos los

porcentajes de acierto, etc... Por último vemos un numero que indica la partición utilizada como test, por lo que tendremos desde el 0 al 4.

Para la **ejecución del programa**, debe descomprimir el zip que va adjuntado con la práctica y una vez descomprimido, moverse dentro de él. Podrá encontrar las distintas carpetas y un fichero llamado Practica3MH.jar. Abra un terminal y ejecute **java -jar Practica3MH.jar**. Comenzará la ejecución y cuando finalice, tendrá los ficheros de salida en **src/output**

5. Análisis de los resultados

5.1. Descripción

En este apartado discutiremos los distintos resultados obtenidos de clasificación. En mi caso, **la semilla utilizada** es 3395+(particion*3+fichero).

- **Enfriamiento Simulado (ES)**

  Consideraremos un $\phi = \mu = 0{,}3$. La temperatura final será 10^{-3}. Los parámetros que definen el bucle interno tomarán valor *max_vecinos*= $10 \cdot n$ y *max_exitos*= $0{,}4 \cdot$*max_vecinos*. El número máximo de evaluaciones será 15000, por lo que M será igual a 15000/*max_vecinos*.

- **Búsqueda Local Reiterada (ILS)**

  En cada ejecución del algoritmo se realizarán 15 iteraciones, es decir, se aplicará 15 veces el algoritmo de BL, la primera vez sobre una solución inicial aleatoria y las 14 restantes sobre soluciones mutadas. Se usará un valor $t = 0{,}1 \cdot n$ en el operador de mutación, es decir, se cambiará el valor del peso en un 10 % de las características. Además, para esta mutación se considerará un $\sigma = 0{,}4$. El número máximo de evaluaciones de cada BL será de 1000 evaluaciones.

- **Evolución Diferencial (DE)**

  El tamaño de la población será de 50 vectores en ambos modelos (*DE/Rand/1 y DE/current-to-best/1*). La probabilidad de cruce *CR* será 0,5 y F valdrá 0,5. El criterio de parada en las dos versiones de DE consistirá en realizar 15000 evaluaciones de la función objetivo.

5.2. Resultados obtenidos

5.2.1. 1-NN

Donde en mi caso, particion 1 esta compuesta por:

- **Train** → particiones 1,2,3,4.
- **Test** → particion 0.

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

	Ozone				**Parkinsons**				**Spectf-heart**			
	%_clas	**%_red**	**Agr.**	**T**	**%_clas**	**%_red**	**Agr.**	**T**	**%_clas**	**%_red**	**Agr.**	**T**
Particion 1	73.44	0.00	36.72	0.00	76.92	0.00	38.46	0.00	67.14	0.00	33.57	0.00
Particion 2	84.38	0.00	42.19	0.00	82.05	0.00	41.03	0.00	77.14	0.00	38.57	0.00
Particion 3	73.44	0.00	36.72	0.00	100.00	0.00	50.00	0.00	65.71	0.00	32.86	0.00
Particion 4	81.25	0.00	40.63	0.00	66.66	0.00	33.33	0.00	62.86	0.00	31.43	0.00
Particion 5	85.94	0.00	42.97	0.00	74.35	0.00	37.18	0.00	69.57	0.00	34.78	0.00
Media	79.69	0.00	39.84	0.00	80.00	0.00	40.00	0.00	68.48	0.00	34.24	0.00

Para la particion 2 se cambia el test por particion 1 y el train serían las demás. Y así sucesivamente. Esto se aplica a **todos los casos de todas las tablas**.

5.2.2. Greedy

Tabla 2: Resultados obtenidos por el algoritmo Greedy en el problema del APC

	Ozone				**Parkinsons**				**Spectf-heart**			
	%_clas	**%_red**	**Agr.**	**T**	**%_clas**	**%_red**	**Agr.**	**T**	**%_clas**	**%_red**	**Agr.**	**T**
Particion 1	73.44	0.00	36.72	0.21	82.05	0.00	41.03	0.15	67.14	0.00	33.57	0.21
Particion 2	85.94	0.00	42.97	0.08	74.36	0.00	37.18	0.01	78.57	0.00	39.29	0.07
Particion 3	73.44	0.00	36.72	0.07	100.00	0.00	50.00	0.01	72.86	0.00	36.43	0.06
Particion 4	75.00	0.00	37.50	0.07	69.23	0.00	34.62	0.02	72.86	0.00	36.43	0.07
Particion 5	71.88	0.00	35.94	0.07	84.62	0.00	42.31	0.01	65.22	0.00	32.61	0.06
Media	75.94	0.00	37.97	0.10	82.05	0.00	41.03	0.04	71.33	0.00	35.66	0.09

5.2.3. Enframiento Simulado (ES)

Tabla 3: Resultados obtenidos por el algoritmo de Enfriamiento Simulado en el problema del APC

	Ozone				**Parkinson**				**Spectf-heart**			
	%_clas	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**
Particion0	71,88	80,56	76,22	88,64	89,74	81,82	85,78	12,08	95,71	81,82	88,77	79,90
Particion1	84,38	70,83	77,60	92,54	76,92	90,91	83,92	10,64	88,57	68,18	78,38	64,51
Particion2	70,31	83,33	76,82	102,93	94,87	86,36	90,62	11,38	80,00	72,73	76,36	75,43
Particion3	81,25	83,33	82,29	99,88	64,10	90,91	77,51	9,95	78,57	75,00	76,79	67,70
Particion4	81,25	83,33	82,29	102,52	69,23	90,91	80,07	9,24	95,65	77,27	86,46	76,53
Media	77,81	80,28	79,05	97,30	78,97	88,18	83,58	10,66	87,70	75,00	81,35	72,81

5.2.4. Búsqueda Local Reiterada (ILS)

Tabla 4: Resultados obtenidos por el algoritmo ILS en el problema del APC

	Ozone				**Parkinson**				**Spectf-heart**			
	%_clas	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**
Particion0	70,31	90,28	80,30	187,06	71,79	81,82	76,81	11,86	98,57	84,09	91,33	106,66
Particion1	76,56	81,94	79,25	184,32	82,05	90,91	86,48	13,43	91,43	84,09	87,76	127,30
Particion2	75,00	86,11	80,56	188,86	94,87	90,91	92,89	13,00	74,29	86,36	80,32	89,15
Particion3	76,56	81,94	79,25	192,61	64,10	90,91	77,51	13,14	78,57	84,09	81,33	107,58
Particion4	78,13	87,50	82,81	184,93	79,49	86,36	82,93	10,45	98,55	88,64	93,59	118,11
Media	75,31	85,56	80,43	187,55	78,46	88,18	83,32	12,38	88,28	85,45	86,87	109,76

5.2.5. Evolución Diferencial (*DE/Rand/1*)

Tabla 5: Resultados obtenidos por el algoritmo Evolución Diferencial *DE/Rand/1* en el problema del APC

	Ozone				**Parkinson**				**Spectf-heart**			
	%_clas	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**
Particion0	71,88	91,67	81,77	188,00	79,49	90,91	85,20	21,36	97,14	90,91	94,03	107,52
Particion1	75,00	90,28	82,64	182,49	82,05	90,91	86,48	20,44	88,57	90,91	89,74	127,59
Particion2	70,31	90,28	80,30	191,11	94,87	90,91	92,89	21,33	75,71	88,64	82,18	131,22
Particion3	76,56	91,67	84,11	182,25	66,67	90,91	78,79	18,90	85,71	90,91	88,31	98,41
Particion4	87,50	91,67	89,58	185,25	66,67	90,91	78,79	22,46	97,10	93,18	95,14	129,51
Media	76,25	91,11	83,68	185,82	77,95	90,91	84,43	20,90	88,85	90,91	89,88	118,85

5.2.6. Evolución Diferencial (*DE/current-to-best/1*)

Tabla 6: Resultados obtenidos por el algoritmo Evolución Diferencial *DE/current-to-best/1* en el problema del APC

	Ozone				**Parkinson**				**Spectf-heart**			
	%_clas	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**	**%_clas**	**%_red**	**F**	**T**
Particion0	75,00	65,28	70,14	172,36	89,74	81,82	85,78	21,02	97,14	75,00	86,07	104,08
Particion1	82,81	65,28	74,05	164,83	76,92	81,82	79,37	21,68	92,86	70,45	81,66	115,33
Particion2	76,56	70,83	73,70	180,20	92,31	90,91	91,61	18,70	70,00	72,73	71,36	126,28
Particion3	78,13	79,17	78,65	199,81	64,10	90,91	77,51	19,14	82,86	75,00	78,93	105,47
Particion4	76,56	63,89	70,23	174,15	74,36	86,36	80,36	17,09	98,55	77,27	87,91	112,72
Media	77,81	68,89	73,35	178,27	79,49	86,36	82,93	19,53	88,28	74,09	81,19	112,78

5.2.7. Tabla resumen

Tabla 7: Resultados resumen en el problema del APC

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
1NN	79,69	0,00	39,84	0,00	80,00	0,00	40,00	0,00	68,48	0,00	34,24	0,00
RELIEF	75,94	0,00	37,97	0,10	82,05	0,00	41,03	0,00	71,33	0,00	35,66	0,09
ES	77,81	80,28	79,05	97,30	78,97	88,18	83,58	10,66	87,70	75,00	81,35	72,81
ILS	75,31	85,56	80,43	187,55	78,46	88,18	83,32	12,38	88,28	85,45	86,87	109,76
DE/rand/1	76,25	91,11	83,68	185,82	77,95	90,91	84,43	20,90	88,85	90,91	89,88	118,85
DE/current-to-best/1	77,81	68,89	73,35	178,27	79,49	86,36	82,93	19,53	88,28	74,09	81,19	112,78

A fin de hacer los datos un poco más visuales, voy a mostrar una tabla donde se clasifican por colores y por columnas, siendo el amarillo el peor resultado de la columna, el naranja un valor medio, y el rojo el mejor valor de la columna.

	Ozone				Parkinson				Spectf-heart			
	%_clas	%_red	F	T	%_clas	%_red	F	T	%_clas	%_red	F	T
1NN	79,69	0,00	39,84	0,00	80,00	0,00	40,00	0,00	68,48	0,00	34,24	0,00
RELIEF	75,94	0,00	37,97	0,10	82,05	0,00	41,03	0,00	71,33	0,00	35,66	0,09
ES	77,81	80,28	79,05	97,30	78,97	88,18	83,58	10,66	87,70	75,00	81,35	72,81
ILS	75,31	85,56	80,43	187,55	78,46	88,18	83,32	12,38	88,28	85,45	86,87	109,76
DE/rand/1	76,25	91,11	83,68	185,82	77,95	90,91	84,43	20,90	88,85	90,91	89,88	118,85
DE/current-to-best/1	77,81	68,89	73,35	178,27	79,49	86,36	82,93	19,53	88,28	74,09	81,19	112,78

5.3. Análisis de resultados

Empezamos analizando **ES**, podemos ver sus resultados en la tabla 3 que se encuentra en la página 17.

- Podemos ver una mejora significativa en todos los sentidos respecto a los algoritmos desarrollados con anterioridad a éste. Esto es debido básicamente a la capacidad de reducción que hace que este valor aumente y por consecuencia, que la media o valor de la función objetivo aumente bastante más.

  Si nos fijamos en el porcentaje de clasificación, no podemos determinar una mejora/empeora sobre en 1NN o en RELIEF ya que vemos que dependiendo del fichero al que estamos evaluando un algoritmo acierta más que otro. Lo que si podemos sacar en conclusión es que en Ozone hemos conseguido reduciendo un 80 % que el porcentaje de acierto no varíe apenas, sin embargo, en parkison, al tener menos características y eliminar un 88 % se ha reducido la tasa de acierto respescto a los otros algoritmos que utilizan todas las características. Por último, en Spectf-heart vemos claramente que hay muchas variables que meten ruido o que no dan información alguna, ya que hemos conseguido que reduciendo un 75 % de las variables, aumentemos incluso más el porcentaje de acierto, por lo que hemos conseguido eliminar variables sin información y potenciar las verdaderamente valiosas.

  Claramente aumenta el tiempo necesario para obtener las soluciones, ésto es simplemente porque el algoritmo ES necesita un cómputo y un procesamiento que el 1NN no tiene (nulo) y el Greedy no tiene prácticamente nada en comparación con éste.

- Respecto a los algoritmos desarrolados en esta misma practica, podemos ver que el ES no es el más potente de ellos, no obstante, modificando los parámetros podemos llegar a obtener unos resultados algo más elevados. No obstante, no hay mucha diferencia ni en tasa de reduccion

como en tasa de acierto con los demás algoritmos, podemos decir que baja un poco en tasa de reduccion respescto a los demás, pero si comparamos el tiempo necesario para obtener las soluciones, vemos que éste algoritmo es el más rapido de ellos y hemos conseguido unas soluciones competitivas respecto a los demás aunque no sean las mejores.

- Como conclusión final, hemos obtenido unos resultados bastante aceptables ya que rondan un 80 % de media, que aunque no sea un valor deseado, está bastante bien. Claramente la búsqueda de trayectorias simples consigue mejores resultados que los algoritmos anteriores, pero no consigue superar a algorimos como Evolución Diferencial aunque mejore en tiempo. Claramente estas conclusiones son para este tipo de problema, no quiere decir que en otro escenario distinto el ES supere a DE.

Seguimos ahora analizando el **ILS** que está en la tabla 4 que se encuentra en la página 18.

- Podemos comprobar como éste algoritmo ha conseguido mejorar respecto a los algoritmos base (1NN, Greedy), la razon es exactamente la misma que para el apartado anterior donde comparabamos el ES con éstos mismos algoritmos.
- Comparándolo ahora con los algoritmos realizados en esta practica, vemos que incluso mejora a ES y consigue obtener unos resultados mejores. Vemos que en tasa de acierto no aumenta apenas, el cambio lo encontramos en la tasa de reducción, este algoritmo ha conseguido eliminar más variables sin perder porcentaje de acierto. Aunque hemos mejorado, estamos en el mismo caso de ES, obtenemos buenos resultados pero no consiguen superar a DE en este problema concreto.
- Como conclusión, utilizando un algoritmo de trayectorias múltiples, hemos conseguido superar al algoritmo de trayectorias simples (aunque sacrificando un poco de tiempo), por lo que volvemos a tener unos resultados aceptables en media, aunque como en el anterior, no ha conseguido ser el mejor algoritmo para resolver este problema concreto con estos parámetros concretos.

El siguiente en analizar será el algoritmo **DE**, en concreto *DE/Rand/1* que tenemos en la tabla 5 que se encuentra en la página 18.

- En los datos vemos claramente la gran diferencia que hay entre este algoritmo y los algoritmos base (1NN, Greedy), sin embargo, como en los casos anteriores esto se refleja claramente por la capacidad de poder reducir variables y así subir el valor medio. Respecto a la tasa de acierto vemos que no varía e incluso consigue menos acierto en 2 casos de 3. Esto ocurre básicamente porque lo que busca es aumentar la media, que es la función objetivo, no la tasa de acierto.
- Si lo comparamos ahora con los algorimos desarrollados en esta practica, podemos concluir que éste algorimo es el mejor de todos en media, ya que consigue mejores valores en todos los casos e incluso en algunos de ellos con una diferencia significativa. Como llevo comentando en los apartados anteriores, la potencia está en reducir más variables que el resto ya que en porcentaje de acierto todos están mas o menos con los mismos valores, es por ello, que este algorimo consigue reducir más que el resto y en consecuencia se convierte en el mejor algoritmo desarollado para solucionar esta práctica con los parámetros que hemos indicado.
- Como conclusión, utilizando un algoritmo de Evolución Diferencial (*DE/Rand/1*) hemos consegido superara tanto a los básicos como a los más complejos y obtener unos resultados

bastante buenos y superando al resto de algoritmos.

Por último voy a analizar el **DE**, en este caso el *DE/current-to-best/1* que tenemos en la tabla 6 que se encuentra en la página 18.

- En este caso, el único cambio realizado ha sido el modelo, como se indica en el problema. Los resultados que he obtenido no han sido satisfactorios, aunque hablando sobre los algoritmos básicos (1NN y Greedy) si que lo son, gracias a la componente de reduccion.
- Si comparamos con los algoritmos de esta práctica, este algoritmo a pesar de ser un algoritmo de Evolución Diferencial como el anterior, al cambiar el modelo a utilizar, hemos obtenido unos resultados poco satisfactorios ya que esta vez no ha sido capaz de reducir tantas variables como en casos anteriores y esto hace que la media baje, ya que en porcentaje de acierto sigue compitiendo con los demás.
- Como conclusión, aunque el algoritmo de Evolución Diferencial nos ha dado el mejor resultado obtenido, al modificarlo y hacer que los hijos tengan también una componente del mejor de esa población, los resultados han empeorado. Esto nos dice que no por ser el mejor significa que realmente sea el camino correcto a seguir. No obstante esto se cumple para este problema concreto con los datos que tenemos y los parámetros que nos han especificado, para cada problema podríamos obtener conclusiones diferentes.

Como conclusión final, observando la rabla resumen[7], hemos obtenido unos buenos resultados en los algoritmos desarrollados en esta práctica, siendo el algoritmo de Evolución Diferencial el que encabeza el ranking seguido del ILS y por último del ES (aunque con poca diferencia estos dos últimos). También hemos podido comprobar que aunque el algoritmo sea "el mejor", si utilizamos otros métodos u otras configuraciones podemos variar y cambiar las posiciones del raking del agoritmo, no obstante, para este problema el algoritmo que podíamos elegir por las soluciones de calidad que nos ha devuelto sería el algoritmo de **Evolución Diferencial**.

Todas las conclusiones se pueden corroborar a través de la tabla 7 en la página 19 donde se podrá ver las distintas medias para cada uno de los algoritmos utilizados y comprobar todos los datos obtenidos.