Práctica 2

Técnicas de Búsqueda basadas en Poblaciones

Problema del Agrupamiento con Restricciones (PAR)

Metaheurística GII 19-20
Grupo MH1 - Miércoles 17:30h

Cumbreras Torrente, Paula
49087324-B
paulacumbreras@correo.ugr.es

ugr | Universidad de Granada

"El reto principal de los científicos informáticos es no confundirse con la complejidad de su propia creación"

-E. W. Dijkstra

Índice de contenidos

Descripción del problema

El problema del agrupamiento con restricciones (PAR) consiste en un problema de agrupamiento o clustering al que se le ha añadido un conjunto de restricciones. El clustering es una técnica de aprendizaje no supervisada cuyo objetivo es agrupar los elementos que tengan alguna similitud entre ellos de un conjunto de datos en varios clusters, simplificando así la información de dichos datos.

Este problema suele ser aplicado en aplicaciones de minería de datos, aplicaciones Web, marketing, diagnóstico médico, análisis de ADN, biología computacional, análisis de plantas y animales, farmacología…

El problema elegido consiste en una generalización del ya mencionado. PAR es una técnica de aprendizaje semi-supervisada. Las restricciones expresan un requerimiento del usuario y describen propiedades que han de contener los grupos de elementos. El conjunto de restricciones que se aplicarán a nuestro problema pueden ser clasificadas en:

- Restricciones fuertes (hard): todas las restricciones de esta clase han de ser cumplidas para que una solución sea factible. Las restricciones fuertes tratadas en nuestro problema serán:
    - Todos los clusters deben contener al menos una instancia:
    - Cada instancia debe pertenecer a un solo cluster:
    - La unión de los clusters debe ser el conjunto de datos X:
- Restricciones débiles (soft): se busca que el número de restricciones violadas de esta clase sea minimizado. Las restricciones débiles a aplicar serán restricciones de instancia:
    - Must-Link: dos instancias dadas han de ser asignadas al mismo cluster.
    - Can't-Link: dos instancias dadas no podrán agruparse en el mismo cluster.

Se nos han proporcionado 4 conjuntos de datos para la realización de del problema:

- Iris: 150 datos con 4 propiedades a agrupar en 3 clases.
- Ecoli: 336 datos con 7 propiedades a agrupar en 8 clases.
- Rand: 150 datos con 2 propiedades a agrupar en 3 clases.
- Newthyroid: 215 datos con 5 propiedades a agrupar en 3 clases

Para cada conjunto de datos se estudiarán 2 instancias distintas de PAR a partir de los conjuntos de restricciones, también proporcionados, los cuales corresponderán al 10% y 20% del total de restricciones posibles.

Para esta práctica que estudiarán 4 algoritmos genéricos (2 estacionarios y 2 generacionales) y 3 algoritmos meméticos. Se compararán los resultados obtenidos con dichos algoritmos, evaluados en función de la desviación general, de la infactibilidad y del tiempo obtenido con diferentes semillas de aleatoriedad, con los resultados obtenidos con los algoritmos de la práctica anterior (Búsqueda Local y Greedy).

Descripción de la aplicación de los algoritmos empleados

El esquema de estructuras de datos empleado para el problema es el siguiente:

- Los elementos se almacenan en Nodos, estructuras formadas por un entero sin signo cluster donde se registrará el cluster al que pertenecen (por defecto -1), y un vector de double datos donde se almacenan las distintas propiedades del elemento.
- El conjunto de todos los Nodos se almacena en un vector de Nodos X en el orden de lectura.
- Las restricciones se almacenan en una matriz (vector de vector) de enteros M.
- Un cromosoma equivaldrá a una partición, consistente en un vector de tantos enteros sin signo como elementos tenga el conjunto seleccionado. En cada posición de este vector se almacenará el número del cluster al que ha sido asignado el elemento de X correspondiente a dicha posición.
- Una población es un vector de 50 cromosomas en caso de los algoritmos genéticos, o de 10 cromosomas en caso de los algoritmos meméticos.

Para permitir la reutilización del código de la práctica anterior, se ha añadido una función que permite adaptar un cromosoma a la estructura de datos anterior:

- Los Cluster quedan representados como una estructura compuesta por un vector de double donde se almacenará su centroide y por un vector de enteros sin signos donde se almacenará el índice en X de los distintos Nodos que lo compongan.
- El conjunto de todos los Clusters se almacena en un vector de Cluster C.

Conversión de cromosoma a antigua estructura de datos

Para todos las posiciones i del cromosoma
  asignar a la variable cluster del elemento X[i] el valor de cromosoma[i]
  añadir al vector datos del cluster C[cromosoma[i]] la posición i
Fin para
Actualizar centroides
Devolver C (X ha sido pasado por referencia)

Una vez se haya ejecutado el algoritmo seleccionado, aparecerán por pantalla una serie de evaluaciones de la partición obtenida:

- Tasa_C: Desviación general de la partición.

Desviación general (práctica 1)

Para todos los clusters
  calcular distancia media intra-cluster
  añadir el valor obtenido al valor de una variable desviación
Devolver desviación entre el número de clusters
{Almacenar distancias medias en un vector}

Distancia media intra-cluster (práctica 1)

```
Para cada elemento i del cluster
  para cada propiedad j del elemento
    añadir en variable[j] la distancia entre el dato j del elemento i y el dato j del
centroide del cluster
Para cada propiedad del vector variable
  añadir su valor a una variable distancia
Devolver raíz cuadrada de distancia entre el número de elementos del cluster
```

- Tasa_Inf: Infactibilidad o número de violaciones de restricciones

Infactibilidad

```
Para cada fila de M
  para cada columna de M
    si M[i][j] es 1 y los elementos i y j del cromosoma son distintos
      incrementar infactibilidad en 1
    si M[i][j] es -1 y los elementos i y j del cromosoma son el mismo
      incrementar infactibilidad en 1
Devolver infactibilidad
{Almacenar número total de restricciones en una variable}
```

- Agregado: Función objetivo o función a minimizar

Función objetivo

```
Calcular infactibilidad {número total de restricciones}
Calcular desviación {vector de distancias medias}
Para cada elemento del vector de distancias
  Buscar la mayor distancia
Lambda es la distancia máxima entre el número total de restricciones
Devolver desviación más infactibilidad por lambda
```

- Tiempo: Calculado desde el inicio del algoritmo hasta el final del mismo, excluyendo la lectura de ficheros e inicialización de datos. Devuelto en nanosegundos.

Como parámetro de entrada habrá que indicar si se desea que se muestre por pantalla el resultado de la partición C. En caso afirmativo se mostrarán los índices en X de los elementos almacenados en cada cluster; un ejemplo de salida sería:

```
Semilla = 8

Tasa_C|Tasa_i|Agreg|Tiempo
0.106822  0  0.106822  457676653

Cluster 0:
{ 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148
149 }
Cluster 1:
{ 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 0 2 }
Cluster 2:
{ 51 52 53 54 55 56 57 58 59 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 50 60 }
```

Aparte de los ya mencionados, otros operadores que han sido necesarios para la implementación de los algoritmos son:

- Cálculo de centroides: Asigna a cada cluster el vector promedio de las instancias de X que los componen

Calculo centroides (práctica 1)

Para cada cluster de la partición
  para cada elemento i del cluster
  para cada propiedad j del elemento
  añadir a variable [i] la propiedad j del elemento i
  para cada propiedad i del vector centroide
  asignar a variable [i] el valor que tenía dividido entre el número de elementos del cluster
  asignar el vector variable al centroide del cluster

- Operadores de cruce: Se han empleado los operadores de cruce uniforme y por segmento fijo, cruzan dos padres devolviendo un cromosoma hijo

Cruce uniforme

Generar un vector selección aleatorio de tamaño num_elementos/2
Para cada i de num_elementos
  si selección contiene i, añadir a hijo el elemento i del primer padre
  si no, añadir a hijo el elemento i del segundo padre
Fin para
Modificar hijo para que ningún cluster esté vacío (no vacío)
Devolver hijo

Cruce por segmento fijo

Obtener índice inicio aleatoriamente
Obtener tamaño del segmento aleatoriamente
Añadir a vector segmento las posiciones comprendidas entre inicio e inicio+tam_segmento módulo num_elementos
Para cada i de num_elementos
  Si segemento contiene i, asignar a hijos[i] el elemento i del primer padre
  Si no, añadir i a vector restantes
Fin para
Generar vector selección aleatorio a partir de restantes, de la mitad de tamaño de restantes
Para cada i de restantes
  si selección contiene i, asignar a hijo[i] el elemento i del primer padre
  si no, asignar a hijo[i] el elemento i del segundo padre
Fin para
Modificar hijo para que ningún cluster esté vacío (no vacío)
Devolver hijo

- Operador para evitar que un cromosoma tenga algún cluster vacío: Realiza modificaciones aleatorias hasta que esta restricción sea cumplida

No vacío

Generar vector check de tamaño num_clus
Para cada elemento i del cromosoma
  Aumentar check[i] en una unidad
Fin para
Mientras no se cumpla la restricción
  Para cada i de num_cluster

```
    si check[i] vale 0
       seleccionar aleatoriamente un cluster rand
       almacenar valor de cromosoma[rand] en aux
       modificar el valor de cromosma[rand] a i
       aumentar check[i] en una unidad
       reducir check[aux] en una unidad
       salir del para
    si  ningún elemento de check vale 0
       se cumple la restricción
  Fin para
Fin mientras
Devolver cromosoma
```

- Operador de mutación: Mutación uniforme, modifica un gen aleatoriamente

Mutación uniforme
Seleccionar posición aleatoriamente Seleccionar nuevo_cluster distinto a cromosoma[posición] aleatoriamente Asignar a cromosoma[posición] nuevo_cluster Modificar cromosoma para que ningún cluster esté vacío (no vacío) Devolver cromosoma

- Método de selección: Torneo binario, selecciona aleatoriamente dos cromosomas de una población y devuelve el mejor de ellos

Torneo binario
Generar aleatoriamente vector torneo con 2 cromosomas Almacenar función objetivo del primer cromosoma Almacenar función objetivo del segundo cromosoma Devolver cromosoma con mejor función objetivo

- Tirar dado: Empleado para probabilidad a nivel gen

Tirar dado
Generar número aleatorio rand entre 0 y 1 Si rand es menor que probabilidad   Devolver true Si no devolver false

- Búsqueda del mejor o peor cromosoma: Dada una población, evalúa sus cromosomas y devuelve el mejor o peor de ellos

Selección

Generar aux (1000 si buscamos mejor, 0 si buscamos peor)
Para cada cromosoma i en población
  Si función objetivo de i es menor/peor que aux (respectivamente)
  cambiar valor de aux a función objetivo de i
  almacenar posición de i en selección
Fin para
Aumentar evaluaciones de función objetivo en tam_población unidades
Devolver selección (evaluaciones pasadas por referencia)

Pseudocódigo del método de búsqueda

Los métodos de búsqueda de esta práctica están basados en poblaciones y presentan un conjunto de posibles soluciones las cuales se mezclan y modifican para hallar una mejor solución. Los algoritmos estudiados imitan el proceso de la selección natural, tan importante en la evolución biológica de las poblaciones.

A continuación se muestra el algoritmo genético general

Generación de población inicial

```
Generar vector población de cromosomas de tamaño 50
Para cada cromosoma i de población
  Para cada elemento j de i
    Asignar a j un cluster generado aleatoriamente
  Fin para
  Modificar i para que ningún cluster quede vacío
Fin para
Copiar población en nueva_población
```

Inicio iteraciones

```
Mientras no se realicen 100000 evaluaciones y haya cambios positivos
  Almacenar función objetivo de población
  Generar cromosomas hijos con operador de cruce *
  Generar cromosomas mutados con mutación uniforme y probabilidad 0.001
  Sustituir cromosomas mutados en nueva_población
  Sustituir hijos * en nueva_población
  Si la función objetivo de nueva_población es menor a la de población
    Copiar nueva_población en población
Fin mientras
```

Fin iteraciones

```
Seleccionar mejor solución
Convertir mejor solución a antigua estructura de datos para su futura evaluación
```

*) Difieren entre los distintos algoritmos genéticos estudiados

- Algoritmo empleado para la mutación

Mutar (común)

Para cada elemento i de la población
  Si tirar dado con probabilidad 0.001*tam_población
  Añadir a vector mutados mutación uniforme del elemento i
  Almacenar posición del elemento i
Fin para
Devolver mutados (posiciones pasadas por parámetro)

- Algoritmo empleado para el cruce

Cruzar estacionario
Para cada i de 2(padres)*1(probabilidad)   Añadir a padres ganador de torneo binario

```
  Aumentar evaluaciones (función objetivo) en 2 unidades
Fin para
Para cada 2*½ (esperanza) de padres
  Switch algoritmo
  AGE-UN
    Añadir a hijos cruce uniforme de padre[i] y padre[i+1]
    Añadir a hijos cruce uniforme de padre[i] y padre[i+1]
  AGE-SF
    Añadir a hijos cruce segmento fijo de padre[i] y padre[i+1]
    Añadir a hijos cruce segmento fijo de padre[i] y padre[i+1]
  Fin switch
Fin para
Devolver hijos
```

Cruzar generacional

```
Para cada i de 50(padres)*0.7(probabilidad)
  Añadir a padres ganador de torneo binario
  Aumentar evaluaciones (función objetivo) en 2 unidades
  Almacenar posición de padre en la población
Fin para
Para cada 50*0.7/2 (esperanza) de padres
  Switch algoritmo
  AGG-UN
    Añadir a hijos cruce uniforme de padre[i] y padre[i+1]
    Añadir a hijos cruce uniforme de padre[i] y padre[i+1]
  AGG-SF
    Comparar función objetivo de padre[i] y padre[i+1]
    Padre1 es padre con menor función objetivo
    Padre2 es padre con mayor función objetivo
    Añadir a hijos cruce segmento fijo de padre1 y padre2
    Añadir a hijos cruce segmento fijo de padre1 y padre2
    Aumentar evaluaciones (función objetivo) en 2 unidades
  Fin switch
Fin para
Devolver hijos (posiciones pasadas por parámetro)
```

- Sustitución de hijos para diferentes algoritmos

Sustitución estacionario

```
Selección del peor cromosoma de la población
Sustituir peor por el primer hijo
Selección del peor cromosoma de la nueva población
Sustituir el peor con el segundo hijo
```

Sustitución generacional

```
Selección del mejor cromosoma de la población
Para cada i de posiciones_hijos
  Si posiciones_hijos = mejor
    Almacenar población[mejor] en variable auxiliar
  Población[posiciones_hijos[i]] es hijos[i]
Fin para
Si se ha almacenado el mejor en variable auxiliar
  Selección del peor cromosoma de la población
  Sustituir el peor con auxiliar
```

Basándonos en los resultados obtenidos al ejecutar los algoritmos genéticos, el algoritmo genérico generacional que presenta mejores resultados resultó ser aquel que usa cruce uniforme (estos resultados serán expuestos en la sección correspondiente al análisis de resultados), por tanto es en ese algoritmo en el que nos basaremos para el desarrollo de los algoritmos meméticos, aplicando sus correspondientes datos:

- La población tendrá 10 cromosomas
- Para la mutación se seguirá empleando una probabilidad del 0.001
- Para el cruce, se volverá a emplear una probabilidad del 0.7, sin embargo, el número de padres será 10, por tanto contaremos con 10*0.7/2 parejas a cruzar.

Otra variación será que almacenaremos en un contador las iteraciones del algoritmo que vayamos realizando, de forma que podamos optimizar la población por cada 10 generaciones. A continuación se explica el algoritmo de búsqueda local suave

```
Almacenar función objetivo del cromosoma

Inicio iteraciones
Mientras ( se realicen cambios o no se produzcan 0.1*num_elementos del cromosma fallos ) y no se haya recorrido el cromosoma entero
  Obtener un vecino con mutación uniforme
  Si la función objetivo del vecino es menor que la del cromosoma
    El cromosoma y su función objetivo pasan a ser el vecino y la suya, respectivamente
    Han habido cambios
  Si no, aumentar fallos en una unidad
  Aumentar contador en una unidad
  Aumentar evaluaciones de la función en una unidad
  Si las evaluaciones superan 100000, salir del mientras
Fin mientras

Fin iteraciones
Devolver cromosoma
```

- Algoritmo empleado para la optimización

Optimizar (10,10)
Desordenar población Para cada cromosoma i de población   Aplicar búsqueda local suave sobre i Fin para
Optimizar (10,01)

Desordenar población
Para cada cromosoma i de población
  Si tirar dado con probabilidad 0.1
  Aplicar búsqueda local suave sobre i
Fin para

Optimizar (10,01)
Selección del mejor cromosoma de la población Aplicar búsqueda local sobre el mejor

Pseudocódigo del algoritmo de comparación

Los algoritmos de comparación empleados han sido los algoritmos de Búsqueda Local y Greedy COPKM empleados para la práctica anterior.

- La Búsqueda Local es un proceso iterativo que parte de una solución aleatoria y la mejora realizando modificaciones locales respecto a la función objetivo.

Búsqueda Local

```
Para cada elemento i de X
  asignar un cluster aleatorio a cluster
  asignar a dicho cluster la posición i
  modificar hasta que ningún cluster quede vacío
Calcular centroides
Almacenar función objetivo
Mientras haya cambios y contador < 100000
  cambios = false
  para cada elemento i de X
    para cada cluster j mientras no se encuentre una mejora
      si cluster no está vacío
        calcular nuevo cluster
      si probar vecino
        eliminar el elemento i del cluster al que pertenecía
        asignar el elemento i al nuevo cluster
        modificar el valor de cluster del elemento i
        mejora encontrada y cambios=true
    fin para
  fin para
  aumentar contador en una unidad
Fin mientras
```

- Los algoritmos greedy o voraces son aquellos que, para resolver el problema, eligen la opción óptima local en cada paso tratando así de llegar a una solución general óptima.

Greedy

```
Almacenar infactibilidad
Calcular centroides aleatorios
Mientras haya cambios
  Para cada elemento de X
    Para cada cluster j de C
      si probar cluster es menor que máximo auxiliar
        vaciar lista de empates
        modificar máximo auxiliar
        añadir j a la lista de empates
      si son iguales añadir j a la lista de empates
    Fin para
    si la lista de empates tiene más de 1 cluster
      si algún cluster k de la lista está vacío, nuevo cluster = k
      si no nuevo cluster = desempatar por distancia
    si no nuevo cluster = el cluster de la lista
    si no es la primera iteración eliminar el elemento del cluster al que pertenecía
      asignar el elemento a nuevo cluster
```

```
        modificar cluster del elemento
    Fin para
    Si infactibilidad es igual a infactibilidad antes, diferencia = false
    Aumentar contador
Fin mientras
```

Además de estos dos algoritmos, también se han implementado 3 algoritmos meméticos que difieren de los pedidos para la práctica en la que sus cromosomas se optimizan en cada generación:

- AM(01,10) realiza la misma optimización que AM(10,10) pero en cada generación.
- AM(01,01) realiza la misma optimización que AM(10,01) pero en cada generación.
- AM(01,01 mejor) realiza la misma optimización que AM(10,01 mejor) pero en cada generación.

Procedimiento y manual de uso

Para el desarrollo de esta práctica no se ha partido de ningún framework de metaheurísticas ni de un código proporcionado, más allá de la librería random dada para la generación de números pseudoaleatorios. La estructura empleada en la práctica consiste en una librería llamada "poblacion.h" donde se han incluido todas las funcionalidades necesarias para los algoritmos evolutivos y en un main, alojado en "cluster.cpp" desde donde se llaman a las funciones necesarias para la ejecución según los parámetros que se indiquen. También se hace uso de la librería "util.h" creada para la práctica anterior. Para las estructuras de datos se ha usado la librería std.

Se ha incluido un archivo makefile en la carpeta de software para la compilación del programa. Se ha creado un único ejecutable para todos los algoritmos, llamado cluster; éste se encuentra en el directorio bin. Para ejecutar el programa, es necesario pasar como parámetros, en el orden indicado, los siguientes datos:

- Modo: hace referencia al algoritmo que se ejecutará. En esta práctica, los parámetros aceptados serán:
    - "COPKM" para la ejecución del algoritmo Greedy
    - "BL" para la ejecución del algoritmo de Búsqueda Local
    - "AGG-UN" para el algoritmo Genético Generacional con cruce uniforme
    - "AGG-SF"para el algoritmo Genético Estacionario con cruce por segmento fijo
    - "AGE-UN" para el algoritmo Genético Estacionario con cruce uniforme
    - "AGE-SF" para el algoritmo Genético Estacionario con cruce por segmento fijo
    - "AM-10" para la ejecución del algoritmo Memético (10,10)
    - "AM-01" para la ejecución del algoritmo Memético (10,01)
    - "AM-M" para la ejecución del algoritmo Memético (10,01 mejor)
    - "AM-10-1" para la ejecución del algoritmo Memético (01,10)
    - "AM-01-1" para la ejecución del algoritmo Memético (01,01)
    - "AM-M-1" para la ejecución del algoritmo Memético (01,01 mejor)
- Conjunto: hace referencia al conjunto de datos sobre el que se ejecutará el problema. Para evitar tener que introducir la ruta completa de todos los ficheros en cada ejecución, éstos han de encontrarse dentro de un directorio llamado "datos", en el mismo directorio desde donde se llama a la función. Los parámetros aceptados serán:
    - "I" para el conjunto Iris
    - "E" para el conjunto Ecoli
    - "R" para el conjunto Rand
    - "NT" para el conjunto Newthyroid
- Porcentaje restricciones: hace referencia el porcentaje del total de restricciones posibles que se aplicarán al problema. Los parámetros aceptados serán "10" y "20"
- Nº repeticiones: cantidad de veces que deseemos que se ejecute el algoritmo
- Resultado: si deseamos que se muestre, además de la salida estándar, el reparto de elementos en los distintos clusters, este parámetro será "S"; en caso contrario será "N"
- [Semillas]: hace referencia a las semillas de generación de números aleatorios. Se pueden introducir tantas como se deseen. Si no se introduce ninguna, se considerarán las semillas 1, 5, 10, 20 y 25.

Experimentos y análisis de resultados

Descripción de los casos del problema empleados y de los valores de los parámetros considerados en las ejecuciones de cada algoritmo

Para cada algoritmos se han realizado 50 ejecuciones por conjunto de datos, es decir, 25 por cada conjunto de restricciones. Se han elegido 5 semillas, en nuestro caso {1, 5, 10, 20 y 25}, y para todas ellas se ha ejecutado el programa 5 veces.

Resultados obtenidos según el formato especificado

Se han incluido los resultados de todas las ejecuciones de los algoritmos en formato ods en archivos individuales con el mismo nombre que el parámetro especificado en la sección anterior para cada algoritmo en el directorio "ejecuciones". En el mismo directorio se han añadido 4 archivos de comparación:

- "comparacion_BL_COPKM.ods" muestra una comparación entre los resultados medios obtenidos para cada semilla con los algoritmos de la primera práctica.
- "comparacion_genericos.ods" muestra una comparación entre los resultados medios obtenidos para cada semilla con los cuatro algoritmos genéticos.
- "comparacion_memeticos.ods" muestra una comparación entre los resultados medios obtenidos para cada semilla con los seis algoritmos meméticos (tres de ellos adicionales).
- "comparacion.ods" muestra la comparación entre las medias de los resultados obtenidos para todas las semillas con todos los algoritmos evaluados hasta el momento. Aunque los tiempos se han obtenido en nanosegundos con una mayor precisión, en esta comparación se muestran en segundos para facilitar la comprensión.

Comparación de todos los algoritmos con 10% de restricciones

10,00 %	Iris				Ecoli			
Algoritmo	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
COPKM	0,09556858	378,2	0,1475478	0,767989085	2,3718714	2638	6,73392	43,16101505
BL	0,104848	13	0,106145	1,848027832	2,5326792	633,8	3,0743884	88,04224211
AGG-UN	0,3028506	497,8	0,4308868	10,33772766	7,483966	1817,8	9,93691	55,98422595
AGG-SF	0,3029402	497,6	0,431727	11,62366093	7,483966	1817,8	9,93691	58,45547155
AGE-UN	0,3029402	497,6	0,431727	0,278142692	7,483966	1817,8	9,93691	30,50873602
AGE-SF	0,3029402	497,6	0,429776	0,071594953	7,481058	1826,2	9,94473	31,47126268
AM-10	0,104848	13	0,106145	11,80203787	3,65238	1209,4	4,714386	60,3105271
AM-01	0,1433944	110,6	0,1684206	7,546834678	4,94897	1439,4	6,4252	35,58245443
AM-M	0,1045456	16,2	0,1061498	7,495388233	6,244824	1630,6	8,162648	22,51644933
AM-10-1	0,104848	13	0,106145	11,21769076	3,80306	1227,4	4,90333	57,36321089
AM-01-1	0,2221474	297,8	0,2933824	6,510237071	4,464394	1348,4	5,732206	39,17025878
AM-M-1	0,18238064	209	0,23270372	7,864408716	7,372876	1773,2	9,657842	25,00122605

10,00 %	Rand				Newthyroid			
Algoritmo	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
COPKM	0,09524704	609,6	0,2109796	0,684134402	1,2921686	1001,4	2,5783302	4,919350623
BL	0,125279	0	0,125279	1,341885525	0,6809574	1130,8	1,2526174	1,68194946
AGG-UN	0,4100034	487,4	0,5767474	11,02062731	2,001892	1153	2,980618	17,55763764
AGG-SF	0,4100034	487,4	0,5767474	12,01559871	2,001892	1153	2,980618	20,36368514
AGE-UN	0,4100034	487,4	0,5767474	0,570211651	2,001892	1153	2,980618	12,30879174
AGE-SF	0,4100034	487,4	0,5767474	0,098550394	2,008792	1156	2,98493	7,201879179
AM-10	0,117006	0	0,117006	11,04941949	0,913521	929,8	1,42108	19,98150287
AM-01	0,293106	296,4	0,3980878	5,271149379	1,5776664	1105,2	2,420698	9,170354793
AM-M	0,1746542	90,4	0,2039962	6,761281482	1,277358	983	1,893436	7,068239721
AM-10-1	0,117006	0	0,117006	11,29396939	0,877064	943,4	1,369816	19,59914403
AM-01-1	0,2926422	291,4	0,3916934	6,605185748	1,6155752	1100,6	2,456532	11,99987059
AM-M-1	0,2903008	287	0,385151	5,389394907	1,6101968	1098,8	2,411536	9,048059894

Comparación de todos los algoritmos con 20% de restricciones

20,00 %	Iris				Ecoli			
Algoritmo	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
COPKM	0,09561876	750,4	0,15028562	2,721282291	3,0555152	5008,8	11,936774	18,26533541
BL	0,105452	21	0,106576	1,840369756	2,572108	1195	3,129222	81,95324424
AGG-UN	0,3029402	971,2	0,436502	12,90393794	7,474082	3612,6	9,967094	58,91278926
AGG-SF	0,3029402	971,2	0,436502	13,41033501	7,474082	3612,6	9,967094	64,08243375
AGE-UN	0,3029402	971,2	0,436502	0,429114119	7,464074	3626	9,959982	30,00835956
AGE-SF	0,3029402	971,2	0,436502	0,297511837	7,484722	3615,6	9,9774	28,44354182
AM-10	0,105452	21	0,106576	13,38771302	3,674756	2642,2	4,84036	66,04018613
AM-01	0,1449664	217,6	0,1739648	9,20153344	5,48475	3202,2	7,301992	40,42561123
AM-M	0,1448388	216,6	0,1706484	6,526331249	6,268368	3436,4	8,413018	29,99484829
AM-10-1	0,105452	21	0,106576	12,21472139	3,591582	2415,8	4,634356	63,55622189
AM-01-1	0,182299	397	0,2321124	6,82873912	6,315282	3296,8	8,324388	25,22225367
AM-M-1	0,2197996	584,6	0,2964466	6,501465922	6,804694	3443,2	9,020206	32,52768624

20,00 %	Rand				Newthyroid			
Algoritmo	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
COPKM	0,09518544	1234,2	0,2194204	0,609796485	1,292019	2031,6	2,6550182	2,807681082
BL	0,125279	0	0,125279	1,562074479	0,712057	2220,6	1,299056	1,781757686
AGG-UN	0,4100034	1016,4	0,5945596	11,92765527	2,005926	2311,6	3,064164	21,08391858
AGG-SF	0,4100034	1016,4	0,5945596	12,856576	2,005926	2311,6	3,064164	21,51804353
AGE-UN	0,4100034	1016,4	0,5945596	0,858426186	2,005926	2311,6	3,064164	4,15240468
AGE-SF	0,4105792	1005,04	0,59464488	0,260743655	2,005926	2311,6	3,064164	4,272447642
AM-10	0,117006	0	0,117006	12,92512779	0,9945152	1931,32	1,55010604	22,17171792
AM-01	0,2337858	390,6	0,3041204	8,189755189	1,6227326	2203,8	2,486924	6,925812982
AM-M	0,2927418	606	0,4025618	5,656732644	1,605308	2207,8	2,452506	12,28326809
AM-10-1	0,117006	0	0,117006	12,41342725	0,8873822	1788,4	1,38242	22,1539363
AM-01-1	0,292499	614,4	0,4035044	6,963725921	1,6151462	2181,8	2,461148	12,08148371
AM-M-1	0,2874518	587,8	0,3892238	9,345692097	1,457735	2113,2	2,209636	9,235827237

Análisis de resultados

He decidido realizar el análisis en función al orden en el que he ido desarrollando la práctica, comparando primero los dos tipos de algoritmos genéticos y luego los meméticos, en lugar de comparar todos los algoritmos entre ellos.

Comenzaremos entonces comparando los resultados obtenidos con los algoritmos genéticos. La característica que tal vez llame más la atención se la similitud entre los valores obtenidos, puesto que en varias ocasiones obtenemos el mismo valor agregado para cada semilla, sin ser éste un mínimo, empleando los cuatro algoritmos genéticos. Sin embargo, los algoritmos genéticos estacionarios resultan el doble de rápidos con el conjunto Ecoli y hasta 12 veces más rápidos con conjuntos pequeños como Iris y Rand.

10,00 %	Iris				Ecoli			
AGG-UN	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
Semilla 1	0,303024	493	0,432888	10195600822	7,49695	1836	9,97825	54969509626
Semilla 5	0,303024	493	0,432888	10094808442	7,5041	1793	9,93699	55963901749
Semilla 10	0,307421	511	0,446538	10125427089	7,41058	1841	9,86718	56104826495
Semilla 20	0,301678	494	0,423491	10350480882	7,52091	1810	9,98039	56107139463
Semilla 25	0,299106	498	0,418629	10922321053	7,48729	1809	9,92174	56775752426
Media	0,3028506	497,8	0,4308868	10,33772766	7,483966	1817,8	9,93691	55,98422595
AGG-SF								
Semilla 1	0,303024	493	0,432888	11549909351	7,49695	1836	9,97825	58531107260
Semilla 5	0,303472	492	0,437089	11687957178	7,5041	1793	9,93699	59611724955
Semilla 10	0,307421	511	0,446538	11679789081	7,41058	1841	9,86718	59389304132
Semilla 20	0,301678	494	0,423491	11818938165	7,52091	1810	9,98039	58073620923
Semilla 25	0,299106	498	0,418629	11381710852	7,48729	1809	9,92174	56671600468
Media	0,3029402	497,6	0,431727	11,62366093	7,483966	1817,8	9,93691	58,45547155

Nota: Los tiempos medios de cada semilla se expresan en nanosegundos. En las medias de éstos han sido convertidos a segundos

En esta tabla se muestra una comparación entre los resultados de AGG-UN y AGG-SF con los conjuntos Iris y Ecoli. Como se puede observar, la diferencia entre los resultados de ambos es escasa. Por un lado, el algoritmo que usa el operador de cruce

uniforme obtiene un valor agregado ligeramente inferior al obtenido con el operador de cruce por segmento fijo para el conjunto Iris. Para los conjuntos Ecoli y Newthyroid se han obtenido exactamente los mismos resultados y para el conjunto Rand la diferencia es de $-8{,}52E^{-5}$. En cuanto al tiempo, AGG-SF resulta un poco más lento para todos los conjuntos, aunque la diferencia no supera, para ninguno de los 4 conjuntos, los 3 segundos. En esta pequeña diferencia es en la que nos ha llevado a seleccionar el operador de cruce uniforme para el desarrollo de los algoritmos genéticos.

La exactitud con la que los resultados de ambos algoritmos coincidían me llevó a pensar que estaba ejecutando el mismo algoritmo dos veces por error. Decidí entonces imprimir, además de los resultados expresados en la tabla, el número de iteraciones del algoritmo y el número de evaluaciones de la función objetivo realizadas, obteniendo para el mismo conjunto y la misma semilla, los siguientes resultados:

AGG-UN

Semilla = 7
Iteraciones = 588
Evaluaciones = 100010

Tasa_C|Tasa_i|Agreg|Tiempo
0.302079 501 0.43421 10579920064

AGG-SF

Semilla = 7
Iteraciones = 490
Evaluaciones = 100010

Tasa_C|Tasa_i|Agreg|Tiempo
0.302079 501 0.43421 11067141332

Como podemos ver, para esta semilla y este conjunto, ambos algoritmos se detienen porque alcanzan el máximo de evaluaciones establecido, no porque hayan llegado a un mínimo; pero no realizan el mismo número de iteraciones: el algoritmo generacional que utiliza el operador de cruce por segmento fijo, para mantener el elitismo, evalúa cada pareja para decidir de qué padre heredará más cada hijo; esto supone que por cada pareja cruzada, el número de evaluaciones aumentará.

Tras observar estos datos, llego a la conclusión de que estos algoritmos, al centrarse en la exploración, “pierden” demasiadas evaluaciones impidiendo que, con el máximo establecido, no puedan alcanzar un mínimo. La similitud de los resultados, conociendo que el número de iteraciones difiere, ha de ser causada por la generación de la solución inicial.

También cabe destacar que, pese a lo que habría imaginado en un principio, AGG-SF, que realiza menos iteraciones (evalúa más veces) resulta más lento que AGG-UN, que realiza más iteraciones, lo cual significa que el peso computacional que supone evaluar un cromosoma es elevado; de no ser así, al escalar rápidamente las evaluaciones, el tiempo también se vería reducido. Por ende, si buscáramos optimizar nuestro programa, indudablemente deberíamos centrarnos en la función objetivo.

10,00 %	Iris				Ecoli			
AGE-UN	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
Semilla 1	0,303024	493	0,432888	123855486,4	7,49695	1836	9,97825	38601286844
Semilla 5	0,303472	492	0,437089	431295070	7,5041	1793	9,93699	21121264036
Semilla 10	0,307421	511	0,446538	372742337,6	7,41058	1841	9,86718	30966317462
Semilla 20	0,301678	494	0,423491	316826895,6	7,52091	1810	9,98039	41145134950
Semilla 25	0,299106	498	0,418629	145993671,4	7,48729	1809	9,92174	20709676802
Media	0,3029402	497,6	0,431727	0,278142692	7,483966	1817,8	9,93691	30,50873602
AGE-SF								
Semilla 1	0,303024	493	0,432888	58589529,2	7,49695	1836	9,97825	20872136750
Semilla 5	0,303472	492	0,437089	137727545,4	7,48956	1835	9,97609	30612475292
Semilla 10	0,307421	511	0,436783	38915693,4	7,41058	1841	9,86718	39247369770
Semilla 20	0,301678	494	0,423491	69777742	7,52091	1810	9,98039	37862538093
Semilla 25	0,299106	498	0,418629	52964253,6	7,48729	1809	9,92174	28761793499
Media	0,3029402	497,6	0,429776	0,071594953	7,481058	1826,2	9,94473	31,47126268

10,00 %	Rand				Newthyroid			
AGE-UN	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
Semilla 1	0,414587	502	0,597759	69674600,8	2,01767	1182	2,99427	13545056262
Semilla 5	0,406012	488	0,572615	80507417,2	1,97012	1163	2,96778	10289885520
Semilla 10	0,410263	483	0,574779	188725586	2,03003	1154	2,99215	10266543854
Semilla 20	0,409941	477	0,568427	2279655338	1,96858	1137	2,97794	17078262820
Semilla 25	0,409214	487	0,570157	232495314,8	2,02306	1129	2,97095	10364210244
Media	0,4100034	487,4	0,5767474	0,570211651	2,001892	1153	2,980618	12,30879174
AGE-SF								
Semilla 1	0,414587	502	0,597759	28479928	2,01767	1182	2,99427	6990072237
Semilla 5	0,406012	488	0,572615	98954858	2,00462	1178	2,98934	10699549217
Semilla 10	0,410263	483	0,574779	257831099	2,03003	1154	2,99215	3632715993
Semilla 20	0,409941	477	0,568427	29156148	1,96858	1137	2,97794	10936512105
Semilla 25	0,409214	487	0,570157	78329939,4	2,02306	1129	2,97095	3750546341
Media	0,4100034	487,4	0,5767474	0,098550394	2,008792	1156	2,98493	7,201879179

Nota: Los tiempos medios de cada semilla se expresan en nanosegundos. En las medias de éstos han sido convertidos a segundos

En estas tablas se muestra la comparación entre los resultados obtenidos con algoritmos estacionarios con los cuatro conjuntos. Podemos notar que la diferencia entre los valores agregados obtenidos con los diferentes operadores de cruce es mayor. Teniendo en cuenta también los resultados obtenidos al emplear un 20% de restricciones, AGE-UN suele obtener mejores valores agregados, aunque al no darse en todos los casos (en Iris con un 10% sucede lo contrario) no podemos tomarlo como verdad absoluta. Al realizar la misma prueba que hicimos para el análisis de los algoritmos generacionales, obtenemos lo siguiente:

AGE-UN

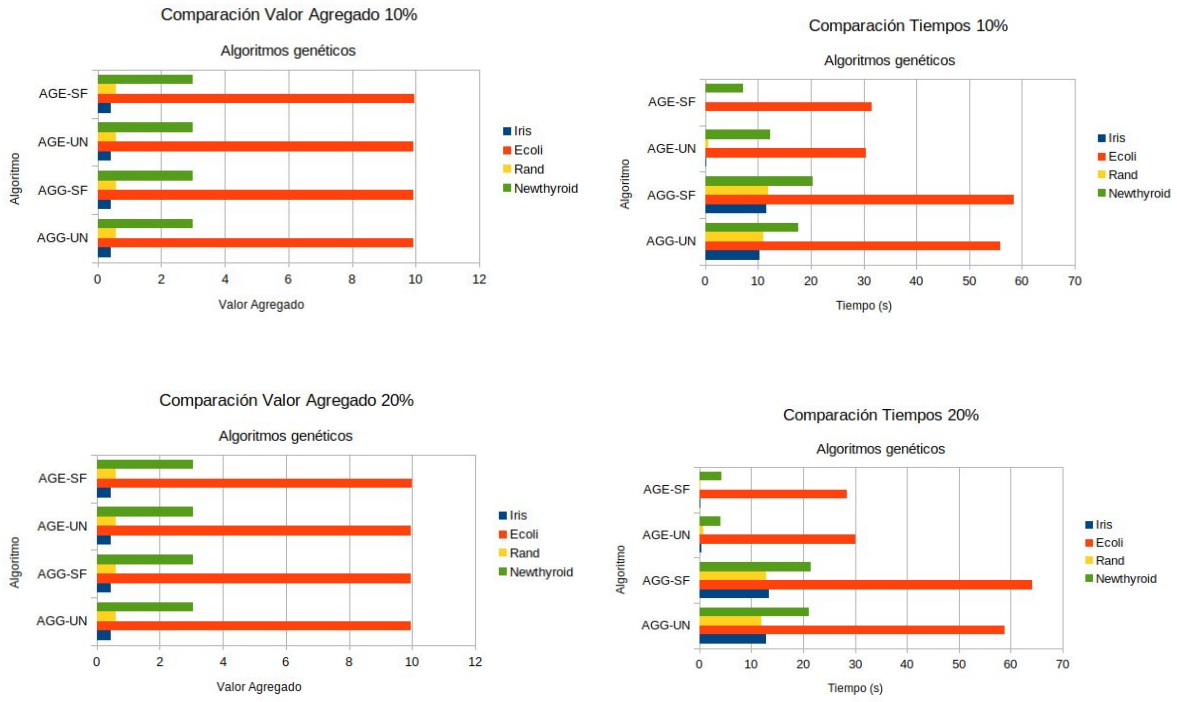
Semilla = 7
Iteraciones = 2
Evaluaciones = 358
Tasa_C|Tasa_i|Agreg|Tiempo
0.302079 501 0.43421 65713372

AGE-SF

Semilla = 7
Iteraciones = 7
Evaluaciones = 1128
Tasa_C|Tasa_i|Agreg|Tiempo
0.302079 501 0.43421 142427408

Ya que se obtiene el mismo valor que con los generacionales y no han alcanzado el máximo de evaluaciones establecido, podemos decir que se trata de un mínimo local en el que los cuatro algoritmos han quedado atrapados: en esta ocasión, ambos algoritmos se han detenido por no producirse cambios en la evaluación de la población tras realizar los cruces y

las mutaciones de las iteraciones segunda (con cruce uniforme) y séptima (con cruce por segmento fijo). Al realizarse menos cruces (solo obtenemos 2 hijos por cada iteración) hay más probabilidades de que no se realicen cambios significativos en la población.



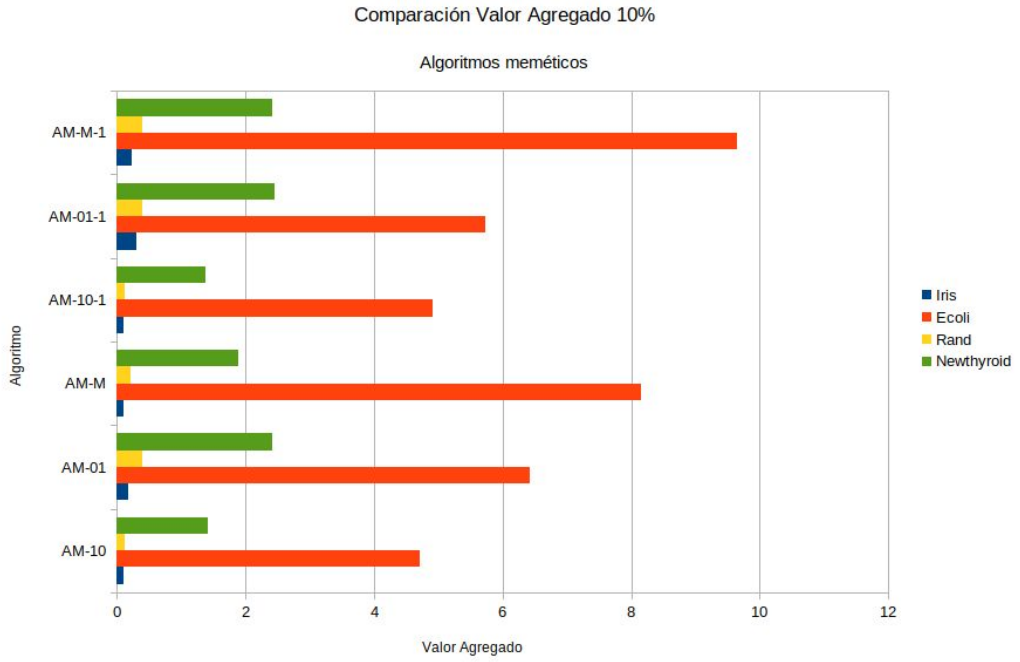
Comparando todos los algoritmos, y dado que los resultados obtenidos son bastante similares, AGE-UN es el que presenta mejor relación entre el valor agregado obtenido y el tiempo empleado. Sin embargo, parece que el tiempo de los algoritmos estacionarios aumenta más rápido que el de los algoritmos generacionales a medida que lo hace el tamaño de la población, por lo que podemos suponer que si quisiéramos aplicarlo sobre un conjunto de datos aún mayor que Ecoli, probablemente tardaría más que un AGG.

Ninguno de los algoritmos genéticos estudiados proporciona un mejor resultado que los proporcionados por BL y COPKM, trabajados en la práctica anterior, aunque esto resulte obvio, pues los algoritmos genéticos dependen en su mayor parte de la aleatoriedad.

Pasamos ahora a analizar los resultados de los algoritmos meméticos, que establecen un equilibrio entre la exploración (algoritmo generacional) y la explotación (búsqueda local suave). A continuación se muestran los resultados medios por semilla para cada algoritmo memético con los conjuntos Iris y Ecoli al 10% de restricciones.

10,00 %	Iris				Ecoli			
AM-10	Tasa_C	Tasa_inf	Agregado	Tiempo	Tasa_C	Tasa_inf	Agregado	Tiempo
Semilla 1	0,104848	13	0,106145	11482689128	3,47785	999	4,37051	58484499675
Semilla 5	0,104848	13	0,106145	11760690374	3,93079	1235	5,06933	58426293853
Semilla 10	0,104848	13	0,106145	11911381671	3,65623	1569	4,9809	62270647752
Semilla 20	0,104848	13	0,106145	11940486021	3,63771	975	4,51051	62171169744
Semilla 25	0,104848	13	0,106145	11914942144	3,55932	1269	4,64068	60200024501
Media	0,104848	13	0,106145	11,80203787	3,65238	1209,4	4,714386	60,3105271
AM-01								
Semilla 1	0,104848	13	0,106145	10320711943	4,41167	1437	5,66378	38051172105
Semilla 5	0,104848	13	0,106145	4146072554	7,49983	1778	9,9391	24203028336
Semilla 10	0,104848	13	0,106145	9152513537	4,42199	1373	5,66751	45117481967
Semilla 20	0,29758	501	0,417523	7047508641	3,85379	1387	5,11772	35582573314
Semilla 25	0,104848	13	0,106145	7067366717	4,55757	1222	5,73789	34958016423
Media	0,1433944	110,6	0,1684206	7,546834678	4,94897	1439,4	6,4252	35,58245443
AM-M								
Semilla 1	0,104092	21	0,106157	8319900983	7,45913	1765	9,8086	11595771053
Semilla 5	0,104848	13	0,106145	10389216821	4,46799	1408	5,74928	11131966832
Semilla 10	0,104092	21	0,106157	6242083925	7,41058	1841	9,86718	21990908759
Semilla 20	0,104848	13	0,106145	6264397316	4,70846	1438	6,11145	32890853030
Semilla 25	0,104848	13	0,106145	6261342119	7,17796	1701	9,27673	34972746981
Media	0,1045456	16,2	0,1061498	7,495388233	6,244824	1630,6	8,162648	22,51644933
AM-10-1								
Semilla 1	0,104848	13	0,106145	11168817019	4,37621	1291	5,55512	57775147668
Semilla 5	0,104848	13	0,106145	10995231224	3,91498	1226	4,98973	57247849484
Semilla 10	0,104848	13	0,106145	10849876923	3,85576	1308	5,01476	57060435743
Semilla 20	0,104848	13	0,106145	11428968261	3,70524	1214	4,83351	57311044125
Semilla 25	0,104848	13	0,106145	11645560351	3,16311	1098	4,12353	57421577445
Media	0,104848	13	0,106145	11,21769076	3,80306	1227,4	4,90333	57,36321089
AM-01-1								
Semilla 1	0,300257	471	0,412502	4276150586	4,15857	1275	5,40835	57673832821
Semilla 5	0,104848	13	0,106145	8603001822	4,63132	1385	5,86939	46142694158
Semilla 10	0,104848	13	0,106145	8673154942	4,59987	1400	5,92725	11600556131
Semilla 20	0,301678	494	0,423491	4427935718	4,59987	1400	5,92725	46156850261
Semilla 25	0,299106	498	0,418629	6570942286	4,33234	1282	5,52879	34277360538
Media	0,2221474	297,8	0,2933824	6,510237071	4,464394	1348,4	5,732206	39,17025878
AM-M-1								
Semilla 1	0,2990092	500	0,4264426	4318991016	7,38037	1756	9,69952	35110278696
Semilla 5	0,104092	21	0,106157	10983080516	7,35262	1778	9,59975	22947484086
Semilla 10	0,104848	13	0,106145	4403529559	7,41058	1841	9,86718	22529366578
Semilla 20	0,104848	13	0,106145	10870731294	7,38997	1760	9,61664	22236029551
Semilla 25	0,299106	498	0,418629	8745711193,6	7,33084	1731	9,50612	22182971348
Media	0,18238064	209	0,23270372	7,864408716	7,372876	1773,2	9,657842	25,00122605

Si se observa la tabla de resultados obtenida, llama la atención el hecho de que los algoritmos meméticos (10,10) y (01,10), para el conjunto Iris, obtienen siempre el mismo resultado independientemente de la semilla, el cual es el valor mínimo. Lo mismo ocurre empleando el 20% de restricciones y con el conjunto Rand (10 y 20% de restricciones). No sucede sin embargo con Ecoli y Newthyroid, cuyos valores no son constantes; deduzco que el número máximo de evaluaciones no es suficiente, aunque aumentar éste podría suponer un aumento considerable en el tiempo que restaría eficiencia al algoritmo. Con los otros algoritmos meméticos, aun alcanzando los mínimos mencionados en la mayoría de los casos, podemos observar como la modificación de la semilla de aleatoriedad altera el resultado que obtenemos, es decir, dependen demasiado de la aleatoriedad. Esta diferencia es debida al equilibrio mencionado anteriormente: los algoritmos (10,10) y (01,10) le dan un mayor peso a la explotación que a la explotación.



En esta gráfica podemos observar los valores agregados medios obtenidos para cada conjuntos con el 10% de restricciones empleando los 6 algoritmos meméticos. Como ya mencionamos anteriormente, para los conjuntos pequeños, los algoritmos AM-10 y AM-10-1 resultan ser los que mejor resultado ofrecen, y esto se mantiene con los otros dos conjuntos, aunque sin alcanzar a su valor mínimo. Entre ambos, el algoritmo AM-10, que realiza la

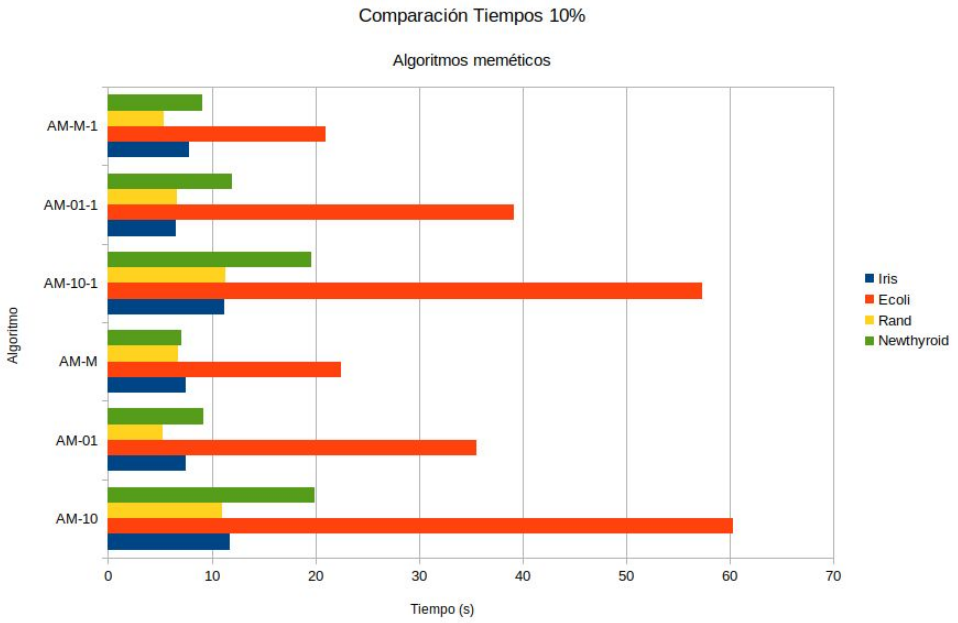
optimización de toda la población cada 10 generaciones, obtiene mejores resultados que el algoritmo AM-10-1, el cual lo realiza para cada generación. El algoritmo memético (01,10) le da demasiado peso a la explotación, aunque sea el segundo algoritmo memético que mejores resultados ofrece. De entre los 6 algoritmos estudiados, el AM-10 es el que mejor ratio presenta para este problema.

El algoritmo que aplica la búsqueda local sobre el mejor cromosoma de cada población para cada generación tiene un comportamiento similar a los algoritmos (01,01) y (10,01) en los conjuntos Iris y Rand, sin embargo, para conjuntos grandes como Ecoli obtiene un valor agregado especialmente alto. Se muestra a continuación los resultados obtenidos al aplicar este algoritmo sobre el conjunto Ecoli, con dos semillas distintas:

Semilla = 7
Iteraciones = 2
Evaluaciones = 200
Tasa_C|Tasa_i|Agreg|Tiempo
7.35286 1797 9.62145 116735647

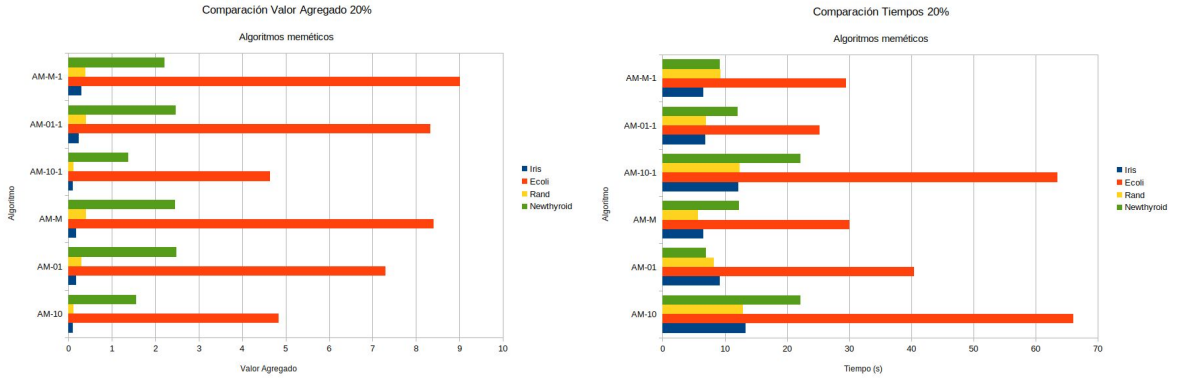
Semilla = 6
Iteraciones = 1219
Evaluaciones = 100070
Tasa_C|Tasa_i|Agreg|Tiempo
4.39073 1369 5.70136 52024740142

Podemos observar que tiene un comportamiento "errático", los valores obtenidos y el tiempo empleado varían demasiado dependiendo de la semilla de aleatoriedad que se elija. Aunque esto ocurra con la mayoría de los algoritmos meméticos, éste llega a obtener valores especialmente malos para el conjunto Ecoli en un tiempo menor al obtenido por otros algoritmos:



Los valores agregados obtenidos dan la impresión de ser inversamente proporcionales al tiempo empleado para cada algoritmo, siendo el algoritmo memético (10,01) el que mejor relación presenta entre ambos factores. Si agrupásemos los algoritmos por el conjunto de

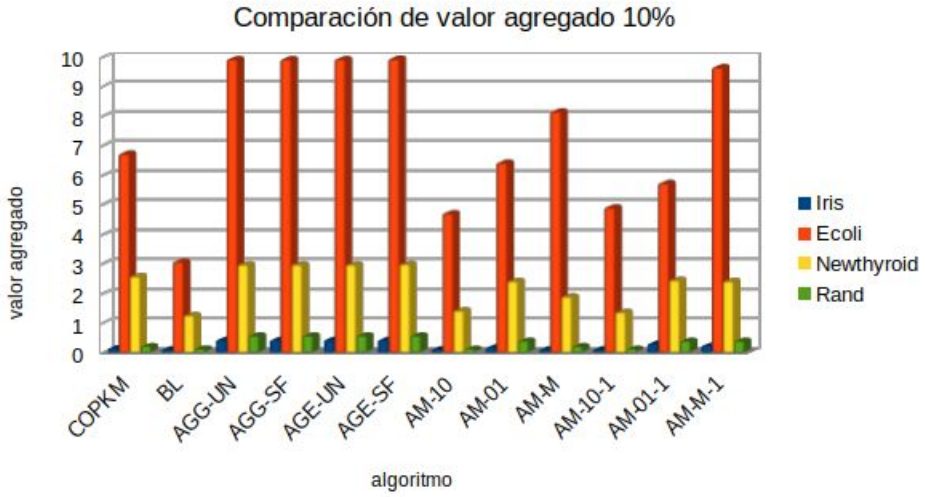
cromosomas sobre el que aplican la búsqueda local, los algoritmos que realizan la optimización cada 10 generaciones obtienen mejores prestaciones (tiempo y valores) que los que lo hacen en cada generación, mostrando de esta forma la importancia de preservar la exploración.

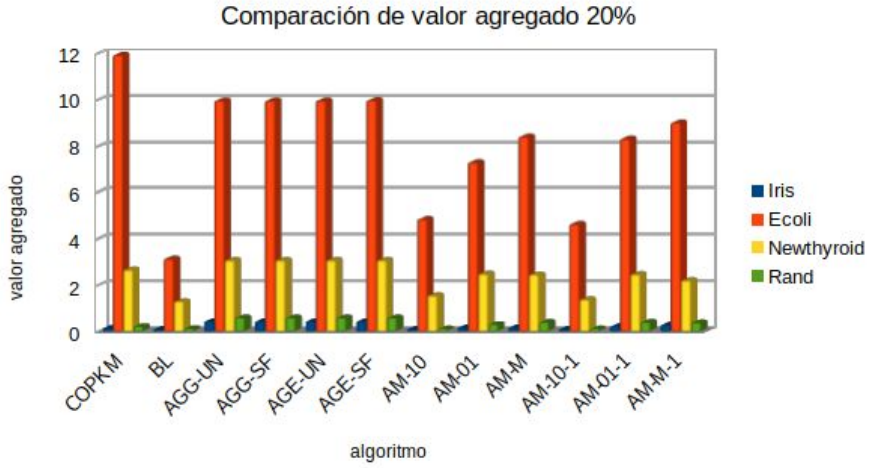


Para el conjunto del 20% de restricciones, aparecen algunas ligeras variaciones:

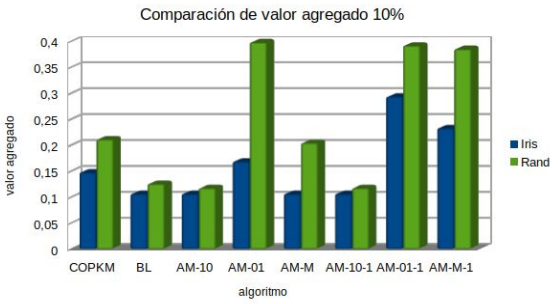
- El algoritmo memético (01,10) obtiene unos mejores resultados, tanto en tiempo como en valor agregado que el algoritmo (10,10), por lo que podríamos decir que para éste problema (20% de restricciones) el ratio óptimo es el presentado por AM-10-1
- El algoritmo (01,01) ha obtenido el menor tiempo de ejecución para el conjunto Ecoli, aunque sigue obteniendo peores valores que el algoritmo (01,10).

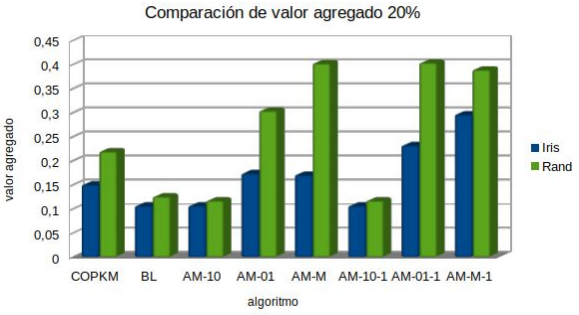
A continuación se muestran las comparaciones entre todos los algoritmos estudiados hasta el momento. Puesto que para los algoritmos evolutivos hemos tomado la función objetivo como evaluación, no he considerado interesante comparar las tasas C (desviación general) e inf (infactibilidad).

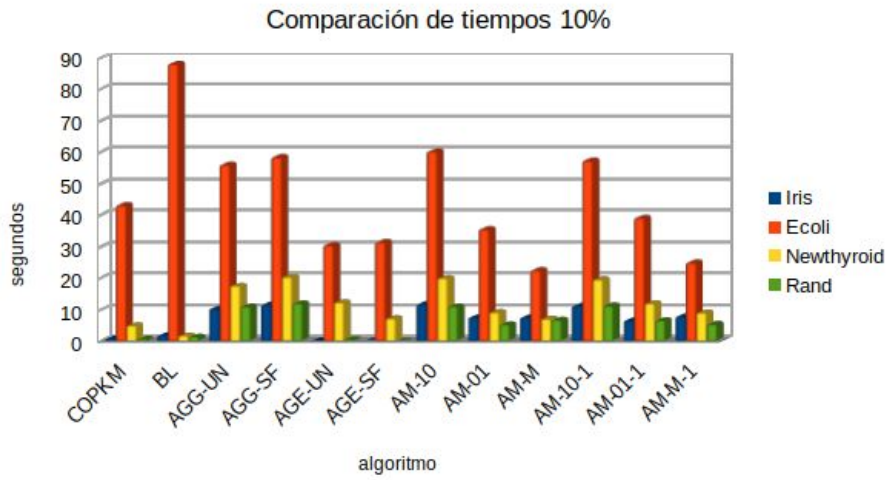


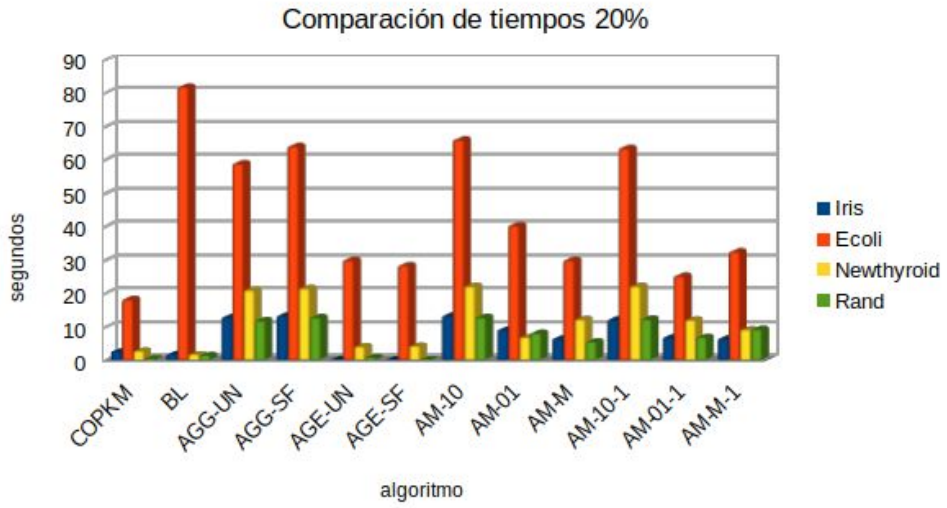


A simple vista, ningún algoritmo parece presentar un valor agregado mejor que el obtenido con la búsqueda local, aunque los algoritmos AM-10 y AM-10-1 consiguen minimizar el valor agregado del conjunto Rand, lo cual no se conseguía con la búsqueda local









La Búsqueda Local, como ya se comentó en la memoria de la primera práctica, es ineficiente para conjuntos de datos grandes, aunque obtiene las mejores evaluaciones.

Bibliografía

- https://es.wikipedia.org-*"Algoritmo evolutivo-Wikipedia, la enciclopedia libre"*
- https://sci2s.ugr.es- *"Algoritmo Genéticos I: Conceptos Básicos"*
- "*BIOBOTS y Algoritmos Evolutivos"* -DotCSV (YouTube)
- https://www.ecured.cu- *"Algoritmo Memético-EcuRed"*