Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Práctica 2: Técnicas de Búsqueda basadas en Poblaciones

Problema del Agrupamiento con Restricciones
Algoritmos Genéticos y Meméticos con sus distintas vertientes

Juan Emilio Martínez Manjón
77024428-G
juanemartinez@correo.ugr.es
Grupo 2 Jueves 17:30h

2019-2020

Índice

1 Descripción del problema

El problema escogido para resolver esta práctica se trata del problema del agrupamiento con restricciones (PAR).

Este problema se resume a grandes rasgos en agrupar datos en varios conjuntos dependiendo de las similitudes entre ellos. Estos conjuntos se llaman clusters, por lo que a este problema se le denomina también clustering.

Cada cluster está representado en el espacio por un centroide, el cual tiene tantas instancias como instancias tengan los datos del conjunto en cuestión. Cada dato está asignado a un centroide dependiendo de la distancia Euclidea a él. Por lo que el objetivo del problema del agrupamiento clásico era asignar cada dato a aquel cluster cuya distancia Euclidea a su centroide sea menor.

El problema PAR añade a esta ecuación las restricciones. Por lo que a parte de asignar un dato a un cluster disminuyendo la desviación general de las asignaciones, también deberemos tener en cuenta una serie de restricciones. Estas restricciones pueden ser del tipo Must Link (ML) que indican que dos datos deben pertenecer al mismo centroide. O Cannot Link (CL) que indican que dos datos deben pertenecer a centroides diferentes.

También podemos distinguir entre dos niveles de restricciones, las fuertes y las débiles. Las restricciones fuertes son las que debemos cumplir en todos los casos. Las restricciones débiles son aquellas que podemos cumplir o no. Y, en el caso de no cumplirlas, produciría un aumento en el grado de infactibilidad.

2 Descripción de la parte común de los algoritmos

A continuación se van a mostrar que forma de representación común podemos encontrar en ambos algoritmos. Además de mostrar pseudocódigo acerca de los operadores comunes y la forma de calcular la función objetivo.

2.1 Representación de los datos

Los centroides están representados como instancias de las clase Centroide. Dicha clase tiene la siguiente estructura:

```
class Centroide
        int numero
        vector datos asignados
        vector indices
```

En esta representación "numero" hace referencia a la etiqueta de dicho centroide, ya que cada conjunto de datos tiene varios centroides. El vector llamado "datos asignados" contiene todos los indices asignados a dicho centroide. El vector llamado "indices" contiene los valores de los parámetros de dicho centroide.

Además de los getters y setters, esta clase nos permite saber si un dato cualquiera está contenido en ese centroide.

Los datos están representados en un vector de listas, donde cada elemento de la lista corresponde a una instancia del dato. Este vector se rellena leyendo un fichero mediante un flujo de entrada. La función de rellenado de datos es común para varios algoritmos, se basa en leer datos y saltarse las comas almacenando los resultados en un vector de listas como he indicado previamente.

La solución final la representamos en un vector de centroides. Donde imprimiremos que indices del conjunto de datos han sido finalmente asignados a cada cluster después de ejecutar el algoritmo.

En este caso, al estar trabajando con algoritmos sobre poblaciones, vamos a representar una población como un conjunto de cromosomas. Donde cada cromosoma es uno de los vectores de centroides antes descritos. Cada cromosoma tendrá la siguiente estructura:

```
struct Cromosoma
        vector elementos
        float desviacion
        float funcion
```

```
int infactibilidad
```

Usando esta estructura podemos almacenar junto a cada vector de centroides su función objetivo, para no tener que recalcularla continuamente.

Además de representar la solución de esta manera también añadiremos sus correspondientes valores de desviación, infactibilidad y función objetivo. Dichos valores nos servirán para realizar una comparativa entre ambos algoritmos. Se imprimen estos valores en cada una de las versiones de los algoritmos genéticos y meméticos.

2.2 Operadores comunes

Los operadores que son comunes a todos los algoritmos nuevos de esta práctica son los siguientes:

- Actualizar un centroide
- Calcular la distancia máxima del conjunto
- Calcular la desviación
- Calcular la función objetivo
- Calcular la infactibilidad
- Comprobar que los índices de una asignación son correctos
- Generación de soluciones iniciales aleatorias
- Proceso de selección en los Algoritmos Genéticos
- Operador de cruce uniforme
- Operador de cruce por segmento fijo
- Operador de mutación

2.2.1 Actualizar un centroide

Para actualizar un centroide se recorrerán todos los datos asignados al centroide en cuestión, y se calculará la media de cada uno de los parámetros. Finalmente se modificarán los parámetros de dicho centroide por los que acabamos de calcular.

El pseudocódigo correspondiente es el siguiente:

Algorithm 1: actualizaCentroide(centroide, data_set)

```
Result: void
Inicializamos un array "medias" con tantas casillas como caracteristicas tengan
 los datos del dataset a 0;
for Cada dato asignado al centroide a actualizar do
    for Cada una de las caracteristicas del dato del bucle superior do
        medias[pos] += caracteristicas[pos];
    end
end
for Cada uno de los elementos en medias do
    medias[pos] = medias[pos] / numero datos asignados;
    indice_centroide[pos] = medias[pos]
end
```

2.2.2 Calcular la distancia máxima del conjunto

Para calcular la distancia máxima del conjunto debemos calcular las distancias dos a dos de todos los elementos del conjunto. Hasta finalmente devolver la mayor de todas.

El pseudocódigo correspondiente es el siguiente:

Algorithm 2: distanciaMaxima(data_set)

```
Result: distancia maxima
Inicializamos una variable distancia_maxima a 0;
for i=0; i < datos totales del dataset do
    for j=i+1; j < datos totales del dataset do
        Creamos el vector auxiliar "resultado";
        for Cada una de las caracteristicas del dato i do
            resultado.push_back((datoj[k]-datoi[k])*(datoj[k]-datoi[k]));
        end
        for Cada uno de los datos del vector resultado do
            distancia += resultado[pos];
        end
        if distancia > distancia_maxima then
            distancia_maxima = distancia;
        end
    end
end
```

2.2.3 Calcular la desviación general

Para calcular la desviación general deberemos calcular la desviación de cada uno de los centroides y sumarlas todas al final. Para calcular la desviación de un centroide deberemos calcular la media de las distancias Euclideas de cada uno de sus datos al propio centroide.

El pseudocódigo correspondiente es el siguiente:

Algorithm 3: calculaDesviacion(centroides, dataset)

```
Result: desviacion
distancia_euclidea = 0.0;
desviacion = 0.0;
sumatoria = 0.0;
for i=0; i < numero centroides do
    sumatoria = 0.0 ;
    for j=0; j < cada uno de los datos asignados al centroide i do
        distancia_euclidea = 0.0;
        distancia_euclidea += diferencias al cuadrado de cada caracteristica del
          dato menos el indice correspondiente del centroide ;
        sumatoria += distancia_euclidea;
    end
    sumatoria = sumatoria / numero de datos del centroide;
    desviacion += sumatoria;
end
desviacion = desviacion / numero_centroides;
```

2.2.4 Calcular la función objetivo

Para calcular la función objetivo necesitamos saber el valor de desviación general, de infactibilidad, y de lambda. Finalmente aplicaremos la formula:

Función = Desviación + (Infactibilidad * Lambda)

El calculo de la infactibilidad es diferente para cada uno de los algoritmos por lo que se explicará detalladamente más adelante. Lambda consiste en una constante resultante de dividir la distancia máxima del dataset entre el número máximo de restricciones (ML + CL).

El pseudocódigo correspondiente es el siguiente:

Algorithm 4: calculaFuncionObjetivo(centroides, dataset, restricciones, lambda)

Result: funcion_objetivo
desviacion = calculaDesviacion(centroides, dataset);
infactibilidad = calculaInfactibilidad(restricciones, dataset);
funcion_objetivo = desviacion + (infactibilidad * lambda);

2.2.5 Cálculo de infactibilidad

Para calcular la infactibilidad vamos a disponer de un struct que nos ayudará a indexar de forma más eficiente que en la matriz de restricciones. El struct tendría los siguientes atributos:

```
struct Restriccion
        int indice1
        int indice2
        int valor
```

Para rellenar el vector de restricciones, se crea una instancia de este struct por cada restricción ML o CL. En el atributo indice1 se guarda la posición de la fila y en el atributo indice2, la posición de la columna. En valor se guarda el tipo de restricción que obtendríamos al indexar dicha fila y columna en la matriz.

Esta función calcula la infactibilidad de un conjunto de indices, usando la codificación de struct de restricciones.

Algorithm 5: calculaInfactibilidad(indices, restricciones)

Result: infactibilidad
infactibilidad = 0;
for *i=0; i < cada una de las instancias del vector de restricciones* **do**
  if *restricciones[i].valor == ML and indices[restricciones[i].indice1] != indices[restricciones[i].indice2]* **then**
    infactibilidad++;
  end
  if *restricciones[i].valor == CL and indices[restricciones[i].indice1] == indices[restricciones[i].indice2]* **then**
    infactibilidad++;
  end
end

2.2.6 Comprobar si la asignación de centroides es válida

Esta función comprueba si las asignaciones a centroides del vector de asignaciones es correcta. Es decir, que no se quede ningún centroide vacío. Esta función es necesaria ya que los indices de cada cromosoma se cambian aleatoriamente. Además tendremos la posibilidad de devolver por referencia un vector con los centroides que se quedan vacíos, para poder repararlos.

Algorithm 6: compruebaIndicesCorrectos(indices)

Result: boolean es_valida
es_valida = true;
Inicializamos un array de booleanos llamado comprobaciones a false;
for *cada uno de los indices* **do**
  comprobaciones[indices i] = true;
end
Si al recorrer el array de comprobaciones, todas las posiciones son true, devolvemos true. Si alguna de las posiciones es false, devolvemos false y añadimos dicho cluster al vector de clusters vacíos.

2.2.7 Generación de soluciones aleatorias

A continuación se mostrará como se inicializan de forma aleatoria cada una de las poblaciones de los diferentes datasets. Cada población estará formada por 50 cromosomas aleatorios. Los cromosomas se inicializarán de la siguiente manera:

Algorithm 7: GeneracionAleatoriaPoblaciones

```
Result: void
Cromosoma aux;
for cada uno de los 50 elementos de la poblacion do
    while indices_iris no contenga indices correctos do
        indices_iris.clear();
        for int i=0; i<150; i++ do
            indices_iris.push_back(Randint(0,2));
        end
    end
    aux.elementos = indices_iris;
    poblacion_iris.push_back(aux);
    while indices_rand no contenga indices correctos do
        indices_rand.clear();
        for int i=0; i<150; i++ do
            indices_rand.push_back(Randint(0,2));
        end
    end
    aux.elementos = indices_rand;
    poblacion_rand.push_back(aux);
    while indices_ecoli no contenga indices correctos do
        indices_ecoli.clear();
        for int i=0; i<336; i++ do
            indices_ecoli.push_back(Randint(0,7));
        end
    end
    aux.elementos = indices_ecoli;
    poblacion_ecoli.push_back(aux);
    while indices_newthyroid no contenga indices correctos do
        indices_newthyroid.clear();
        for int i=0; i<215; i++ do
            indices_newthyroid.push_back(Randint(0,2));
        end
    end
    aux.elementos = indices_newthyroid;
    poblacion_newthyroid.push_back(aux);
end
```

2.2.8 Proceso de selección en los algoritmos genéticos

A continuación se mostrará la forma de seleccionar los padres en cada generación de un algoritmo genético. La única diferencia en este apartado entre los algoritmos generacionales y los estacionarios es que en el primero seleccionamos 50 padres y en el otro, solamente 2. La forma de elegir un padre en ambos casos será realizando un torneo binario con dos individuos elegidos aleatoriamente.

Algorithm 8: SeleccionPadres

```
Result: padres_seleccionados
for Desde i = 0 hasta que i sea menor que 50 (AGG) ó 2 (AGE) do
    padre_aleatorio1 = Randint(0,49);
    padre_aleatorio2 = Randint(0,49);
    if funcion_padre1 < funcion_padre2 then
        padres_seleccionados.push_back(padre_aleatorio1);
    else
        padres_seleccionados.push_back(padre_aleatorio2);
    end
end
```

2.2.9 Operador de cruce uniforme

A continuación se mostrará una de las funciones con la que se cruzan los padres seleccionados en el proceso de selección. En el caso de los algoritmos generacionales, se cruzarán 17 parejas consecutivas de padres. Este valor se obtiene de realizar la operación: Numero de parejas (25) * probabilidad de cruce (0.7). En el caso de los algoritmos estacionarios, se cruzará siempre la única pareja de padres que tenemos. Cada pareja que se cruce generará dos hijos que los sustituirán. El siguiente pseudocódigo es el relacionado con el cruce uniforme, que obtiene información uniforme de cada padre.

Algorithm 9: funcion_cruce_uniforme(padres, cruces_esperados, num_centroides, cambios_f_objetivo, centroides, dataset, restrictions, lambda, num_caracteristicas)

```
Result: padres_cruzados
vector indices_que_cruzan = false;
for i = 0 hasta que i sea menor que el numero de cruces esperados do
    for n = 0 hasta que n sea menor que 2 do
        for j = 0 hasta que j sea menor que el tamanio del cromosoma / 2 do
            while indice_a_cruzar no se repita do
                indice_a_cruzar = Randint(0, tamanio_cromosoma-1);
            end
            indices_que_cruzan[indice_a_cruzar] = true;
        end
        for j = 0 hasta que j sea menor que el tamanio de un cromosoma do
            if indices_que_cruzan[j] then
                hijo_cruzado.push_back(padre1[j]);
            else
                hijo_cruzado.push_back(padre2[j]);
            end
        end
        while Indices del hijo cruzado no sean correctos do
            Reparamos los clusters vacios del hijo cruzado rellenandolos
             aleatoriamente
        end
        Cromosoma aux;
        aux.elementos = hijo_cruzado;
        aux.funcion = calculaFuncionObjetivo(hijo_cruzado);
        padres_cruzados.push_back(aux);
    end
end
Metemos en padres_cruzados el resto de padres que no cruzan
return padres_cruzados;
```

2.2.10 Operador de cruce por segmento fijo

A continuación se mostrará una de las funciones con la que se cruzan los padres seleccionados en el proceso de selección. Se mantiene todo lo explicado sobre el proceso de cruzamiento en la subsección anterior. El siguiente pseudocódigo es el relacionado con el cruce por segmento fijo.

Algorithm 10: funcion_cruce_segmento_fijo(padres, cruces_esperados, num_centroides, cambios_f_objetivo, centroides, dataset, restrictions, lambda, num_caracteristicas)

```
Result: padres_cruzados
vector indices_que_cruzan = false;
vector indices_que_cruzan_fijos = false;
for i = 0 hasta que i sea menor que el numero de cruces esperados do
    for n = 0 hasta que n sea menor que 2 do
        inicio_segmento = Randint(0,tamanio_cromosoma-1);
        tamanio_segmento = Randint(0,tamanio_cromosoma-1);
        if ((inicio_segmento + tamanio_segmento) mod tamanio_cromosoma) >
         inicio_segmento then
            for int j=inicio_segmento; j < ((inicio_segmento +
             tamanio_segmento) mod tamanio_cromosoma); j++ do
                indices_que_cruzan_fijos[j] = true;
            end
        else
            for int j=inicio_segmento; j < tamanio_cromosoma; j++ do
                indices_que_cruzan_fijos[j] = true;
            end
            for int j=0; j < ((inicio_segmento + tamanio_segmento) mod
             tamanio_cromosoma); j++ do
                indices_que_cruzan_fijos[j] = true;
            end
        end
        for j = 0 hasta que j sea menor que el tamanio del cromosoma / 2 do
            while indice_a_cruzar no se repita y no sea uno de los indices fijos do
                indice_a_cruzar = Randint(0, tamanio_cromosoma-1);
            end
            indices_que_cruzan[indice_a_cruzar] = true;
        end
        for j = 0 hasta que j sea menor que el tamanio de un cromosoma do
            if indices_que_cruzan[j] or indices_que_cruzan_fijos[j] then
                hijo_cruzado.push_back(padre1[j]);
            else
                hijo_cruzado.push_back(padre2[j]);
            end
        end
        while Indices del hijo cruzado no sean correctos do
            Reparamos los clusters vacios del hijo cruzado rellenandolos
             aleatoriamente
        end
        Cromosoma aux;
        aux.elementos = hijo_cruzado;
        aux.funcion = calculaFuncionObjetivo(hijo_cruzado);
        padres_cruzados.push_back(aux);
    end
end
Metemos en padres_cruzados el resto de padres que no cruzan
return padres_cruzados;
```

2.2.11 Operador de mutación

A continuación se mostrará el operador de mutación que se aplica sobre un número determinado de padres cruzados. El numero de mutaciones que se realizan varían dependiendo del tipo de algoritmo genético. En el caso de los algoritmos generacionales se realizaran un número de mutaciones igual a: Numero de padres (50) * Tamanio dataset * Probabilidad mutación (0.001). En el caso de los algoritmos estacionarios no se multiplicará por el número de padres.

Algorithm 11: MutacionPadresCruzados

```
Result: void
for j = 0 hasta que j sea menor que el numero de mutaciones esperadas do
    while Los valores del auxiliar no sean correctos y el nuevo indice no sea
     diferente al que ya habia do
        aux = padre_cruzado[j];
        gen_aleatorio = Randint(0,dataset.size()-2);
        nuevo_indice_aleatorio = Randint(0, centroides.size()-1);
        aux[gen_aleatorio] = nuevo_indice_aleatorio;
    end
    (padre_cruzado[j])[gen_aleatorio] = nuevo_indice_aleatorio;
    padre_cruzado[j].funcion = calculaFuncionObjetivo(padre_cruzado[j]);
end
```

3 Implementación y desarrollo de cada algoritmo

A continuación se van a detallar las estructuras y operador relevantes de cada algoritmo, así como la implementación del procesamiento de cada uno de ellos.

3.1 Algoritmos Genéticos

En este apartado vamos a mostrar el pseudocódigo del proceso de evolución y reemplazamiento de los algoritmos genéticos. No es necesario hacer distinción entre los dos tipos de algoritmos genéticos que tenemos (AGG y AGE) ya que el esquema general es el mismo. Solo difieren entre ellos en el número de padres que se seleccionan en cada generación y el número estimado de cruces y mutaciones de cada uno. El proceso de reemplazamiento también difiere entre un tipo y otro, por lo que será lo primero que mostremos.

Algorithm 12: ReemplazamientoAGG

```
Result: void
for Cada uno de los cromosomas en la poblacion do
    if poblacion[i].funcion < mejor_funcion_objetivo then
        mejor_funcion_objetivo = poblacion[i].funcion;
        mejor_cromosoma = i;
    end
end
if El mejor cromosoma de la poblacion no esta ya en los padres cruzados then
    for Cada uno de los cromosomas de los padres cruzados do
        if padres_cruzados[l].funcion > peor_funcion_objetivo then
            peor_funcion_objetivo = padres_cruzados[l].funcion;
            peor_cromosoma = l;
        end
    end
    padres_cruzados[peor_cromosoma] = poblacion[mejor_cromosoma];
end
poblacion = padres_cruzados;
```

Algorithm 13: ReemplazamientoAGE

```
Result: void
for Cada uno de los cromosomas en la poblacion do
    if poblacion[i].funcion > peor_funcion_objetivo then
        peor_funcion_objetivo = poblacion[i].funcion;
        cromosoma_anterior = peor_cromosoma;
        peor_cromosoma = i;
    end
end
if padres_cruzados[0].funcion <= padres_cruzados[1].funcion then
    mejor_cromosoma = 0;
    otro_cromosoma = 1;
else
    mejor_cromosoma = 1;
    otro_cromosoma = 0;
end
if padres_cruzados[mejor_cromosoma].funcion <
 poblacion[peor_cromosoma].funcion then
    poblacion[peor_cromosoma] = padres_cruzados[mejor_cromosoma];
end
if padres_cruzados[otro_cromosoma].funcion <
 poblacion[cromosoma_anterior].funcion then
    poblacion[cromosoma_anterior] = padres_cruzados[otro_cromosoma];
end
```

Finalmente se mostrará el pseudocódigo del proceso de evolución, donde se hará uso de todas las funciones y operadores anteriormente descritos. Por lo que me limitaré a indicar llamadas a ellos donde corresponda.

Algorithm 14: procesaAGG_AGE(poblacion, centroides, dataset, restricciones, num_caracteristicas, LAMBDA, tipo_cruce)

```
Result: void
Recorremos la poblacion aleatoriamente inicializada y actualizamos la funcion
 objetivo de cada cromosoma.
evaluaciones_funcion += 50;
while evaluaciones_funcion < 100000 do
    padres_seleccionados = SeleccionPadres();
    if tipo_cruce == 0 then
        padres_cruzados = funcion_cruce_uniforme(padres_seleccionados);
    else
        padres_cruzados = funcion_cruce_segmento_fijo(padres_seleccionados);
    end
    evaluaciones_funcion += llamadas_funcion_cruce;
    padres_mutados = MutacionPadresCruzados();
    evaluaciones_funcion += llamadas_funcion_mutacion;
    Ahora, si estamos ejecutando un AGE realizaremos un
     ReemplazamientoAGE(), en caso contrario un ReemplazamientoAGG()
end
```

3.2 Algoritmos Meméticos

Para empezar, tengo que indicar que mi implementación de los algoritmos meméticos tiene como base la de los algoritmos genéticos generacionales con cruce uniforme. A partir de esta implementación se le añade al final del algoritmo una llamada a una búsqueda local suave que, dependiendo del tipo de algoritmo memético, se aplica sobre diferentes cromosomas.

Por lo que primero voy a mostrar el pseudocódigo de esta búsqueda local suave (BLS), que es la que supone la diferencia fundamental entre un algoritmo genético y memético.

Algorithm 15: BLS(a_cambiar, numero_clusters, numero_fallos, llamadas, seed, centroides, dataset, num_caracteristicas, restricciones, LAMBDA)

Result: void
Barajamos un conjunto de indices que van desde 0 al numero de elementos del cromosoma a_cambiar
while *(mejora or fallos < numero_fallos) and (i < a_cambiar.elementos.size())* **do**
    mejora = false;
    Cromosoma aux = a_cambiar;
    for *Desde j = 0 hasta que j sea menor que el numero de clusters* **do**
        if *El cluster j != Elemento en la posicion indices[i] de aux* **then**
            aux[indices[i]] = j;
        end
        aux.funcion = calculaFuncionObjetivo(aux);
        llamadas_funcion_objetivo++;
        if *aux.funcion < a_cambiar.funcion* **then**
            a_cambiar = aux;
            mejora = true;
        end
    end
    if *!mejora* **then**
        fallos++;
    end
    i++;
end

Finalmente, la estructura del proceso de evolución de los diferentes algoritmos meméticos será la siguiente:

Algorithm 16: procesaAM(poblacion, centroides, dataset, restricciones, num_caracteristicas, LAMBDA, tipo_BL, semilla)

```
Result: void
Recorremos la poblacion aleatoriamente inicializada y actualizamos la funcion
 objetivo de cada cromosoma.
evaluaciones_funcion += 50;
while evaluaciones_funcion < 100000 do
    padres_seleccionados = SeleccionPadres();
    padres_cruzados = funcion_cruce_uniforme(padres_seleccionados);
    evaluaciones_funcion += llamadas_funcion_cruce;
    padres_mutados = MutacionPadresCruzados();
    evaluaciones_funcion += llamadas_funcion_mutacion;
    Ahora realizaremos un ReemplazamientoAGG()
    numero_generaciones++;
    if numero_generaciones == 10 then
        if tipo_BL == AM-1.0 then
            for Cada cromosoma i de la poblacion do
                BLS(poblacion[i]);
                evaluaciones_funcion += sumador_contador;
            end
        end
        if tipo_BL == AM-0.1 then
            for Cada cromosoma i de la poblacion do
                float prob = Randfloat(0,1);
                if prob < 0.1 then
                    BLS(poblacion[i]);
                    evaluaciones_funcion += sumador_contador;
                end
            end
        else
            Calculamos cual es el mejor cromosoma de la poblacion, ya que
             10*0.1 = 1
            BLS(poblacion[cromosoma_elegido]);
            evaluaciones_funcion += sumador_contador;
        end
        numero_generaciones = 0;
    end
end
```

4 Desarrollo de la práctica

4.1 Procedimiento seguido para desarrollar la práctica

La práctica se ha desarrollado en el lenguaje de programación C++ sin usar ningún tipo de framework de metaheurísticas. La programación de la práctica ha sido realizada al completo de forma manual, a excepción de los métodos del fichero "random.h", que pertenecen a la web de la asignatura.

Se han programado dos ficheros cpp, uno para cada algoritmo, donde en cada ejecución podremos ver los resultados de uno de los datasets de esta práctica. Esto lo he hecho así para poder reiniciar la semilla en cada ejecución y no se ejecuten todos los datasets de golpe.

La práctica se comenzó desde el fichero cpp de la búsqueda local (BL), donde se fueron añadiendo todas las funciones y métodos necesarios para poder implementar los algoritmos sobre poblaciones.

En ambos ficheros, antes de llamar al proceso de búsqueda, se cargan los vectores de datos y restricciones de todos los datasets con la información leída desde un flujo de entrada.

4.2 Manual de usuario

Para ejecutar los programas pertinentes es necesario usar la orden "make" para compilar los ficheros del makefile que adjunto en la práctica.

Para ejecutar el programa de los algoritmos genéticos, es necesario usar la siguiente sintaxis:

```
./AG <semilla> <porcentaje_restricciones> <Tipo_AG> <Tipo_Cruce> <Dataset>
```

La semilla puede ser cualquier número al azar

El porcentaje de restricciones es un número que puede ser o 10 o 20.

El tipo de AG puede ser 1 si se quiere ejecutar el AGG ó 2 si se quiere ejecutar el AGE.

El tipo de cruce puede ser 1 si se quiere ejecutar el cruce uniforme ó 2 si se quiere ejecutar el cruce de segmento fijo.

El tipo dataset puede ser 1 (Iris), 2 (Rand), 3 (Ecoli) o 4 (Newthyroid).

Para ejecutar el programa de los algoritmos meméticos, es necesario usar la siguiente sintaxis:

```
./AM <semilla> <porcentaje_restricciones> <Tipo_AM> <Dataset>
```

La semilla puede ser cualquier número al azar

El porcentaje de restricciones es un número que puede ser o 10 o 20.

El tipo de AM puede ser 1 si se quiere ejecutar el AM-1.0, 2 si se quiere ejecutar el AM-0.1, o 3 si se quiere ejecutar el AM-0.1mej.

El tipo dataset puede ser 1 (Iris), 2 (Rand), 3 (Ecoli) o 4 (Newthyroid).

5 Experimentos y análisis de resultados

Se han realizado un total de 280 ejecuciones divididas de la siguiente forma (cada ejecución corresponde a un solo dataset):

```
20 ejecuciones AGG-UN con 10% de restricciones
20 ejecuciones AGG-UN con 20% de restricciones
20 ejecuciones AGG-SF con 10% de restricciones
20 ejecuciones AGG-SF con 20% de restricciones
20 ejecuciones AGE-UN con 10% de restricciones
20 ejecuciones AGE-UN con 20% de restricciones
20 ejecuciones AGE-SF con 10% de restricciones
20 ejecuciones AGE-SF con 20% de restricciones
20 ejecuciones AM-1.0 con 10% de restricciones
20 ejecuciones AM-1.0 con 20% de restricciones
20 ejecuciones AM-0.1 con 10% de restricciones
20 ejecuciones AM-0.1 con 20% de restricciones
20 ejecuciones AM-0.1mej con 10% de restricciones
20 ejecuciones AM-0.1mej con 20% de restricciones
```

Cada una de las ejecuciones anteriormente descritas nos devuelve el valor de "Desviacion", "Infactibilidad", "Función objetivo" y "Tiempo de ejecución" de los tres datasets.

Las semillas que se han usado para las 5 ejecuciones son las siguientes:

- 545
- 650
- 17
- 1010
- 1234

Se utiliza la misma semilla para la misma ejecución de ambos algoritmos, es decir, semilla 545 para la ejecución 1 tanto de AG como de AM. Esto se hace así para que se puedan comparar los resultados.

5.1 Tablas de resultados y análisis de las mismas

Resultados obtenidos por el algoritmo AGG-UN en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,15	21,06	139,00	24,78	123,22	0,72	0,00	0,72	11,24	13,55	15,00	14,00	33,07
Ejecución 2: Seed 650	0,67	0,00	0,67	18,42	21,44	110,00	24,39	123,18	0,72	0,00	0,72	11,51	13,55	15,00	14,00	33,77
Ejecución 3: Seed 17	0,67	0,00	0,67	18,83	21,62	120,00	24,84	123,75	0,72	0,00	0,72	11,62	13,55	15,00	14,00	34,31
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,00	21,93	69,00	23,78	123,50	0,72	0,00	0,72	11,52	13,55	15,00	14,00	33,48
Ejecuión 5: Seed 1234	0,67	0,00	0,67	18,84	22,31	94,00	24,83	121,20	0,72	0,00	0,72	11,56	13,55	15,00	14,00	34,46
Media	0,67	0,00	0,67	18,85	21,67	106,40	24,52	122,97	0,72	0,00	0,72	11,49	13,55	15,00	14,00	33,82

Resultados obtenidos por el algoritmo AGG-UN en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,20	21,55	221,00	24,52	127,65	0,72	0,00	0,72	12,24	13,55	41,00	14,18	35,28
Ejecución 2: Seed 650	0,67	0,00	0,67	19,11	22,78	222,00	25,76	128,26	0,72	0,00	0,72	12,07	13,55	41,00	14,18	34,66
Ejecución 3: Seed 17	0,67	0,00	0,67	19,30	21,95	162,00	24,12	125,28	0,72	0,00	0,72	12,08	13,55	41,00	14,18	35,44
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,05	21,78	177,00	24,16	127,33	0,72	0,00	0,72	11,90	13,55	41,00	14,18	34,45
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,09	24,87	161,00	27,03	124,36	0,72	0,00	0,72	12,22	13,55	41,00	14,18	34,52
Media	0,67	0,00	0,67	19,15	22,59	188,60	25,12	126,58	0,72	0,00	0,72	12,10	13,55	41,00	14,18	34,87

Resultados obtenidos por el algoritmo AGG-SF en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,17	21,26	125,00	24,61	120,98	0,72	0,00	0,72	11,29	13,55	15,00	14,00	33,14
Ejecución 2: Seed 650	0,67	0,00	0,67	19,20	21,73	91,00	24,17	122,19	0,72	0,00	0,72	11,56	10,88	100,00	13,88	33,58
Ejecución 3: Seed 17	0,67	0,00	0,67	18,62	22,99	118,00	26,15	123,21	0,72	0,00	0,72	11,38	10,89	102,00	13,95	33,31
Ejecución 4: Seed 1010	0,67	0,00	0,67	18,88	23,05	165,00	27,48	123,96	0,72	0,00	0,72	11,71	10,82	118,00	14,37	34,00
Ejecuión 5: Seed 1234	0,67	0,00	0,67	18,97	24,12	172,00	28,74	120,57	0,72	0,00	0,72	11,38	10,82	105,00	13,97	33,32
Media	0,67	0,00	0,67	18,97	22,63	134,20	26,23	122,18	0,72	0,00	0,72	11,46	11,39	88,00	14,03	33,47

Resultados obtenidos por el algoritmo AGG-SF en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,49	22,30	206,00	25,07	127,25	0,72	0,00	0,72	12,27	13,55	41,00	14,18	34,82
Ejecución 2: Seed 650	0,67	0,00	0,67	19,53	21,70	251,00	25,07	128,28	0,72	0,00	0,72	12,32	10,81	265,00	14,88	35,14
Ejecución 3: Seed 17	0,67	0,00	0,67	19,59	21,95	200,00	24,63	127,68	0,72	0,00	0,72	12,29	10,82	255,00	14,73	35,82
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,07	21,75	285,00	25,57	127,58	0,72	0,00	0,72	12,35	13,55	41,00	14,18	35,70
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,03	24,03	244,00	27,30	125,50	0,72	0,00	0,72	12,12	13,55	41,00	14,18	35,72
Media	0,67	0,00	0,67	19,34	22,35	237,20	25,53	127,26	0,72	0,00	0,72	12,27	12,46	128,60	14,43	35,44

Resultados obtenidos por el algoritmo AGE-UN en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,34	22,43	68,00	24,25	127,88	0,72	0,00	0,72	12,12	13,55	15,00	14,00	34,99
Ejecución 2: Seed 650	0,67	0,00	0,67	19,50	21,30	103,00	24,07	127,49	0,72	0,00	0,72	11,96	13,55	15,00	14,00	35,99
Ejecución 3: Seed 17	0,67	0,00	0,67	19,53	21,95	115,00	25,03	124,35	0,72	0,00	0,72	12,00	10,71	95,00	13,56	35,57
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,63	21,47	89,00	23,86	126,55	0,72	0,00	0,72	12,08	10,88	98,00	13,82	35,99
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,00	22,08	81,00	24,25	124,02	0,72	0,00	0,72	11,69	10,82	109,00	14,09	34,95
Media	0,67	0,00	0,67	19,40	21,85	91,20	24,29	126,05	0,72	0,00	0,72	11,97	11,90	66,40	13,90	35,50

Resultados obtenidos por el algoritmo AGE-UN en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	20,11	21,98	177,00	24,36	131,68	0,72	0,00	0,72	12,73	10,81	261,00	14,82	37,26
Ejecución 2: Seed 650	0,67	0,00	0,67	19,95	21,35	282,00	25,13	131,39	0,72	0,00	0,72	12,78	13,55	41,00	14,18	37,63
Ejecución 3: Seed 17	0,67	0,00	0,67	20,15	22,25	161,00	24,41	130,63	0,72	0,00	0,72	12,57	13,55	41,00	14,18	37,38
Ejecución 4: Seed 1010	0,67	0,00	0,67	20,11	23,13	180,00	25,55	134,01	0,72	0,00	0,72	12,55	10,89	233,00	14,46	37,48
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,74	21,88	150,00	23,89	131,11	0,72	0,00	0,72	12,88	10,87	235,00	14,48	37,54
Media	0,67	0,00	0,67	20,01	22,12	190,00	24,67	131,77	0,72	0,00	0,72	12,70	11,93	162,20	14,42	37,46

Resultados obtenidos por el algoritmo AGE-SF en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,64	23,37	88,00	25,73	127,04	0,72	0,00	0,72	11,87	10,81	113,00	14,20	34,83
Ejecución 2: Seed 650	0,67	0,00	0,67	19,40	22,23	152,00	26,31	126,64	0,72	0,00	0,72	11,78	13,55	15,00	14,00	37,96
Ejecución 3: Seed 17	0,67	0,00	0,67	18,95	21,71	98,00	24,34	127,28	0,72	0,00	0,72	11,75	10,82	108,00	14,06	35,57
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,23	23,13	142,00	26,94	123,89	0,72	0,00	0,72	11,74	13,55	15,00	14,00	34,67
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,01	21,40	118,00	24,56	123,86	0,72	0,00	0,72	11,74	10,82	114,00	14,24	34,72
Media	0,67	0,00	0,67	19,25	22,37	119,60	25,58	125,74	0,72	0,00	0,72	11,78	11,91	73,00	14,10	35,55

Resultados obtenidos por el algoritmo AGE-SF en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	20,25	21,89	161,00	24,05	131,80	0,72	0,00	0,72	12,13	13,55	41,00	14,18	36,92
Ejecución 2: Seed 650	0,67	0,00	0,67	20,02	25,47	163,00	27,66	131,30	0,72	0,00	0,72	12,68	13,55	41,00	14,18	37,30
Ejecución 3: Seed 17	0,67	0,00	0,67	20,26	21,91	130,00	23,66	130,52	0,72	0,00	0,72	12,32	13,55	41,00	14,18	37,40
Ejecución 4: Seed 1010	0,67	0,00	0,67	21,46	21,97	218,00	24,89	128,39	0,72	0,00	0,72	12,56	13,55	41,00	14,18	37,03
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,91	21,80	223,00	24,79	131,37	0,72	0,00	0,72	12,15	10,90	235,00	14,50	36,45
Media	0,67	0,00	0,67	20,38	22,61	179,00	25,01	130,68	0,72	0,00	0,72	12,37	13,02	79,80	14,24	37,02

Resultados obtenidos por el algoritmo AM-1.0 en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	18,57	22,43	114,00	25,49	123,48	0,72	0,00	0,72	11,51	10,88	97,00	13,79	34,72
Ejecución 2: Seed 650	0,67	0,00	0,67	18,93	21,77	152,00	25,85	123,02	0,72	0,00	0,72	11,70	13,55	15,00	14,00	34,28
Ejecución 3: Seed 17	0,67	0,00	0,67	18,57	22,05	142,00	25,86	122,38	0,72	0,00	0,72	11,61	13,55	15,00	14,00	34,42
Ejecución 4: Seed 1010	0,67	0,00	0,67	18,75	22,61	157,00	26,82	121,07	0,72	0,00	0,72	11,53	10,90	99,00	13,87	34,09
Ejecuión 5: Seed 1234	0,67	0,00	0,67	18,61	21,66	171,00	26,24	121,70	0,72	0,00	0,72	11,69	10,90	97,00	13,81	34,48
Media	0,67	0,00	0,67	18,69	22,10	147,20	26,05	122,33	0,72	0,00	0,72	11,61	11,95	64,60	13,89	34,40

Resultados obtenidos por el algoritmo AM-1.0 en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,62	22,42	261,00	25,92	127,29	0,72	0,00	0,72	12,20	13,55	41,00	14,18	36,17
Ejecución 2: Seed 650	0,67	0,00	0,67	19,16	21,86	257,00	25,30	127,57	0,72	0,00	0,72	12,52	13,55	41,00	14,18	36,57
Ejecución 3: Seed 17	0,67	0,00	0,67	20,66	21,03	275,00	24,72	127,45	0,72	0,00	0,72	12,39	13,55	41,00	14,18	35,95
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,97	24,42	166,00	26,64	127,62	0,72	0,00	0,72	12,61	10,89	236,00	14,51	36,36
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,67	23,69	222,00	26,67	124,95	0,72	0,00	0,72	12,61	13,55	41,00	14,18	35,99
Media	0,67	0,00	0,67	19,82	22,68	236,20	25,85	126,98	0,72	0,00	0,72	12,47	13,02	80,00	14,24	36,21

Resultados obtenidos por el algoritmo AM-0.1 en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	18,95	22,06	72,00	23,99	125,32	0,72	0,00	0,72	11,80	10,87	96,00	13,76	34,86
Ejecución 2: Seed 650	0,67	0,00	0,67	19,25	22,21	49,00	23,52	123,63	0,72	0,00	0,72	11,86	13,55	15,00	14,00	34,68
Ejecución 3: Seed 17	0,67	0,00	0,67	18,92	22,39	71,00	24,29	121,93	0,72	0,00	0,72	11,89	13,55	15,00	14,00	34,55
Ejecución 4: Seed 1010	0,67	0,00	0,67	18,53	21,39	91,00	23,83	124,05	0,72	0,00	0,72	11,61	10,82	118,00	14,36	33,62
Ejecuión 5: Seed 1234	0,67	0,00	0,67	18,70	22,25	54,00	23,70	122,17	0,72	0,00	0,72	11,62	10,88	100,00	13,88	33,77
Media	0,67	0,00	0,67	18,87	22,06	67,40	23,86	123,42	0,72	0,00	0,72	11,75	11,93	68,80	14,00	34,30

Resultados obtenidos por el algoritmo AM-0.1 en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,72	21,95	165,00	24,16	125,93	0,72	0,00	0,72	12,56	13,55	41,00	14,18	35,92
Ejecución 2: Seed 650	0,67	0,00	0,67	19,30	21,75	121,00	23,37	128,45	0,72	0,00	0,72	12,75	13,55	41,00	14,18	36,41
Ejecución 3: Seed 17	0,67	0,00	0,67	19,78	22,06	159,00	24,19	126,13	0,72	0,00	0,72	12,50	13,55	41,00	14,18	36,21
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,72	21,91	230,00	25,00	126,08	0,72	0,00	0,72	12,51	13,55	41,00	14,18	36,86
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,75	22,18	192,00	24,76	125,85	0,72	0,00	0,72	12,55	13,55	41,00	14,18	35,89
Media	0,67	0,00	0,67	19,65	21,97	173,40	24,30	126,49	0,72	0,00	0,72	12,57	13,55	41,00	14,18	36,26

Resultados obtenidos por el algoritmo AM-0.1mej en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,02	21,56	83,00	23,79	124,97	0,72	0,00	0,72	11,79	10,87	96,00	13,76	35,09
Ejecución 2: Seed 650	0,67	0,00	0,67	18,77	21,72	84,00	23,98	124,87	0,72	0,00	0,72	12,03	13,55	15,00	14,00	33,90
Ejecución 3: Seed 17	0,67	0,00	0,67	18,99	21,26	84,00	23,52	122,51	0,72	0,00	0,72	12,04	13,55	15,00	14,00	34,88
Ejecución 4: Seed 1010	0,67	0,00	0,67	18,70	22,31	61,00	23,95	124,60	0,72	0,00	0,72	11,88	13,55	15,00	14,00	34,60
Ejecuión 5: Seed 1234	0,67	0,00	0,67	18,58	22,41	67,00	24,21	122,63	0,72	0,00	0,72	11,67	10,82	108,00	14,06	34,97
Media	0,67	0,00	0,67	18,81	21,85	75,80	23,89	123,91	0,72	0,00	0,72	11,88	12,47	49,80	13,96	34,69

Resultados obtenidos por el algoritmo AM-0.1mej en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)	*Tasa_C*	*Tasa_inf*	*Agr.*	T(s)
Ejecución 1: Seed 545	0,67	0,00	0,67	19,34	21,89	130,00	23,64	126,17	0,72	0,00	0,72	12,30	13,55	41,00	14,18	35,95
Ejecución 2: Seed 650	0,67	0,00	0,67	19,58	21,74	152,00	23,77	126,06	0,72	0,00	0,72	12,38	13,55	41,00	14,18	35,82
Ejecución 3: Seed 17	0,67	0,00	0,67	19,79	21,46	221,00	24,42	129,07	0,72	0,00	0,72	12,68	13,55	41,00	14,18	35,26
Ejecución 4: Seed 1010	0,67	0,00	0,67	19,74	22,16	157,00	24,27	125,72	0,72	0,00	0,72	12,52	10,90	235,00	14,50	35,99
Ejecuión 5: Seed 1234	0,67	0,00	0,67	19,25	21,93	146,00	23,89	128,76	0,72	0,00	0,72	12,36	13,55	41,00	14,18	36,42
Media	0,67	0,00	0,67	19,54	21,83	161,20	24,00	127,15	0,72	0,00	0,72	12,45	13,02	79,80	14,24	35,89

Resultados medios finales en el PAR con un 10 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0,73	24,80	0,89	0,31	34,58	23,60	35,21	5,32	0,76	0,00	0,76	0,24	15,65	20,00	16,25	0,55
BL	0,67	4,00	0,67	1,38	23,24	98,20	25,87	41,01	0,76	0,00	0,76	0,81	11,48	108,00	14,72	3,36
AGG-UN	0,67	0,00	0,67	18,85	21,67	106,40	24,52	122,97	0,72	0,00	0,72	11,49	13,55	15,00	14,00	33,82
AGG-SF	0,67	0,00	0,67	18,97	22,63	134,20	26,23	122,18	0,72	0,00	0,72	11,46	11,39	88,00	14,03	33,47
AGE-UN	0,67	0,00	0,67	19,40	21,85	91,20	24,29	126,05	0,72	0,00	0,72	11,97	11,90	66,40	13,90	35,50
AGE-SF	0,67	0,00	0,67	19,25	22,37	119,60	25,58	125,74	0,72	0,00	0,72	11,78	11,91	73,00	14,10	35,55
AM-(10,1.0)	0,67	0,00	0,67	18,69	22,10	147,20	26,05	122,33	0,72	0,00	0,72	11,61	11,95	64,60	13,89	34,40
AM-(10,0.1)	0,67	0,00	0,67	18,87	22,06	67,40	23,86	123,42	0,72	0,00	0,72	11,75	11,93	68,80	14,00	34,30
AM-(10,0.1mej)	0,67	0,00	0,67	18,81	21,85	75,80	23,89	123,91	0,72	0,00	0,72	11,88	12,47	49,80	13,96	34,69

Resultados medios finales en el PAR con un 20 por ciento de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0,67	0,00	0,67	0,28	29,72	0,00	29,72	4,00	0,76	0,00	0,76	0,12	14,29	0,00	14,29	0,49
BL	0,67	0,00	0,67	1,07	23,79	148,60	25,78	48,13	0,76	0,00	0,76	0,78	13,57	41,00	14,20	2,91
AGG-UN	0,67	0,00	0,67	19,15	22,59	188,60	25,12	126,58	0,72	0,00	0,72	12,10	13,55	41,00	14,18	34,87
AGG-SF	0,67	0,00	0,67	19,34	22,35	237,20	25,53	127,26	0,72	0,00	0,72	12,27	12,46	128,60	14,43	35,44
AGE-UN	0,67	0,00	0,67	20,01	22,12	190,00	24,67	131,77	0,72	0,00	0,72	12,70	11,93	162,20	14,42	37,46
AGE-SF	0,67	0,00	0,67	20,38	22,61	179,00	25,01	130,68	0,72	0,00	0,72	12,37	13,02	79,80	14,24	37,02
AM-(10,1.0)	0,67	0,00	0,67	19,82	22,68	236,20	25,85	126,98	0,72	0,00	0,72	12,47	13,02	80,00	14,24	36,21
AM-(10,0.1)	0,67	0,00	0,67	19,65	21,97	173,40	24,30	126,49	0,72	0,00	0,72	12,57	13,55	41,00	14,18	36,26
AM-(10,0.1mej)	0,67	0,00	0,67	19,54	21,83	161,20	24,00	127,15	0,72	0,00	0,72	12,45	13,02	79,80	14,24	35,89

Vamos a realizar un análisis de los diferentes resultados de los algoritmos de la práctica observando las tablas de resultados medios finales. Para realizar el análisis nos centraremos en el dataset Ecoli, ya que en mi opinión al haber tanta diversidad de clusters los resultados son más variables que en el resto de datasets.

En general me he fijado que la desviación de los algoritmos genéticos mejora muy

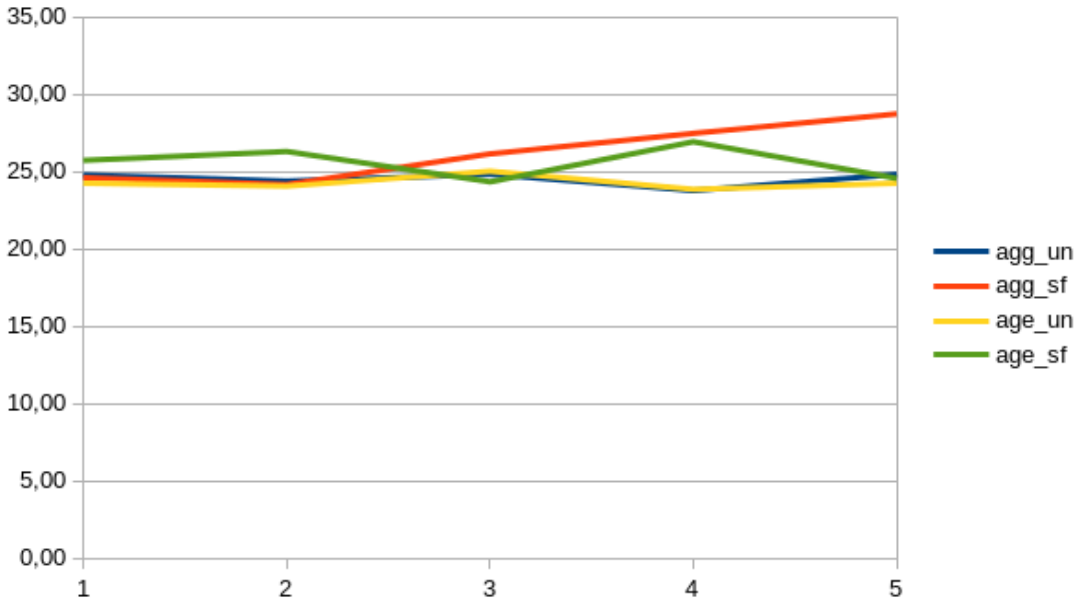
poco con respecto a la búsqueda local, aunque ese poco indicaría que los algoritmos genéticos son más íntegros. Y, en referencia a la infactibilidad, los valores que obtengo de los algoritmos genéticos son muy variables entre las diferentes versiones. A esto último si que le veo sentido ya que depende mucho de la semilla que usemos en cada momento, porque estos algoritmos tienen una alta dependencia de valores aleatorios.

Comentar también que el no poder identificar grandes diferencias entre los algoritmos se debe a que estamos usando muy pocos datasets. En investigación se suelen usar unos 50 data sets diferentes y se suele hacer un análisis estadístico bayesiano para detectar las diferencias.

Y en referencia a los algoritmos meméticos, las versiones 1.0 y 0.1 no mejoran demasiado a los genéticos. Incluso pueden llegar a empeorar las soluciones anteriores. Yo diría que esto ocurre porque la búsqueda local suave que estamos aplicando, se centra más en explorar que en explotar óptimos. Por lo que al aplicar BLS a tantos cromosomas nos alejamos de la solución local y empeoramos un poco.

En cambio la versión 0.1mej si que mejora en todos los casos, ya que solo le estamos aplicando la BLS al mejor cromosoma. Hay que decir también que esta última versión es la única que está publicada en un paper científico, por lo que es normal que esté más optimizada que las demás.

A continuación se mostrará una gráfica para ilustrar las diferencias entre las diferentes versiones de los algoritmos genéticos, usando el dataset Ecoli con 10 porciento de restricciones como ejemplo.

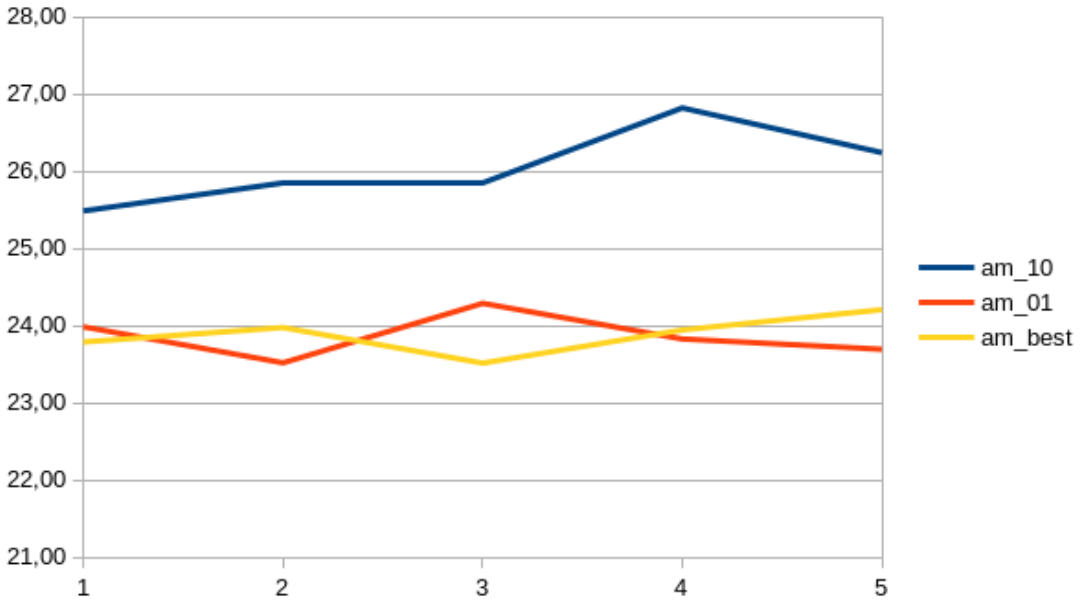


Al observar esta gráfica podemos realizar dos comparativas, una entre AGG y AGE. Y otra entre el cruce uniforme y el cruce por segmento fijo.

En referencia a los cruces, podemos ver que el cruce uniforme obtiene valores más bajos y por lo tanto mejores que el cruce por segmento fijo. Tambíen se puede ver que el cruce uniforme es menos variable en función de la semilla que el otro. Mi explicación para esto es que el cruce uniforme obtiene más información de los padres que el otro, ya que obtiene la mitad de valores de cada padre. Y esto se traduce en una mayor explotación que exploración, por lo que se mejora la función objetivo. En cambio, el cruce por segmento fijo obtiene partes de ambos padres de forma muy probabilística. Por lo que favorece la exploración a la explotación y puede mejorar o empeorar nuestra función.

En referencia a los tipos de algoritmos, igual no se aprecia tan bien, pero vemos que AGE tiene una tendencia a dar mejores resultados de función objetivo que AGG. Aunque la diferencia es mínima.

Ahora se mostrará una gráfica para ilustrar las diferencias entre las diferentes versiones de los algoritmos meméticos, usando el dataset Ecoli con 10 porciento de restricciones como ejemplo.

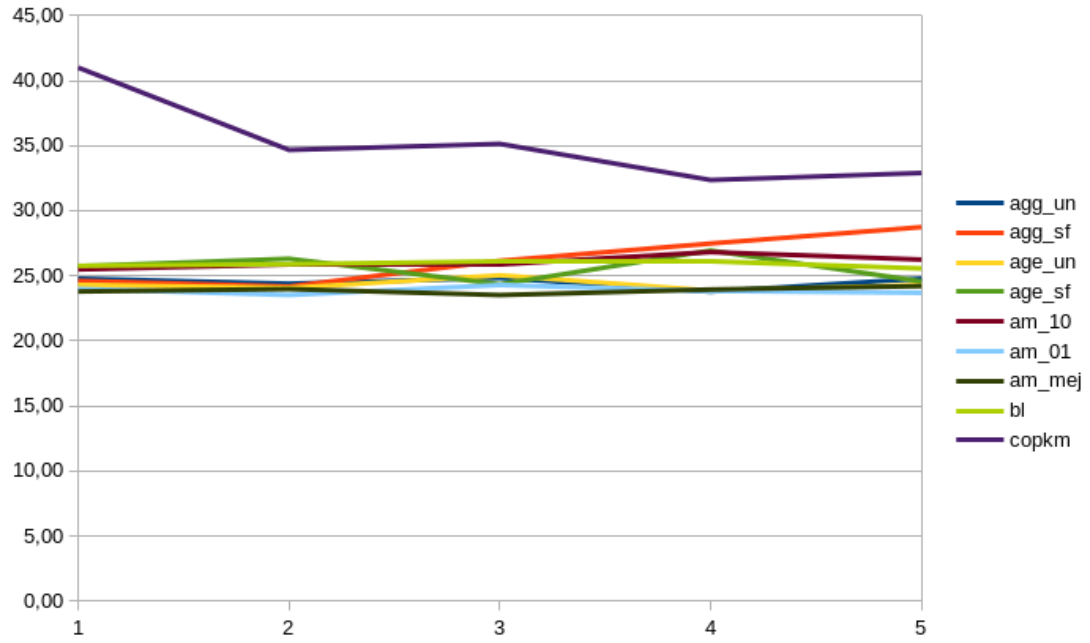


Al observar las gráficas vemos que efectivamente la versión AM-1.0 no sale rentable, ya que al aplicar la BLS a todos los cromosomas de la población, nos centramos más en

explorar que en explotar. Por lo que los resultados empeoran incluso a los genéticos.

En referencia a las otras dos versiones, vemos que dan valores de igual variabilidad, por lo que pueden ser buenas alternativas. Aunque de todas ellas, la única que está publicada en un paper científico es la versión AM-0.1mej, por lo que es la más optimizada.

Finalmente se mostrará una gráfica para realizar una comparativa entre los algoritmos de esta práctica y la anterior, usando el dataset Ecoli con 10 porciento de restricciones como ejemplo.



En la gráfica se puede observar que todas las metaheurísticas vistas hasta ahora mejoran con creces los resultados del algoritmo COPKM. En cambio, dentro de las propias metaheurísticas no vemos claras diferencias entre la BL de la práctica anterior y los nuevos algoritmos. Incluso vemos que la BL da mejores resultados que algunos de los algoritmos genéticos en ocasiones. Esto se puede razonar con lo antes explicado de que con solamente un dataset no se pueden realizar comparativas claras del todo, ya que en investigación se usan aproximadamente 50 datasets. Lo que si podemos asegurar, es que nuestra mejor metaheurística hasta el momento es el algoritmo memético en su versión 0.1mej.