Universidad de la Laguna

Facultad de Ingeniería Informática

Maximum diversity problem

DISEÑO Y ANÁLISIS DE ALGORITMOS

Carlos Díaz Calzadilla

Mayo 2020

Contents

1 Introducción

Para comenzar, hay que describir el problema a tratar. En el *Maximum diversity problem* lo que se persigue es encontrar el subconjunto de elementos de diversidad máxima de un conjunto dado de elementos.
Dado un conjunto S = $(s_1, s_2, ..., s_n)$ de n elementos, en el que cada elemento s_i, es un vector $s_i = (s_{i_1}, s_{i_2}, ..., s_{i_k})$. Sea, asimismo, d_{i_j} la distancia entre los elementos i y j. Por otro lado, si $m < n$ es el tamaño del subconjunto que se busca el problema puede formularse como:

$$Maximizar\ z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{i_j} x_i x_j$$

Sujeto a:

$$\sum_{i=1}^{n} x_i = m$$

$$x_i \in \{0, 1\} \quad i = 1, 2, ..., n$$

donde los valores de x representan:

- 1 → s_i pertenece a la solucion.
- 0 → s_i NO pertenece a la solucion.

Durante este informe, se van a describir los algortítmos heurísticos que se han implementado, además de los resultados obtenidos a partir de los mismos en formato de tabla.

2 Especificaciones del ordenador

En este apartado se van a hacer mención, de forma breve las especificaciones del ordenador donde se han hecho las pruebas de los diferentes algoritmos a la hora de obtener los resultados que se verán más adelante en las tablas. Ahora en la siguiente imagen se puede ver una breve descripción de las mismas:

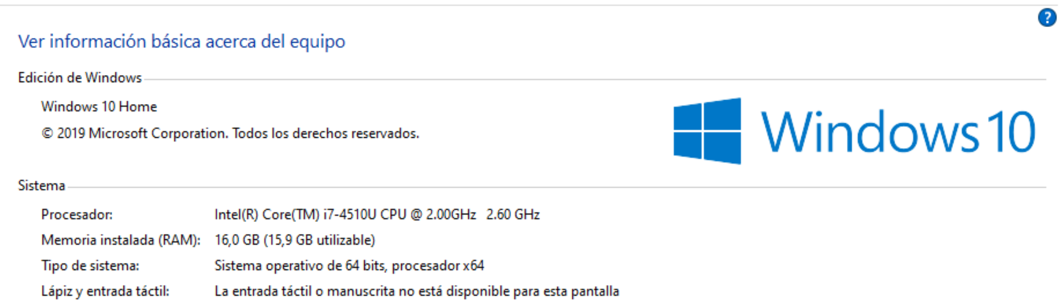

Figure 1: Especificaciones del ordenador

3 Estructuras de datos usadas

Para poder implemtentar estos algoritmos, habría que destacar que se ha usado el patrón de diseño *Strategy*. Este da una mayor facilidad debido a que permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades. Por ende, se ha creado una clase *Exec* que es la que se va a encargar de gestionar los diferentes elementos como el tiempo de ejecución, la lectura del fichero, ejecución del algoritmo,...

También se ha creado una clase *Algorithm* que va a ser la clase general de todos los algoritmos. A su vez, a esta se le ha añadido un atributo que representa los datos que se leen del fichero (para una mayor facilidad a la hora de acceder a los datos). Entonces, como se estaba mencionando, cada algoritmo hereda de este general, teniendo que definir un método *run* que es el principal que va a ejecutar el algoritmo. A continuación se puede ver una representación de la jerarquía que se mencionaba antes:

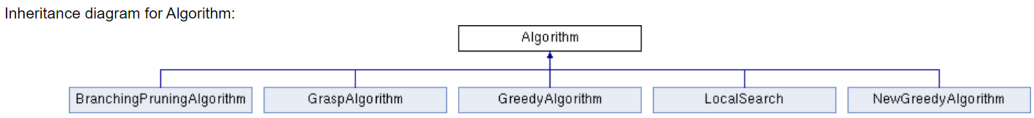

Figure 2: Jerarquía de los algoritmos

Cabe mencionar otro aspecto importante. Para poder afrontar el algoritmo de *Ramificación y Poda* se creó una clase nodo (porque al tener una representación y estructuras de datos similares a las de un árbol facilitaría la tarea de implementar el algoritmo). Por ende, este nodo va a estar formado por un vector que va a contener la solución parcial y de algunas variables booleanas para saber si el mismo es podado o se ha expandido,etc. Por otro lado, destacar que el algoritmo de poda va a ser el que va a contener un árbol de nodos, facilitando mucho la implementación y comprensión del código.

4 Algortimos voraces

4.1 Voraz constructivo

El primer algoritmo es el voraz proporcionado en el pseudocódigo. Este algoritmo parte de una solución inicial vacía, luego se crea una conjunto *Elem* formado por todos los elementos y se determina el centro de gravedad de dicho conjunto. Luego, de forma iterativa (hasta que el tamaño de la solución sea el deseado), se va a obtener el elemento más alejado entre el centro de gravedad y el conjunto *Elem*. Posteriormente, se añade dicho elemento a la solución y se quita del conjunto *Elem*. Y se determina el nuevo centro de nuestra solución. Una vez que esto ha acabado se devuelve la solución (son los índices de los vectores).

4.1.1 Tabla de resultados

Problema	n	K	m	z	S	CPU(µs)
max_div_15_2	15	2	2	11.8592	{9, 7}	39
max_div_15_2	15	2	3	25.7262	{9, 7, 4}	43
max_div_15_2	15	2	4	48.4139	{9, 7, 4, 11}	47
max_div_15_2	15	2	5	73.5619	{9, 7, 4, 11, 2}	54
max_div_20_2	20	2	2	8.51033	{18, 19}	47
max_div_20_2	20	2	3	21.9961	{18, 19, 9}	59
max_div_20_2	20	2	4	39.5682	{18, 19, 9, 3}	94
max_div_20_2	20	2	5	61.2393	{18, 19, 9, 3, 13}	73
max_div_30_2	30	2	2	11.6571	{9, 28}	61
max_div_30_2	30	2	3	28.9443	{9, 28, 2}	50
max_div_30_2	30	2	4	52.7712	{9, 28, 2, 11}	62
max_div_30_2	30	2	5	80.9102	{9, 28, 2, 11, 13}	66
max_div_15_3	15	3	2	13.2732	{12, 9}	43
max_div_15_3	15	3	3	30.3241	{12, 9, 5}	31
max_div_15_3	15	3	4	59.7638	{12, 9, 5, 11}	73
max_div_15_3	15	3	5	94.7487	{12, 9, 5, 11, 14}	99
max_div_20_3	20	3	2	11.8003	{13, 14}	47
max_div_20_3	20	3	3	30.8727	{13, 14, 8}	51
max_div_20_3	20	3	4	56.5347	{13, 14, 8, 3}	54
max_div_20_3	20	3	5	92.8298	{13, 14, 8, 3, 17}	44
max_div_30_3	30	3	2	13.2732	{17, 7}	53
max_div_30_3	30	3	3	33.8423	{17, 7, 24}	64
max_div_30_3	30	3	4	63.5184	{17, 7, 24, 14}	81
max_div_30_3	30	3	5	99.5088	{17, 7, 24, 14, 15}	61

Figure 3: Tabla resultados Voraz constructivo

4.2 Voraz destructivo

Para esta otra versión del voraz, la principal diferencia es que la solución desde la que vamos a partir contiene todos los nodos (la solución está formada por todos los elementos y en cada iteración se le van eliminando nodos). En primer lugar creamos un conjunto *Elem* que va a contener todos los elementos, al igual que la solución. De este vamos a determinar el centro de gravedad. Luego, vamos a iterar en un bucle hasta que el tamaño de la solución sea el deseado, determinando cual es elemento más cercano del conjunto *Elem* al centro, luego se busca en la solución y se elmina (a su vez se elimina de *Elem*) y se calcula el nuevo centro de la solución. Cuando el tamaño de la solución es el deseado se devuelve.

4.2.1 Tabla de resultados

Problema	n	K	m	z	S	CPU(µs)
max_div_15_2	15	2	2	11.8592	{7, 9}	73
max_div_15_2	15	2	3	23.7964	{2, 7, 9}	79
max_div_15_2	15	2	4	46.6528	{2, 7, 9, 11}	76
max_div_15_2	15	2	5	73.5619	{2, 4, 7, 9, 11}	59
max_div_20_2	20	2	2	8.51033	{18, 19}	89
max_div_20_2	20	2	3	21.9961	{9, 18, 19}	111
max_div_20_2	20	2	4	39.5682	{3, 9, 18, 19}	98
max_div_20_2	20	2	5	60.4301	{3, 9, 11, 18, 19}	131
max_div_30_2	30	2	2	11.6571	{9, 28}	138
max_div_30_2	30	2	3	28.9443	{2, 9, 28}	193
max_div_30_2	30	2	4	52.7712	{2, 9, 11, 28}	184
max_div_30_2	30	2	5	80.9102	{2, 9, 11, 13, 28}	180
max_div_15_3	15	3	2	13.2732	{9, 12}	76
max_div_15_3	15	3	3	30.3241	{5, 9, 12}	91
max_div_15_3	15	3	4	58.7287	{5, 9, 12, 14}	84
max_div_15_3	15	3	5	96.0858	{4, 5, 9, 12, 14}	158
max_div_20_3	20	3	2	11.5909	{3, 17}	184
max_div_20_3	20	3	3	29.1574	{3, 13, 17}	114
max_div_20_3	20	3	4	56.6903	{3, 13, 14, 17}	145
max_div_20_3	20	3	5	92.8298	{3, 8, 13, 14, 17}	200
max_div_30_3	30	3	2	12.6137	{6, 17}	161
max_div_30_3	30	3	3	34.2905	{6, 17, 24}	255
max_div_30_3	30	3	4	63.702	{6, 14, 17, 24}	219
max_div_30_3	30	3	5	99.592	{6, 14, 15, 17, 24}	157

Figure 4: Tabla resultados Voraz destructivo

5 Búsqueda local y GRASP

5.1 Búsqueda local

Este algoritmo parte de una solución dada por otro. En nuestro caso, se han hecho dos ejecuciones (partiendo que en una ejecución se genera la tabla), cambiando el algoritmo a usar. Los dos que se han escogido han sido el greedy constructivo y destructivo. Entonces partiendo de dicha solución y de la distancia que se obtiene de la misma. Se va a iterar hasta que no se encuentre una mejor solución, es decir, hasta encontrar una solución peor. Durante esas iteraciones lo que se hace es realiza una búsqueda local haciendo intercambios de los elementos en la solución con otros que no lo están, para ver si con alguna de las combinaciones la solución mejora. Si mejora, la solución se actualiza y si empeora, se acaba el algoritmo y se devuelve la mejor solución (siempre se va a encontrar una solución mejor o igual a la inicial).

5.1.1 Tabla de resultados

Como se puede ver en la *figura 5*, vemos que lo que se comentaba arriba es cierto, es decir, siempre se va a obtener al menos la misma solución inicial o una mejor. A su vez, se puede ver que en algunos casos la solución mejora, como son los casos que cuando el valor de m es 5, el valor de la solución suele mejorar. Esto respecto al constructivo. Luego al fijarnos en la *figura 6*, vemos que los resultados similares, pero como el algoritmo destructivo suele dar mejores soluciones que el constructivo, esto hace que no varíe mucho el resultado. También mencionar que si nos fijamos en los tiempos de ejecución, vemos que algunas veces, cuando la solución mejora el tiempo de CPU aumenta ligeramente.

Problema	n	K	m	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	11.8592	11.8592	{9, 7}	{9, 7}	21
max_div_15_2.txt	15	2	3	25.7262	27.3727	{9, 7, 4}	{9, 7, 1}	79
max_div_15_2.txt	15	2	4	48.4139	49.5462	{9, 7, 4, 11}	{9, 7, 4, 6}	128
max_div_15_2.txt	15	2	5	73.5619	76.6535	{9, 7, 4, 11, 2}	{9, 7, 1, 11, 2}	199
max_div_20_2.txt	20	2	2	8.51033	8.51033	{18, 19}	{18, 19}	26
max_div_20_2.txt	20	2	3	21.9961	21.9961	{18, 19, 9}	{18, 19, 9}	36
max_div_20_2.txt	20	2	4	39.5682	40.0023	{18, 19, 9, 3}	{2, 19, 9, 3}	179
max_div_20_2.txt	20	2	5	61.2393	62.8729	{18, 19, 9, 3, 13}	{18, 19, 9, 3, 2}	431
max_div_30_2.txt	30	3	2	11.6571	11.6571	{9, 28}	{9, 28}	42
max_div_30_2.txt	30	3	3	28.9443	28.9443	{9, 28, 2}	{9, 28, 2}	67
max_div_30_2.txt	30	3	4	52.7712	52.7712	{9, 28, 2, 11}	{9, 28, 2, 11}	100
max_div_30_2.txt	30	3	5	80.9102	80.9102	{9, 28, 2, 11, 13}	{9, 28, 2, 11, 13}	248
max_div_15_3.txt	15	3	2	13.2732	13.2732	{12, 9}	{12, 9}	33
max_div_15_3.txt	15	3	3	30.3241	31.8685	{12, 9, 5}	{12, 7, 5}	89
max_div_15_3.txt	15	3	4	59.7638	59.7638	{12, 9, 5, 11}	{12, 9, 5, 11}	46
max_div_15_3.txt	15	3	5	94.7487	96.0858	{12, 9, 5, 11, 14}	{12, 9, 5, 4, 14}	254
max_div_20_3.txt	20	3	2	11.8003	11.8003	{13, 14}	{13, 14}	27
max_div_20_3.txt	20	3	3	30.8727	30.8727	{13, 14, 8}	{13, 14, 8}	43
max_div_20_3.txt	20	3	4	56.5347	56.6903	{13, 14, 8, 3}	{13, 14, 17, 3}	213
max_div_20_3.txt	20	3	5	92.8298	92.8298	{13, 14, 8, 3, 17}	{13, 14, 8, 3, 17}	98
max_div_30_3.txt	30	3	2	13.0737	13.0737	{17, 7}	{17, 7}	39
max_div_30_3.txt	30	3	3	33.8423	34.2905	{17, 7, 24}	{17, 6, 24}	179
max_div_30_3.txt	30	3	4	63.5184	63.702	{17, 7, 24, 14}	{17, 6, 24, 14}	364
max_div_30_3.txt	30	3	5	99.5088	99.592	{17, 7, 24, 14, 15}	{17, 6, 24, 14, 15}	687

Figure 5: Tabla resultados LocalSearch con Solucción Inicial dada por Greedy Constructivo

Problema	n	K	m	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	11.8592	11.8592	{7, 9}	{7, 9}	23
max_div_15_2.txt	15	2	3	23.7964	27.3727	{2, 7, 9}	{1, 7, 9}	88
max_div_15_2.txt	15	2	4	46.6528	48.4139	{2, 7, 9, 11}	{4, 7, 9, 11}	144
max_div_15_2.txt	15	2	5	73.5619	76.6535	{2, 4, 7, 9, 11}	{2, 1, 7, 9, 11}	247
max_div_20_2.txt	20	2	2	8.51033	8.51033	{18, 19}	{18, 19}	26
max_div_20_2.txt	20	2	3	21.9961	21.9961	{9, 18, 19}	{9, 18, 19}	36
max_div_20_2.txt	20	2	4	39.5682	40.0023	{3, 9, 18, 19}	{3, 9, 2, 19}	248
max_div_20_2.txt	20	2	5	60.4301	62.8729	{3, 9, 11, 18, 19}	{3, 9, 2, 18, 19}	327
max_div_30_2.txt	30	3	2	11.6571	11.6571	{9, 28}	{9, 28}	42
max_div_30_2.txt	30	3	3	28.9443	28.9443	{2, 9, 28}	{2, 9, 28}	63
max_div_30_2.txt	30	3	4	52.7712	52.7712	{2, 9, 11, 28}	{2, 9, 11, 28}	222
max_div_30_2.txt	30	3	5	80.9102	80.9102	{2, 9, 11, 13, 28}	{2, 9, 11, 13, 28}	271
max_div_15_3.txt	15	3	2	13.2732	13.2732	{9, 12}	{9, 12}	37
max_div_15_3.txt	15	3	3	30.3241	31.8685	{5, 9, 12}	{5, 7, 12}	166
max_div_15_3.txt	15	3	4	58.7287	59.7638	{5, 9, 12, 14}	{5, 9, 12, 11}	311
max_div_15_3.txt	15	3	5	96.0858	96.0858	{4, 5, 9, 12, 14}	{4, 5, 9, 12, 14}	69
max_div_20_3.txt	20	3	2	11.5909	11.5909	{3, 17}	{3, 17}	30
max_div_20_3.txt	20	3	3	29.1574	29.1574	{3, 13, 17}	{3, 13, 17}	39
max_div_20_3.txt	20	3	4	56.6903	56.6903	{3, 13, 14, 17}	{3, 13, 14, 17}	63
max_div_20_3.txt	20	3	5	92.8298	92.8298	{3, 8, 13, 14, 17}	{3, 8, 13, 14, 17}	98
max_div_30_3.txt	30	3	2	12.6137	13.0737	{6, 17}	{7, 17}	110
max_div_30_3.txt	30	3	3	34.2905	34.2905	{6, 17, 24}	{6, 17, 24}	70
max_div_30_3.txt	30	3	4	63.702	63.702	{6, 14, 17, 24}	{6, 14, 17, 24}	97
max_div_30_3.txt	30	3	5	99.592	99.592	{6, 14, 15, 17, 24}	{6, 14, 15, 17, 24}	158

Figure 6: Tabla resultados LocalSearch con Solucción Inicial dada por Greedy Destructivo

5.2 GRASP

Para el algoritmo Grasp se van a hacer las tres fases, donde distinguimos:

- *Preprocesamiento*: Durante esta fase, se va a hacer un preprocesado. En este caso, el preprocesado que se ha hecho es generar una solución de tamaño m de forma aleatoria. Simplificando, se va a crear una solución de tamaño m con valores aleatorios.
- *Construcción*: Para ello se ha generado un número aleatorio entre el número total de nodos y luego hace tantas iteraciones como de grande sea ese elemento, creando el RCL, luego cogiendo un elemento aleatorio del RCL y por último añadiendo a la solucion.
- *Búsqueda Local*: Persigue buscar una mejor solución a partir de intercambios que se van haciendo con los elementos en la solución y los que no están en la misma. En caso de que haya una mejoría se actualiza la mejor solución.

Ahora, si observamos los resultados de la *figura 7* y *figura 8*. Se han dividido en dos tablas debido a que una se obtiene los resultados de los ficheros con k = 2, y en la otra aquellos que tiene valor de k = 3. A la hora de analizar los resultados obtenidos, en la *figura 7*, nos fijamos que los resultados obtenidos son mejores. Sobre todo esta diferencia debe ser más notable cuando el valor que define el tamaño del RCL es mayor o cuando hay un numero mayor de iteraciones. A su vez, en la *figura 8* vemos que también alcanza el óptimo en la mayor parte de los casos.

5.2.1 Tabla de resultados

Problema	n	K	m	Iter	—LRC—	z	S	CPU(µs)
max_div_15_2	15	2	2	10	2	11.8592	{9, 7}	328
max_div_15_2	15	2	2	10	3	11.8592	{7, 9}	444
max_div_15_2	15	2	2	20	2	11.8592	{7, 9}	762
max_div_15_2	15	2	2	20	3	11.8592	{7, 9}	386
max_div_15_2	15	2	3	10	2	27.3727	{1, 9, 7}	573
max_div_15_2	15	2	3	10	3	27.3727	{1, 7, 9}	641
max_div_15_2	15	2	3	20	2	27.3727	{1, 7, 9}	598
max_div_15_2	15	2	3	20	3	27.3727	{1, 9, 7}	603
max_div_15_2	15	2	4	10	2	49.8268	{1, 6, 9, 7}	962
max_div_15_2	15	2	4	10	3	49.8268	{7, 9, 6, 1}	956
max_div_15_2	15	2	4	20	2	49.8268	{9, 7, 1, 6}	892
max_div_15_2	15	2	4	20	3	49.8268	{6, 9, 7, 1}	970
max_div_15_2	15	2	5	10	2	79.1295	{9, 7, 4, 6, 1}	1977
max_div_15_2	15	2	5	10	3	78.9346	{1, 2, 6, 4, 7}	1741
max_div_15_2	15	2	5	20	2	79.1295	{1, 9, 7, 4, 6}	1438
max_div_15_2	15	2	5	20	3	79.1295	{1, 9, 7, 4, 6}	2192
max_div_20_2	20	2	2	10	2	8.51033	{19, 18}	412
max_div_20_2	20	2	2	10	3	8.51033	{19, 18}	404
max_div_20_2	20	2	2	20	2	8.51033	{19, 18}	484
max_div_20_2	20	2	2	20	3	8.51033	{19, 18}	517
max_div_20_2	20	2	3	10	2	21.9961	{19, 18, 9}	1046
max_div_20_2	20	2	3	10	3	21.9961	{9, 19, 18}	1080
max_div_20_2	20	2	3	20	2	21.9961	{9, 19, 18}	1344
max_div_20_2	20	2	3	20	3	21.9961	{19, 18, 9}	1440
max_div_20_2	20	2	4	10	2	39.8292	{2, 19, 9, 18}	1801
max_div_20_2	20	2	4	10	3	40.0023	{19, 3, 9, 2}	2730
max_div_20_2	20	2	4	20	2	39.8292	{9, 18, 19, 2}	1780
max_div_20_2	20	2	4	20	3	39.8292	{9, 18, 19, 2}	1532
max_div_20_2	20	2	5	10	2	63.6517	{19, 18, 9, 14, 2}	1863
max_div_20_2	20	2	5	10	3	63.6517	{2, 18, 19, 14, 2}	2378
max_div_20_2	20	2	5	20	2	63.6517	{2, 18, 19, 14, 2}	2314
max_div_20_2	20	2	5	20	3	63.6517	{18, 19, 9, 2, 14}	3681
max_div_30_2	30	2	2	10	2	11.6571	{28, 9}	613
max_div_30_2	30	2	2	10	3	11.6571	{9, 28}	772
max_div_30_2	30	2	2	20	2	11.6571	{9, 28}	906
max_div_30_2	30	2	2	20	3	11.6571	{9, 28}	558
max_div_30_2	30	2	3	10	2	28.9443	{2, 9, 28}	2080
max_div_30_2	30	2	3	10	3	28.9443	{2, 28, 9}	1137
max_div_30_2	30	2	3	20	2	28.9443	{28, 9, 2}	1936
max_div_30_2	30	2	3	20	3	28.9443	{9, 28, 2}	1565
max_div_30_2	30	2	4	10	2	52.7712	{2, 28, 11, 9}	4357
max_div_30_2	30	2	4	10	3	52.7712	{11, 28, 9, 2}	3473
max_div_30_2	30	2	4	20	2	52.7712	{9, 28, 2, 11}	1948
max_div_30_2	30	2	4	20	3	52.7712	{11, 9, 2, 28}	2300
max_div_30_2	30	2	5	10	2	80.9102	{13, 9, 28, 2, 11}	2971
max_div_30_2	30	2	5	10	3	80.9102	{13, 2, 9, 28, 11}	3269
max_div_30_2	30	2	5	20	2	80.9102	{2, 28, 9, 11, 13}	5867
max_div_30_2	30	2	5	20	3	80.9102	{2, 11, 28, 9, 13}	3052

Figure 7: Tabla resultados Grasp con K = 2

Problema	n	K	m	Iter	—LRC—	z	S	CPU(µs)
max_div_15_3	15	3	2	10	3	13.2732	{9, 12}	365
max_div_15_3	15	3	2	10	3	12.2732	{9, 12}	500
max_div_15_3	15	3	2	20	2	13.2732	{9, 12}	345
max_div_15_3	15	3	2	20	3	13.2732	{9, 12}	333
max_div_15_3	15	3	3	10	2	31.8685	{12, 7, 5}	636
max_div_15_3	15	3	3	10	3	31.8685	{7, 12, 5}	580
max_div_15_3	15	3	3	20	2	31.8685	{7, 12, 5}	803
max_div_15_3	15	3	3	20	3	31.8685	{12, 5, 7}	1086
max_div_15_3	15	3	4	10	2	59.7638	{5, 11, 9, 12}	1035
max_div_15_3	15	3	4	10	3	59.7429	{5, 4, 7, 12}	1139
max_div_15_3	15	3	4	20	2	59.7638	{12, 11, 9, 5}	1553
max_div_15_3	15	3	4	20	3	59.7638	{11, 5, 9, 12}	1939
max_div_15_3	15	3	5	10	2	96.0858	{4, 9, 5, 12, 14}	1762
max_div_15_3	15	3	5	10	3	93.9763	{12, 7, 5, 9, 11}	4018
max_div_15_3	15	3	5	20	2	96.0858	{14, 12, 9, 4, 5}	1679
max_div_15_3	15	3	5	20	3	96.0858	{5, 12, 9, 4, 14}	2278
max_div_20_3	20	2	2	10	2	11.8003	{14, 13}	444
max_div_20_3	20	3	2	10	3	11.8003	{14, 13}	495
max_div_20_3	20	3	2	20	2	11.8003	{14, 13}	1112
max_div_20_3	20	3	2	20	3	11.8003	{14, 13}	540
max_div_20_3	20	3	3	10	2	30.8727	{8, 14, 13}	992
max_div_20_3	20	3	3	10	3	30.8727	{8, 14, 13}	1929
max_div_20_3	20	3	3	20	2	29.4688	{11, 17, 14}	901
max_div_20_3	20	3	3	20	3	30.8727	{8, 14, 13}	767
max_div_20_3	20	3	4	10	2	56.6903	{3, 17, 14, 13}	1279
max_div_20_3	20	3	4	10	3	56.6903	{17, 14, 13, 3}	1623
max_div_20_3	20	3	4	20	2	56.6903	{13, 14, 17, 3}	1945
max_div_20_3	20	3	4	20	3	56.6903	{14, 17, 3, 13}	1438
max_div_20_3	20	3	5	10	2	92.8297	{8, 14, 3, 13, 17}	2600
max_div_20_3	20	3	5	10	3	92.8297	{8, 17, 3, 14, 13}	2743
max_div_20_3	20	3	5	20	2	92.8297	{14, 8, 3, 13, 17}	2164
max_div_20_3	20	3	5	20	3	92.8297	{3, 13, 17, 14, 8}	3000
max_div_30_3	30	3	2	10	2	13.0737	{7, 17}	599
max_div_30_3	30	3	2	10	3	13.0737	{7, 17}	939
max_div_30_3	30	3	2	20	2	13.0737	{7, 17}	875
max_div_30_3	30	3	2	20	3	13.0737	{17, 7}	822
max_div_30_3	30	3	3	10	2	34.2905	{6, 17, 24}	1564
max_div_30_3	30	3	3	10	3	34.2905	{6, 24, 17}	1106
max_div_30_3	30	3	3	20	2	34.2905	{6, 17, 24}	1473
max_div_30_3	30	3	3	20	3	34.2905	{17, 6, 24}	2430
max_div_30_3	30	3	4	10	2	63.5184	{14, 17, 7, 24}	2043
max_div_30_3	30	3	4	10	3	63.5184	{24, 14, 17, 7}	2419
max_div_30_3	30	3	4	20	2	63.702	{6, 17, 24, 14}	2153
max_div_30_3	30	3	4	20	3	63.702	{6, 14, 24, 17}	2735
max_div_30_3	30	3	5	10	2	99.592	{17, 6, 14, 15, 24}	8761
max_div_30_3	30	3	5	10	3	99.592	{14, 24, 15, 17, 6}	5130
max_div_30_3	30	3	5	20	2	99.592	{24, 17, 6, 15, 14}	7800
max_div_30_3	30	3	5	20	3	99.592	{14, 6, 15, 17, 24}	4580

Figure 8: Tabla resultados Grasp con K = 3

6 Ramificación y poda

El algoritmo de ramificación y poda se basa en encontrar la heurística sea mayor o igual a la mejor posible solución obtenible por esa rama. Una vez entendido esto, pasemos a explicar el algoritmo. Para empezar se va a signar una cota inferior. En nuestro caso esa cota va a venir definida por alguno de los tres algorimos, voraz constructivo, destructivo y GRASP. Entonces, lo primero que hay que hacer es inicializar el arbol, creando asi el primer nivel que va a formar parte de nuestro arbol. Ahora hemos de tener en cuenta la cota superior. En nuestro caso, esta va a estar definida a partir de combinar UB2 y UB3 (estos conceptos se extrayeron del artículo de Rafael Martí, Micael Gallego y Abraham Duarte). Mencionar que vamos a tener dos conjuntos *sel* que va a estar formado por la solución parcial con la que se va a explorar y *unsel* que contiene aquellos nodos mayores a los del conjunto de sel.
Ahora, hemos de tener en cuenta que vamos a definir:

$$z_1 = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(s_i, s_j)$$

$$z_2 = \sum_{i=1}^{k} \sum_{j=1}^{m-k} d(s_i, u_j)$$

$$z_3 = \sum_{i=1}^{m-k-1} \sum_{j=i+1}^{m-k} d(u_i, u_j)$$

De aquí entendemos que z_1 es el sumatorio de las distancias formadas por los nodos que están en *sel*, luego z_2 es el sumatorio de las distancias de los nodos de *sel*, con los mejores nodos posibles de *unsel* y, por último, z_3 es el sumatorio de las distancias de los nodos de *unsel* que formarian parte de la mejor solucion que se obtiene de la solucion parcial actual. Ahora, para encontrar un UB que esté ajustado utilizando un escaso esfuerzo computacional consiste en que básicamente sea la suma de los valores z_1 y z_2, asumiendo que nuestra cota superior va a ser la combinación de *UPB2* y *UPB3*, donde UPB2 va a formarse a partir de las p mejores aristas que están entre *sel* y *unsel* y UPB3 van a ser las mejores aristas de *unsel* y *unsel*.

Las mayores diferencias que se van a ver en las tablas que se van a mostrar a continuación, son en el número de nodos generados en el árbol y el tiempo de CPU que va a tardar en ejecutar el mismo. Estos valores van a mejorar cuanto más ajustada es la solución inicial que le hemos proporcionado al algoritmo. Tener en cuenta que se han hecho dos pruebas dependiendo de la cota. En el caso de las tablas *figura 9*, *figura 10* y *figura 11* se coge el nodo con la cota más pequeña. Sin embargo, en el caso de las tablas *figura 12*, *figura 13* y *figura 14* se coge el nodo con mayor profundidad

6.1 Estrategia de ramificación - cota más pequeña

6.1.1 Tabla de resultados (voraz constructivo)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{9, 7}	{7, 9}	1541
max_div_15_2.txt	15	2	3	317	25.7262	27.3727	{9, 7, 4}	{1, 7, 9}	17554
max_div_15_2.txt	15	2	4	1041	48.4139	49.8268	{9, 7, 4, 11}	{1, 6, 7, 9}	61274
max_div_15_2.txt	15	2	5	2586	73.5619	79.1295	{9, 7, 4, 11, 2}	{1, 4, 6, 7, 9}	139142
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{18, 19}	{18, 19}	2867
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{18, 19, 9}	{9, 18, 19}	43016
max_div_20_2.txt	20	2	4	1632	39.5682	40.0023	{18, 19, 9, 3}	{2, 3, 9, 19}	197531
max_div_20_2.txt	20	2	5	5805	61.2393	63.6517	{18, 19, 9, 3, 13}	{2, 9, 14, 18, 19}	671454
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{9, 28}	{9, 28}	28214
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{9, 28, 2}	{2, 9, 28}	213086
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{9, 28, 2, 11}	{2, 9, 11, 28}	1736775
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{9, 28, 2, 11, 13}	{2, 9, 11, 13, 28}	11602215
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{12, 9}	{9, 12}	1311
max_div_15_3.txt	15	3	3	190	30.3241	31.8685	{12, 9, 5}	{5, 7, 12}	11107
max_div_15_3.txt	15	3	4	521	59.7638	59.7638	{12, 9, 5, 11}	{5, 9, 11, 12}	28327
max_div_15_3.txt	15	3	5	1071	94.7487	96.0858	{12, 9, 5, 11, 14}	{4, 5, 9, 12, 14}	58476
max_div_20_3.txt	20	3	2	29	11.8003	11.8003	{13, 14}	{13, 14}	4421
max_div_20_3.txt	20	3	3	210	30.8727	30.8727	{13, 14, 8}	{8, 13, 14}	28541
max_div_20_3.txt	20	3	4	1030	56.5347	56.6903	{13, 14, 8, 3}	{3, 13, 14, 17}	128920
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{13, 14, 8, 3, 17}	{3, 8, 13, 14, 17}	289962
max_div_30_3.txt	30	3	2	62	13.0737	13.0737	{17, 7}	{7, 17}	36934
max_div_30_3.txt	30	3	3	595	33.8423	34.2905	{17, 7, 24}	{6, 17, 24}	326925
max_div_30_3.txt	30	3	4	3392	63.5184	63.702	{17, 7, 24, 14}	{6, 14, 17, 24}	1738553
max_div_30_3.txt	30	3	5	20584	99.5088	99.592	{17, 7, 24, 14, 15}	{6, 14, 15, 17, 24}	9843720

Figure 9: Tabla resultados Ramificación y Poda (cota más pequña) con Voraz Constructivo

6.1.2 Tabla de resultados (voraz destructivo)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{7, 9}	{7, 9}	1649
max_div_15_2.txt	15	2	3	335	23.7964	27.3727	{2, 7, 9}	{1, 7, 9}	18066
max_div_15_2.txt	15	2	4	1072	46.6528	49.8268	{2, 7, 9, 11}	{1, 6, 7, 9}	57666
max_div_15_2.txt	15	2	5	2586	73.5619	79.1295	{2, 4, 7, 9, 11}	{1, 4, 6, 7, 9}	131908
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{18, 19}	{18, 19}	2738
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{9, 18, 19}	{9, 18, 19}	42333
max_div_20_2.txt	20	2	4	1632	39.5682	40.0023	{3, 9, 18, 19}	{2, 3, 9, 19}	194786
max_div_20_2.txt	20	2	5	6078	60.4301	63.6517	{3, 9, 11, 18, 19}	{2, 9, 14, 18, 19}	709785
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{9, 28}	{9, 28}	28447
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{2, 9, 28}	{2, 9, 28}	217067
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{2, 9, 11, 28}	{2, 9, 11, 28}	1754885
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{2, 9, 11, 13, 28}	{2, 9, 11, 13, 28}	11541034
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{9, 12}	{9, 12}	1226
max_div_15_3.txt	15	3	3	190	30.3241	31.8685	{5, 9, 12}	{5, 7, 12}	10786
max_div_15_3.txt	15	3	4	541	58.7287	59.7638	{5, 9, 12, 14}	{5, 9, 11, 12}	30138
max_div_15_3.txt	15	3	5	1010	96.0858	96.0858	{4, 5, 9, 12, 14}	{4, 5, 9, 12, 14}	54596
max_div_20_3.txt	20	3	2	47	11.5909	11.8003	{3, 17}	{13, 14}	6663
max_div_20_3.txt	20	3	3	240	29.1574	30.8727	{3, 13, 17}	{8, 13, 14}	33063
max_div_20_3.txt	20	3	4	995	56.6903	56.6903	{3, 13, 14, 17}	{3, 13, 14, 17}	132579
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{3, 8, 13, 14, 17}	{3, 8, 13, 14, 17}	284196
max_div_30_3.txt	30	3	2	85	12.6137	13.0737	{6, 17}	{7, 17}	50147
max_div_30_3.txt	30	3	3	519	34.2905	34.2905	{6, 17, 24}	{6, 17, 24}	281092
max_div_30_3.txt	30	3	4	3342	63.702	63.702	{6, 14, 17, 24}	{6, 14, 17, 24}	1694778
max_div_30_3.txt	30	3	5	20336	99.592	99.592	{6, 14, 15, 17, 24}	{6, 14, 15, 17, 24}	9738991

Figure 10: Tabla resultados Ramificación y Poda (cota más pequña) con Voraz Destructivo

6.1.3 Tabla de resultados (GRASP)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{7, 9}	{7, 9}	1572
max_div_15_2.txt	15	2	3	273	27.3727	27.3727	{9, 7, 1}	{1, 7, 9}	15933
max_div_15_2.txt	15	2	4	1034	49.8268	49.8268	{6, 9, 7, 1}	{1, 6, 7, 9}	59118
max_div_15_2.txt	15	2	5	2391	79.1295	79.1295	{7, 9, 1, 6, 4}	{1, 4, 6, 7, 9}	120343
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{18, 19}	{18, 19}	3482
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{9, 18, 19}	{9, 18, 19}	44228
max_div_20_2.txt	20	2	4	1534	40.0023	40.0023	{3, 9, 19, 2}	{2, 3, 9, 19}	186969
max_div_20_2.txt	20	2	5	4744	63.6517	63.6517	{9, 18, 19, 14, 2}	{2, 9, 14, 18, 19}	586876
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{28, 9}	{9, 28}	34846
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{2, 28, 9}	{2, 9, 28}	226276
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{9, 28, 2, 11}	{2, 9, 11, 28}	2128694
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{13, 11, 28, 2, 9}	{2, 9, 11, 13, 28}	13203980
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{12, 9}	{9, 12}	1195
max_div_15_3.txt	15	3	3	165	30.9867	31.8685	{4, 12, 9}	{5, 7, 12}	9410
max_div_15_3.txt	15	3	4	521	59.7638	59.7638	{12, 9, 5, 11}	{5, 9, 11, 12}	28869
max_div_15_3.txt	15	3	5	1010	96.0858	96.0858	{9, 12, 4, 5, 14}	{4, 5, 9, 12, 14}	54695
max_div_20_3.txt	20	3	2	29	11.8003	11.8003	{13, 14}	{13, 14}	4220
max_div_20_3.txt	20	3	3	210	30.8727	30.8727	{14, 13, 8}	{8, 13, 14}	28871
max_div_20_3.txt	20	3	4	995	56.6903	56.6903	{3, 17, 14, 13}	{3, 13, 14, 17}	144163
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{13, 8, 17, 3, 14}	{3, 8, 13, 14, 17}	291448
max_div_30_3.txt	30	3	2	62	13.0737	13.0737	{7, 17}	{7, 17}	36788
max_div_30_3.txt	30	3	3	519	34.2905	34.2905	{6, 24, 17}	{6, 17, 24}	297443
max_div_30_3.txt	30	3	4	3392	63.5184	63.702	{14, 24, 7, 17}	{6, 14, 17, 24}	1964221
max_div_30_3.txt	30	3	5	20584	99.5088	99.592	{7, 17, 24, 14, 15}	{6, 14, 15, 17, 24}	10470692

Figure 11: Tabla resultados Ramificación y Poda (cota más pequña) con Grasp

6.2 Estrategia de ramificación - nodo más profundo

6.2.1 Tabla de resultados (voraz constructivo)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{9, 7}	{7, 9}	1538
max_div_15_2.txt	15	2	3	273	25.7262	27.3727	{9, 7, 4}	{1, 7, 9}	16721
max_div_15_2.txt	15	2	4	1034	48.4139	49.8268	{9, 7, 4, 11}	{1, 6, 7, 9}	58004
max_div_15_2.txt	15	2	5	2393	73.5619	79.1295	{9, 7, 4, 11, 2}	{1, 4, 6, 7, 9}	130146
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{18, 19}	{18, 19}	3044
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{18, 19, 9}	{9, 18, 19}	47816
max_div_20_2.txt	20	2	4	1534	39.5682	40.0023	{18, 19, 9, 3}	{2, 3, 9, 19}	201581
max_div_20_2.txt	20	2	5	5329	61.2393	63.6517	{18, 19, 9, 3, 13}	{2, 9, 14, 18, 19}	657563
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{9, 28}	{9, 28}	28771
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{9, 28, 2}	{2, 9, 28}	221320
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{9, 28, 2, 11}	{2, 9, 11, 28}	1801818
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{9, 28, 2, 11, 13}	{2, 9, 11, 13, 28}	11973726
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{12, 9}	{9, 12}	1220
max_div_15_3.txt	15	3	3	187	30.3241	31.8685	{12, 9, 5}	{5, 7, 12}	11291
max_div_15_3.txt	15	3	4	521	59.7638	59.7638	{12, 9, 5, 11}	{5, 9, 11, 12}	30856
max_div_15_3.txt	15	3	5	1115	94.7487	96.0858	{12, 9, 5, 11, 14}	{4, 5, 9, 12, 14}	69996
max_div_20_3.txt	20	3	2	29	11.8003	11.8003	{13, 14}	{13, 14}	4259
max_div_20_3.txt	20	3	3	210	30.8727	30.8727	{13, 14, 8}	{8, 13, 14}	30336
max_div_20_3.txt	20	3	4	1015	56.5347	56.6903	{13, 14, 8, 3}	{3, 13, 14, 17}	138380
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{13, 14, 8, 3, 17}	{3, 8, 13, 14, 17}	312131
max_div_30_3.txt	30	3	2	62	13.0737	13.0737	{17, 7}	{7, 17}	37866
max_div_30_3.txt	30	3	3	563	33.8423	34.2905	{17, 7, 24}	{6, 17, 24}	324889
max_div_30_3.txt	30	3	4	3379	63.5184	63.702	{17, 7, 24, 14}	{6, 14, 17, 24}	2063431
max_div_30_3.txt	30	3	5	20681	99.5088	99.592	{17, 7, 24, 14, 15}	{6, 14, 15, 17, 24}	10785267

Figure 12: Tabla resultados Ramificación y Poda (profundidad) con Voraz Constructivo

6.2.2 Tabla de resultados (voraz destructivo)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{7, 9}	{7, 9}	1456
max_div_15_2.txt	15	2	3	273	23.7964	27.3727	{2, 7, 9}	{1, 7, 9}	16148
max_div_15_2.txt	15	2	4	1064	46.6528	49.8268	{2, 7, 9, 11}	{1, 6, 7, 9}	62920
max_div_15_2.txt	15	2	5	2393	73.5619	79.1295	{2, 4, 7, 9, 11}	{1, 4, 6, 7, 9}	134417
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{18, 19}	{18, 19}	2807
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{9, 18, 19}	{9, 18, 19}	44337
max_div_20_2.txt	20	2	4	1534	39.5682	40.0023	{3, 9, 18, 19}	{2, 3, 9, 19}	205156
max_div_20_2.txt	20	2	5	5522	60.4301	63.6517	{3, 9, 11, 18, 19}	{2, 9, 14, 18, 19}	665610
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{9, 28}	{9, 28}	28596
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{2, 9, 28}	{2, 9, 28}	231411
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{2, 9, 11, 28}	{2, 9, 11, 28}	1794200
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{2, 9, 11, 13, 28}	{2, 9, 11, 13, 28}	11946205
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{9, 12}	{9, 12}	1239
max_div_15_3.txt	15	3	3	187	30.3241	31.8685	{5, 9, 12}	{5, 7, 12}	11539
max_div_15_3.txt	15	3	4	559	58.7287	59.7638	{5, 9, 12, 14}	{5, 9, 11, 12}	32362
max_div_15_3.txt	15	3	5	1010	96.0858	96.0858	{4, 5, 9, 12, 14}	{4, 5, 9, 12, 14}	57349
max_div_20_3.txt	20	3	2	45	11.5909	11.8003	{3, 17}	{13, 14}	6591
max_div_20_3.txt	20	3	3	243	29.1574	30.8727	{3, 13, 17}	{8, 13, 14}	33559
max_div_20_3.txt	20	3	4	995	56.6903	56.6903	{3, 13, 14, 17}	{3, 13, 14, 17}	140745
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{3, 8, 13, 14, 17}	{3, 8, 13, 14, 17}	295935
max_div_30_3.txt	30	3	2	85	12.6137	13.0737	{6, 17}	{7, 17}	49438
max_div_30_3.txt	30	3	3	519	34.2905	34.2905	{6, 17, 24}	{6, 17, 24}	284119
max_div_30_3.txt	30	3	4	3342	63.702	63.702	{6, 14, 17, 24}	{6, 14, 17, 24}	1742562
max_div_30_3.txt	30	3	5	20336	99.592	99.592	{6, 14, 15, 17, 24}	{6, 14, 15, 17, 24}	9960045

Figure 13: Tabla resultados Ramificación y Poda (profundidad) con Voraz Destructivo

6.2.3 Tabla de resultados (GRASP)

Problema	n	K	m	generados	z_0	z	S_0	S	CPU(µs)
max_div_15_2.txt	15	2	2	25	11.8592	11.8592	{7, 9}	{7, 9}	1480
max_div_15_2.txt	15	2	3	273	27.3727	27.3727	{1, 7, 9}	{1, 7, 9}	17720
max_div_15_2.txt	15	2	4	1034	49.8268	49.8268	{7, 9, 1, 6}	{1, 6, 7, 9}	61034
max_div_15_2.txt	15	2	5	2391	79.1295	79.1295	{6, 9, 7, 4, 1}	{1, 4, 6, 7, 9}	129263
max_div_20_2.txt	20	2	2	19	8.51033	8.51033	{19, 18}	{18, 19}	2793
max_div_20_2.txt	20	2	3	332	21.9961	21.9961	{19, 18, 9}	{9, 18, 19}	43324
max_div_20_2.txt	20	2	4	1534	40.0023	40.0023	{3, 9, 19, 2}	{2, 3, 9, 19}	191601
max_div_20_2.txt	20	2	5	4744	63.6517	63.6517	{9, 14, 19, 2, 18}	{2, 9, 14, 18, 19}	589436
max_div_30_2.txt	30	3	2	49	11.6571	11.6571	{28, 9}	{9, 28}	31006
max_div_30_2.txt	30	3	3	401	28.9443	28.9443	{9, 28, 2}	{2, 9, 28}	216681
max_div_30_2.txt	30	3	4	3519	52.7712	52.7712	{2, 11, 9, 28}	{2, 9, 11, 28}	1820289
max_div_30_2.txt	30	3	5	24912	80.9102	80.9102	{13, 9, 28, 2, 11}	{2, 9, 11, 13, 28}	11820014
max_div_15_3.txt	15	3	2	20	13.2732	13.2732	{9, 12}	{9, 12}	1229
max_div_15_3.txt	15	3	3	149	31.8685	31.8685	{5, 12, 7}	{5, 7, 12}	8860
max_div_15_3.txt	15	3	4	521	59.7638	59.7638	{5, 12, 9, 11}	{5, 9, 11, 12}	29979
max_div_15_3.txt	15	3	5	1010	96.0858	96.0858	{14, 4, 5, 12, 9}	{4, 5, 9, 12, 14}	56090
max_div_20_3.txt	20	3	2	29	11.8003	11.8003	{13, 14}	{13, 14}	4287
max_div_20_3.txt	20	3	3	210	30.8727	30.8727	{13, 14, 8}	{8, 13, 14}	29266
max_div_20_3.txt	20	3	4	995	56.6903	56.6903	{17, 3, 13, 14}	{3, 13, 14, 17}	127630
max_div_20_3.txt	20	3	5	2349	92.8298	92.8298	{3, 13, 14, 8, 17}	{3, 8, 13, 14, 17}	297617
max_div_30_3.txt	30	3	2	62	13.0737	13.0737	{7, 17}	{7, 17}	39353
max_div_30_3.txt	30	3	3	519	34.2905	34.2905	{6, 17, 24}	{6, 17, 24}	299700
max_div_30_3.txt	30	3	4	3342	63.702	63.702	{6, 17, 24, 14}	{6, 14, 17, 24}	1760960
max_div_30_3.txt	30	3	5	20336	99.592	99.592	{6, 15, 24, 17, 14}	{6, 14, 15, 17, 24}	12072823

Figure 14: Tabla resultados Ramificación y Poda (profundidad) con Grasp con LRC = 2 y max iter = 10

7 Conclusion

Una vez que hemos analizado como actuan todos los algoritmos en base a los valores que hemos definido, vemos que a la hora de usar este tipo de algorimos, aunque sea más costoso es mejor utilizar grasp, ramificación y poda, poruqe probablemente con ficheros de mayor tamaño, ya sean 50 nodos, se obtendrá la solución cercana a la óptima. Aunque estos algoritmos suelen requerir mucho más esfuerzo computacional, suelen brindar mejores resultados. Los algortimos vorazes, ambas implementaciones nos hemos fijado que los resultados son aproximados a los mejores y que su tiempo de cómputo suele ser mejor (tardan menos). Por ende, pienso que si se trabajara con problemas mayores, probablemente este comprotamiento seguirá siendo similar y se obtendrán buenos resultados, aunque tarde más. Hacer mención al artículo que facilitó la implementación del algoritmo de ramificación y poda. [1]

References

[1] Abraham Duarte Rafael Martí, Micael Gallego. *A branch and bound algorithm for the maximum diversity problem.* 2008.