Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Práctica 3: Técnicas de Búsqueda por trayectorias

Problema del Agrupamiento con Restricciones

Algoritmo de Enfriamiento Simulado (ES)
Algoritmo de Búsqueda Multiarranque Básica (BMB)
Algoritmo de Búsqueda Local Reiterada (ILS)
Algoritmo Híbrido ILS-ES

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

2019-2020

Índice

1 Descripción del problema **2**

2 Descripción de la parte común de los algoritmos **3**
2.1 Representación de los datos 3
2.2 Operadores comunes 4
2.2.1 Actualizar un centroide 4
2.2.2 Calcular la distancia máxima del conjunto 4
2.2.3 Calcular la desviación general 5
2.2.4 Calcular la función objetivo 6
2.2.5 Cálculo de infactibilidad 7
2.2.6 Comprobar si la asignación de centroides es válida 7
2.2.7 Generación de soluciones aleatorias 8

3 Implementación y desarrollo de cada algoritmo **10**
3.1 Algoritmo de Enfriamiento Simulado (ES) 10
3.2 Búsqueda Multiarranque Básica (BMB) 12
3.3 Algoritmo de Búsqueda Local Reiterada (ILS) 14

4 Desarrollo de la práctica **16**
4.1 Procedimiento seguido para desarrollar la práctica 16
4.2 Manual de usuario 16

5 Experimentos y análisis de resultados **17**
5.1 Tablas de resultados y análisis de las mismas 18

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.

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 los algoritmos.

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

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 i=0; hasta que i sea menor que 10 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
    inicializaciones_iris.push_back(indices_iris);
    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
    inicializaciones_rand.push_back(indices_rand);
    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
    inicializaciones_ecoli.push_back(indices_ecoli);
    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
    inicializaciones_newthyroid.push_back(indices_newthyroid);
end
```

Este algoritmo corresponde a la inicialización de soluciones aleatorias de la BMB. En el caso de ILS, el algoritmo sería el mismo pero sin estar contenido en un bucle for. Nos bastaría sacar una inicialización de indices aleatoria por dataset.

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 Algoritmo de Enfriamiento Simulado (ES)

En este apartado vamos a mostrar el esquema de cálculo de la temperatura inicial, el esquema de enfriamiento, y el propio esquema de búsqueda del Enfriamiento Simulado.

Para calcular la temperatura inicial se tomará que la probabilidad de aceptar una solución un 0.3 por 1 peor que la inicial es de 0.3.

Algorithm 8: getTempInicial(float coste_s0)

```
Result: float temperatura_inicial
numerador = 0.3*coste_s0;
denominador = -log(0.3);
return (numerador/denominador);
```

Vamos a usar 100000 evaluaciones en el siguiente pseudocódigo, el cual está adaptado a una sola ejecución de Enfriamiento Simulado. En el caso del algoritmo híbrido ILS-ES se usarán solamente 10000.

Algorithm 9: enfriaTemperatura(float temperatura, float t_inicial, float t_final, int tam_dataset)

```
Result: float temperatura_enfriada
maximos_vecinos = (10*tam_dataset);
M = 100000/maximos_vecinos;
Beta = (t_inicial - t_final) / (M*t_inicial*t_final);
temperatura_enfriada = temperatura / (1 + Beta*temperatura);
return temperatura_enfriada;
```

Finalmente se mostrará el pseudocódigo del proceso de búsqueda, 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 10: ES(data_set, indices, restrictions, centroides, problema, lambda, semilla)

```
Result: void
Inicializamos los centroides con los datos de "indices", los cuales hemos
 generado aleatoriamente;
mejor_valor =
 calculaFuncionObjetivo(indices,centroides,data_set,restrictions,lambda);
evaluaciones++;
actual = mejor_valor;
temperatura_inicial = getTempInicial(mejor_valor);
temperatura = temperatura_inicial;
while (evaluaciones < 100000) and (vecinos_aceptados > 0) and (temperatura
 > 0.001) do
    while vecinos_aceptados < aceptaciones_maximas and vecinos_generados <
     generaciones_maximas do
        Seleccionamos un elemento aleatorio y lo cambiamos por un cluster
         diferente al actual;
        vecinos_generados++;
        if Los nuevos indices no dejan ningun cluster vacio then
            Actualizamos los centroides con los nuevos indices;
            a_comparar = calculaFuncionObje-
             tivo(indices,centroides,data_set,restrictions,lambda);

            evaluaciones++;
            aleatorio = Randfloat(0,1);
            diferencia_funciones = a_comparar - actual;
            if diferencia_funciones < 0 or aleatorio <=
             (exp(-diferencia_funciones/temperatura)) then
                vecinos_aceptados++;
                actual = a_comparar;
                if a_comparar < mejor_valor then
                    Actualizamos el mejor resultado hasta el momento;
                end
            end
        end
    end
    temperatura =
     enfriaTemperatura(temperatura,temperatura_inicial,0.001,data_set.size());
end
```

3.2 Búsqueda Multiarranque Básica (BMB)

Antes de mostrar el pseudocódigo del proceso de búsqueda, voy a mostrar el pseudocódigo de la Búsqueda Local (BL) que se ha utilizado para implementar este algoritmo y la ILS. En ese pseudocódigo viene implícito el método de creación de la lista de candidatos, el de exploración del entorno, y el operador de generación de vecino.

Algorithm 11: BL(dataset, indices, restricciones, centroides, lambda, semilla)

```
Result: void
mejor_valor = calculaFuncionObjetivo;
Metemos en un vector de pairs de dos enteros todos los posibles cambios que se
 podrían hacer en el vector de indices.
Barajamos de forma aleatoria el vector de pairs usando la semilla.
for i=0; i < numero de elementos en el vector de pairs do
    vector_auxiliar = indices;
    vector_auxiliar[vector_pairs[i].first] = vector_pairs[i].second;
    contador++;
    if Los indices del vector auxiliar son correctos then
        podemos calcular la funcion objetivo
    end
    if Podemos calcular la funcion objetivo then
        Metemos en un vector auxiliar de centroides la información de nuestros
         centroides actuales por si es necesario restaurarlos.
        Rellenamos y actualizamos nuestros centroides con los indices del vector
         auxiliar.
        if FuncionObjetivo(nuevos_centroides) < mejor_valor then
            mejor_valor = FuncionObjetivo(nuevos_centroides);
            indices = vector_auxiliar;
            Reiniciamos el for principal usando un booleano o poniendo su indice
             i a 0.
            Volvemos a rellenar el vector de pairs de dos enteros con los nuevos
             posibles cambios que se podrían hacer en el vector de índices y lo
             barajamos, usando de semilla el contador actual.
        else
            Restauramos el valor de nuestros centroides, que habiamos guardado
             en un vector auxiliar.
        end
    end
    if contador == 10000 then
        Salimos del bucle for más externo y nos quedamos con la mejor solución
         que hayamos sacado hasta ese momento.
    end
end
```

Una vez mostrado el esquema de la BL, pasaré a mostrar el pseudocódigo del proceso de búsqueda de la BMB.

Algorithm 12: BMB(inicializaciones, data_set, restrictions, centroides, problema, lambda, semilla)

```
Result: void
for int i=0; i < inicializaciones.size(); i++ do
    Actualizamos los centroides con el valor de los indices de la inicializacion i;
    BL(data_set,inicializaciones[i],restrictions,centroides,problema,lambda,semilla,inf,fun,desv);
    if fun < mejor_funcion then
        Actualizamos la mejor solucion encontrada hasta el momento;
    end
end
```

3.3 Algoritmo de Búsqueda Local Reiterada (ILS)

Antes de mostrar el pseudocódigo del proceso de búsqueda voy a mostrar el esquema del operador de mutación utilizado. La Búsqueda Local (BL) que se utiliza en este apartado es la misma que hay descrita en la subsección anterior.

Algorithm 13: mutaIndices(indices, num_centroides)

```
Result: vector de indices mutado
Creamos un vector de booleanos "necesita_cambiar" con tantos datos como el
 tamaño del dataset y lo inicializamos a false;
if ((inicio_segmento + v) mod indices.size()-1) > inicio_segmento then
    for j=inicio_segmento; j < ((inicio_segmento + v) mod indices.size()-1);
     j++ do
        necesita_cambiar[j] = true;
    end
else
    for j=inicio_segmento; j < indices.size()-1; j++ do
        necesita_cambiar[j] = true;
    end
    for j=0; j < ((inicio_segmento + v) mod indices.size()-1); j++ do
        necesita_cambiar[j] = true;
    end
end
Recorremos el vector de booleanos y cambiamos aleatoriamente todos los indices
 marcados como true, comprobando que no se quede ningun cluster vacio;
mutado = indices;
return mutado;
```

A continuación pasaré a mostrar el algoritmo de búsqueda del ILS, el cual llama a este operador de mutación y a la BL.

Algorithm 14: ILS(data_set, indices, restrictions, centroides, problema, lambda, semilla)

Result: void
Actualizamos los centroides con los valores de los indices inicializados aleatoriamente;
S = BL (data_set,indices,restrictions,centroides,problema,lambda,semilla);
for *(int i=0; i < 9; i++* **do**
    S_prima = mutaIndices(S);
    S_doble_prima = BL(S_prima);
    if *funcion_S_doble_prima < funcion_S* **then**
        S = S_doble_prima;
    end
    if *funcion_S < mejor_funcion* **then**
        Actualizamos el valor de la mejor solucion hasta el momento;
    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 cuatro ficheros cpp, uno para cada algoritmo, donde en cada ejecución podremos ver los resultados de los datasets de esta práctica.

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 nuevos algoritmos.

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 cualquier programa de esta práctica, es necesario usar la siguiente sintaxis:

```
./Nombre <semilla> <porcentaje_restricciones>
```

El nombre puede ser "BMB" o "ILS" o "ILS-ES" o "ES".

La semilla puede ser cualquier número al azar

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

5 Experimentos y análisis de resultados

Se han realizado un total de 40 ejecuciones divididas de la siguiente forma:

```
5 ejecuciones BMB con 10% de restricciones.
5 ejecuciones BMB con 20% de restricciones.
5 ejecuciones ES con 10% de restricciones.
5 ejecuciones ES con 20% de restricciones.
5 ejecuciones ILS con 10% de restricciones.
5 ejecuciones ILS con 20% de restricciones.
5 ejecuciones ILS-ES con 10% de restricciones.
5 ejecuciones ILS-ES 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 cuatro 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 todos los algoritmos, es decir, semilla 545 para la ejecución 1 tanto de BMB como de ILS como de ES como de ILS-ES. 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 BMB 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	2,33	24,97	117,00	28,11	43,99	0,72	0,00	0,72	1,29	13,54	15,00	13,99	6,87
Ejecución 2: Seed 650	0,67	0,00	0,67	2,09	25,95	132,00	29,49	44,35	0,72	0,00	0,72	1,24	13,53	15,00	13,98	6,74
Ejecución 3: Seed 17	0,67	0,00	0,67	1,94	24,74	138,00	28,44	43,23	0,72	0,00	0,72	1,34	13,54	15,00	13,99	8,16
Ejecución 4: Seed 1010	0,67	0,00	0,67	2,08	24,98	162,00	29,32	43,80	0,72	0,00	0,72	1,23	13,54	15,00	13,99	7,04
Ejecuión 5: Seed 1234	0,67	0,00	0,67	2,03	23,90	151,00	27,95	43,05	0,72	0,00	0,72	1,20	13,53	15,00	13,99	6,29
Media	0,67	0,00	0,67	2,09	24,91	140,00	28,66	43,68	0,72	0,00	0,72	1,26	13,54	15,00	13,99	7,02

Resultados obtenidos por el algoritmo BMB 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	2,11	24,11	332,00	28,57	53,43	0,72	0,00	0,72	1,35	13,54	41,00	14,17	7,02
Ejecución 2: Seed 650	0,67	0,00	0,67	2,05	24,84	223,00	27,83	52,07	0,72	0,00	0,72	1,24	13,53	41,00	14,16	7,01
Ejecución 3: Seed 17	0,67	0,00	0,67	2,08	24,76	115,00	26,30	51,85	0,72	0,00	0,72	1,34	13,54	41,00	14,17	6,31
Ejecución 4: Seed 1010	0,67	0,00	0,67	2,14	25,13	213,00	27,99	53,43	0,72	0,00	0,72	1,29	13,54	41,00	14,17	7,08
Ejecuión 5: Seed 1234	0,67	0,00	0,67	1,93	24,93	270,00	28,55	53,22	0,72	0,00	0,72	1,32	13,54	41,00	14,17	6,48
Media	0,67	0,00	0,67	2,06	24,75	230,60	27,85	52,80	0,72	0,00	0,72	1,31	13,54	41,00	14,17	6,78

Resultados obtenidos por el algoritmo ES 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	1,91	21,67	68,00	23,49	43,47	0,72	0,00	0,72	0,88	10,82	109,00	14,09	2,46
Ejecución 2: Seed 650	0,67	0,00	0,67	1,86	22,16	66,00	23,93	79,65	0,72	0,00	0,72	0,61	13,55	15,00	14,00	3,08
Ejecución 3: Seed 17	0,67	0,00	0,67	1,56	22,04	77,00	24,11	38,98	0,72	0,00	0,72	0,81	13,55	15,00	14,00	3,14
Ejecución 4: Seed 1010	0,67	0,00	0,67	1,55	22,39	74,00	24,37	29,71	0,72	0,00	0,72	1,04	13,55	15,00	14,00	6,17
Ejecuión 5: Seed 1234	0,67	0,00	0,67	1,10	21,31	119,00	24,50	54,09	0,72	0,00	0,72	0,65	10,82	112,00	14,18	2,44
Media	0,67	0,00	0,67	1,60	21,91	80,80	24,08	49,18	0,72	0,00	0,72	0,80	12,46	53,20	14,05	3,46

Resultados obtenidos por el algoritmo ES 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	1,25	21,83	154,00	23,90	31,13	0,72	0,00	0,72	0,57	13,55	41,00	14,18	2,58
Ejecución 2: Seed 650	0,67	0,00	0,67	1,37	21,87	163,00	24,05	51,24	0,72	0,00	0,72	0,44	10,89	233,00	14,47	1,81
Ejecución 3: Seed 17	0,67	0,00	0,67	1,35	21,90	141,00	23,79	40,01	0,72	0,00	0,72	1,02	13,55	41,00	14,18	2,64
Ejecución 4: Seed 1010	0,67	0,00	0,67	1,01	21,85	143,00	23,77	27,05	0,72	0,00	0,72	0,70	13,55	41,00	14,18	2,59
Ejecuión 5: Seed 1234	0,67	0,00	0,67	0,79	19,57	261,00	23,07	69,47	0,72	0,00	0,72	1,00	10,88	230,00	14,41	2,54
Media	0,67	0,00	0,67	1,15	21,40	172,40	23,72	43,78	0,72	0,00	0,72	0,75	12,48	117,20	14,28	2,43

Resultados obtenidos por el algoritmo ILS 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	0,94	24,78	141,00	28,56	36,45	0,72	0,00	0,72	0,37	11,18	113,00	14,57	2,60
Ejecución 2: Seed 650	0,67	0,00	0,67	1,13	25,28	224,00	31,29	38,73	0,72	0,00	0,72	0,42	11,14	115,00	14,59	3,02
Ejecución 3: Seed 17	0,67	0,00	0,67	1,03	23,03	193,00	28,21	39,99	0,72	0,00	0,72	0,37	13,54	15,00	13,99	3,11
Ejecución 4: Seed 1010	0,67	0,00	0,67	1,06	26,20	231,00	32,40	40,59	0,72	0,00	0,72	0,31	11,19	97,00	14,10	3,17
Ejecución 5: Seed 1234	0,67	0,00	0,67	0,90	23,91	201,00	29,30	38,34	0,72	0,00	0,72	0,36	11,25	97,00	14,16	3,11
Media	0,67	0,00	0,67	1,01	24,64	198,00	29,95	38,82	0,72	0,00	0,72	0,37	11,66	87,40	14,28	3,00

Resultados obtenidos por el algoritmo ILS 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	1,07	24,44	519,00	31,41	40,86	0,72	0,00	0,72	0,37	13,05	187,00	15,92	2,93
Ejecución 2: Seed 650	0,67	0,00	0,67	1,05	25,74	339,00	30,29	47,53	0,72	0,00	0,72	0,44	13,53	41,00	14,16	3,47
Ejecución 3: Seed 17	0,67	0,00	0,67	1,14	24,79	415,00	30,36	39,70	0,72	0,00	0,72	0,41	13,53	41,00	14,16	3,99
Ejecución 4: Seed 1010	0,67	0,00	0,67	1,14	26,42	509,00	33,25	43,37	0,72	0,00	0,72	0,34	13,54	41,00	14,16	3,52
Ejecución 5: Seed 1234	0,67	0,00	0,67	1,12	24,49	309,00	28,64	45,08	0,72	0,00	0,72	0,41	13,54	41,00	14,17	3,44
Media	0,67	0,00	0,67	1,10	25,18	418,20	30,79	43,31	0,72	0,00	0,72	0,40	13,44	70,20	14,51	3,47

Resultados obtenidos por el algoritmo ILS-ES 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	7,85	28,26	273,00	35,59	73,13	0,72	0,00	0,72	4,60	10,88	97,00	13,79	19,51
Ejecución 2: Seed 650	0,67	0,00	0,67	7,91	28,93	281,00	36,47	71,16	0,72	0,00	0,72	4,40	13,55	15,00	14,00	19,66
Ejecución 3: Seed 17	0,67	0,00	0,67	7,95	25,76	324,00	34,45	64,91	0,72	0,00	0,72	4,67	10,87	96,00	13,76	20,25
Ejecución 4: Seed 1010	0,67	0,00	0,67	6,92	25,97	183,00	30,88	66,98	0,72	0,00	0,72	4,24	10,90	97,00	13,81	21,07
Ejecución 5: Seed 1234	0,67	0,00	0,67	7,56	25,15	219,00	31,03	69,47	0,72	0,00	0,72	4,31	10,88	97,00	13,79	21,75
Media	0,67	0,00	0,67	7,64	26,81	256,00	33,68	69,13	0,72	0,00	0,72	4,44	11,42	80,40	13,83	20,45

Resultados obtenidos por el algoritmo ILS-ES 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	7,84	26,23	702,00	35,65	68,75	0,72	0,00	0,72	4,74	13,55	41,00	14,18	22,80
Ejecución 2: Seed 650	0,67	0,00	0,67	7,39	23,98	611,00	32,18	70,44	0,72	0,00	0,72	4,64	13,55	41,00	14,18	22,60
Ejecución 3: Seed 17	0,67	0,00	0,67	7,96	25,21	534,00	32,37	74,40	0,72	0,00	0,72	4,54	13,55	41,00	14,18	21,08
Ejecución 4: Seed 1010	0,67	0,00	0,67	7,24	26,04	388,00	31,25	75,31	0,72	0,00	0,72	4,28	13,55	41,00	14,18	23,81
Ejecución 5: Seed 1234	0,67	0,00	0,67	7,34	26,58	563,00	34,13	73,91	0,72	0,00	0,72	4,65	13,55	41,00	14,18	20,89
Media	0,67	0,00	0,67	7,55	25,61	559,60	33,11	72,56	0,72	0,00	0,72	4,57	13,55	41,00	14,18	22,24

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
BMB	0,67	0,00	0,67	2,09	24,91	140,00	28,66	43,68	0,72	0,00	0,72	1,26	13,54	15,00	13,99	7,02
ES	0,67	0,00	0,67	1,60	21,91	80,80	24,08	49,18	0,72	0,00	0,72	0,80	12,46	53,20	14,05	3,46
ILS	0,67	0,00	0,67	1,01	24,64	198,00	29,95	38,82	0,72	0,00	0,72	0,37	11,66	87,40	14,28	3,00
ILS-ES	0,67	0,00	0,67	7,64	26,81	256,00	33,68	69,13	0,72	0,00	0,72	4,44	11,42	80,40	13,83	20,45

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
BMB	0,67	0,00	0,67	2,06	24,75	230,60	27,85	52,80	0,72	0,00	0,72	1,31	13,54	41,00	14,17	6,78
ES	0,67	0,00	0,67	1,15	21,40	172,40	23,72	43,78	0,72	0,00	0,72	0,75	12,48	117,20	14,28	2,43
ILS	0,67	0,00	0,67	1,10	25,18	418,20	30,79	43,31	0,72	0,00	0,72	0,40	13,44	70,20	14,51	3,47
ILS-ES	0,67	0,00	0,67	7,55	25,61	559,60	33,11	72,56	0,72	0,00	0,72	4,57	13,55	41,00	14,18	22,24

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.

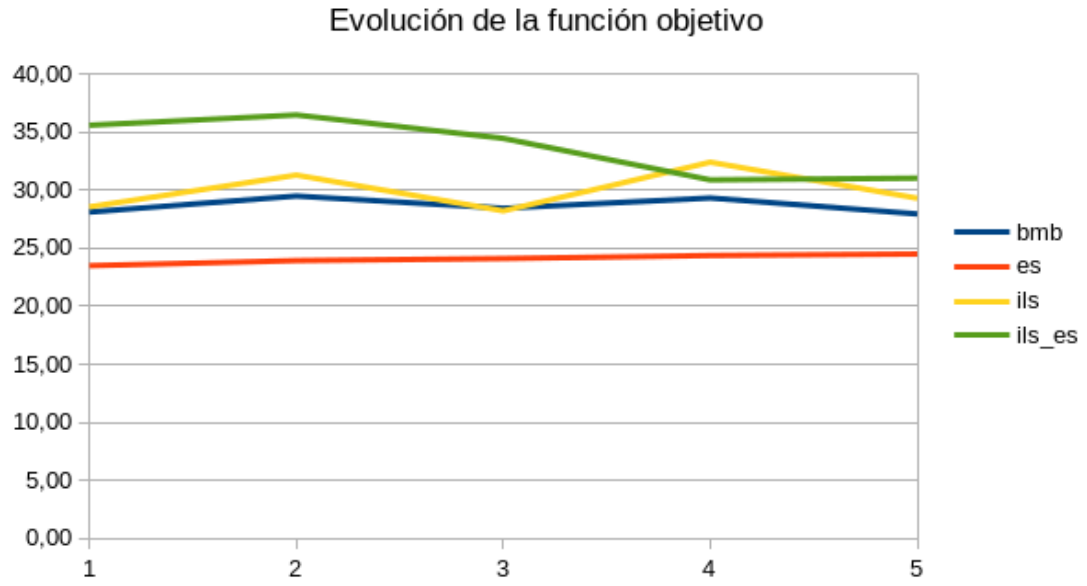
Podemos ver que el algoritmo que mejor se comporta de entre todos los mostrados es el enfriamiento simulado. Esto ocurre porque es el algoritmo que mantiene un mejor equilibrio intensificación/diversificación. Al empezar diversifica bastante porque la temperatura es alta y el criterio de Cauchy nos permite coger soluciones peores. Pero cuanto más se enfría, comienza a intensificar cada vez más, mejorando la solución con la que se había quedado hasta el momento.

Hay que decir también que los parámetros que se han elegido estaban adaptados a una sola ejecución de ES. Es por eso que nuestro algoritmo ILS-ES obtiene unos resultados tan mediocres. ILS-ES es un algoritmo el cual busca una intensificación fuerte tipo BL, que al usar ES enfría muy rápido por lo que aumenta la intensificación del algoritmo.

Tanto la ILS como el la BMB obtienen resultados mediocres. El caso de la BMB se debe a que diversifica demasiado al utilizar 10 inicializaciones aleatorias. Y el caso de la ILS se debe a que intensifica demasiado al usar solamente 10000 iteraciones de la BL y utilizar un operador de mutación tan potente. Si hubiésemos usado 100000 iteraciones seguramente los resultados serían mejores, pero sacrificaríamos demasiado tiempo de ejecución.

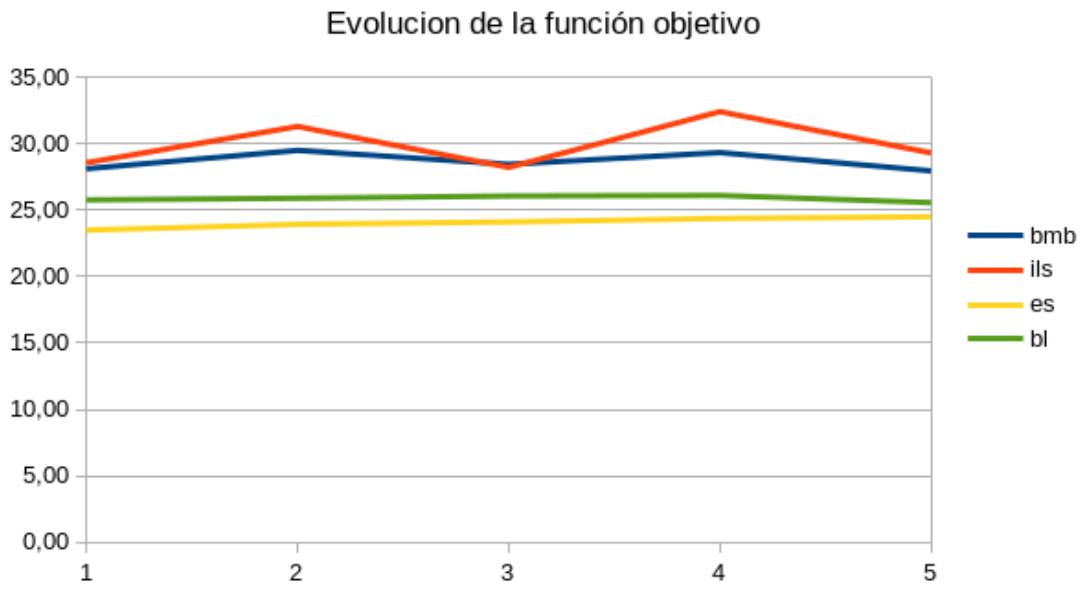
A continuación se mostrará una gráfica para ilustrar las diferencias entre las diferentes

versiones de los algoritmos, usando el dataset Ecoli con 10 porciento de restricciones como ejemplo.



Podemos ver en la gráfica que el enfriamiento simulado obtiene valores estables de la función objetivo para todas las ejecuciones. Mientras que las dos versiones de ILS varían mucho dependiendo de la inicialización que tomemos, ya que ILS se trata de un algoritmo que intensifica mucho como hemos dicho antes. Si intensificamos una solución que nos puede llevar a buen puerto, obtendremos una función objetivo mejor. En caso contrario, la curva de la función subirá como se puede ver en la gráfica.

Ahora vamos a comparar las técnicas basadas en trayectorias simples con las técnicas basadas en trayectorias múltiples. Para ello representaremos en una gráfica la evolución de la función objetivo de la BMB y la ILS (trayectorias múltiples); y la ES y la BL (trayectorias simples).



Podemos observar en la tabla que los algoritmos correspondientes a las trayectorias simples obtienen mejores resultados que las trayectorias compuestas. Sobre todo el enfriamiento simulado.