Práctica 7: k means

Diseño y Análisis de Algoritmos

Helena García Díaz

18 de abril de 2023

Índice

1. Introducción

Durante esta práctica se tratará el problema de agrupamiento de puntos (también conocido como *clustering* en inglés). Dicho problema consiste en dividir un conjunto de datos en grupos o *clusters*, de tal manera que los elementos dentro de cada grupo sean similares entre sí y diferentes de los elementos de otros grupos.

En este caso, consideraremos el problema de agrupamiento como un conjunto de puntos de servicio que proveen a un conjunto de puntos clientes. La totalidad de los mismos vendrán determinados en las instancias previstas [2]. Salvo cuando se resuelva el problema con *k means* donde se calcularán los puntos de servicio como centroides de los mismos.

Entonces, nuestro problema consistirá en minimizar las distancias de los puntos clientes a los puntos de servicios. Para ello, se empleará la fórmula de la p-mediana 1, que será nuestra función objetivo para evaluar la calidad de las soluciones.

$$SSE = \sum_{r=1}^{k} \sum_{p \in C_r} [d_2(C_r, p)]^2 \tag{1}$$

Donde C_r es el punto de servicio r-ésimo y p son todos los puntos clientes asignados a ese punto de servicio.

2. Metodología

La búsqueda de grupos se ha implementado con diferentes estrategias con el objetivo de estudiar y comparar su eficiencia y eficacia en la calidad de las soluciones.

Se ha implementado el agrupamiento con 3 algoritmos diferentes, que se explicarán uno a uno a continuación. En estos algoritmos se han considerado un número inicial de grupos estableciendo el número en el 10 % del número total de puntos, limitando a un mínimo de dos grupos.

- K means.
- GRASP *(Greedy Randomized Adaptive Search).*
- GVNS *(General Variable Neighborhood Search).*

2.1. Algoritmos

2.1.1. k means

Para realizar el agrupamiento con *k-means*, en lugar de trabajar con puntos de la instancia que actúen como puntos de servicio, se trabajará calculando los centroides del cluster formado. Se comienza eligiendo x puntos aleatorios que

actuarán como centroides y se establece como condición de parada el hecho de que estos centroides no cambien de una iteración a otra.

Entonces, se comienza a iterar. Primero, se forman los grupos entorno a los centroides dados y se asigna cada punto al cluster cuyo centroide esté más cerca. Luego, se pasan a calcular los nuevos centroides. Cuando estos centroides no cambien con respecto a la iteración anterior, se parará el proceso y se devolverá la solución final.

Se debe destacar que en este tipo de algoritmo, el número de puntos de servicio no cambiará de una iteración a otra, sino que quedan definidos de manera invariable desde el principio.

2.1.2. GRASP *(Greedy Randomized Adaptive Search)*

En este caso, se realiza sólo una fase constructiva empleando una LRC (Lista Restringida de Candidatos).

Para formar los grupos, se comienza con una solución vacía y se elige primero un punto al azar de entre todos los dados. Se podría determinar haciendo consideraciones iniciales tales como elegir el punto más "*centrado*"de todos, pero se ha elegido dejarlo al azar, para no restringir las posibles soluciones que se obtengan.

Una vez tenemos al menos un punto en el *cluster* que reúne a los puntos de servicio, se calcula la distancia a todos los puntos restantes, considerando la distancia mínima al borde de ese *cluster* de puntos de servicio. Una vez se tienen todas esas distancias, se ordenan y se crea una LRC con los x puntos más alejados. De esa lista, se escogerá un punto al azar que pasará a formar parte del *cluster* de puntos de servicio. Este procedimiento se repetirá hasta que se llegue a la cantidad de puntos de servicio que se estableció inicialmente.

A partir de esos puntos de servicio establecidos, se hará un procesamiento por el que los puntos clientes se asignan al punto de servicio más cercano y se devuelve la solución dada.

A la hora de hacer el estudio del algoritmo, se irá variando el tamaño de la LRC para ver cómo o si esta tiene influencia en los resultados obtenidos.

2.1.3. GVNS *(General Variable Neighborhood Search)*

Este algoritmo consta de dos partes principales: La fase de *shaking* y la fase de mejora. Es una metaheurística que utiliza diferentes estructuras de vecindad para explorar el espacio de soluciones,

Primero se emplea la fase constructiva que mencionamos en *GRASP* para construir una solución inicial. A partir de ella, se realizarán los siguientes pasos.

Primero, se establece un tamaño para el vecindario. Este determinará el número de puntos que se verán alterados por nuestras estructuras de entorno. Es decir, que si en esta fase el valor de k es de 1, estaré intercambiando dos puntos de servicio en la solución cada vez. Si es 2, se aumentará el intercambio a dos puntos de servicio cada vez. Así, hasta el k_{max} que hayamos considerado.

En este caso, se harán diferentes ejecuciones variando este valor, para ver cómo influye en el resultado obtenido.

Una vez determinado el entorno definido por k, se realiza un salto aleatorio dentro del mismo.

Luego, pasamos a la fase de mejora. En ella, se realizarán una búsqueda local siguiendo un algoritmo de *Variable Neighborhood descent* (*VND* por sus siglas en inglés).

En ella, se comienza explorando una de las estructuras de entorno. En este caso, la de intercambio, hasta que se encuentra la combinación de puntos de servicio en la misma que minimiza nuestra función objetivo, realizando una búsqueda local. Luego, se pasa a la siguiente estructura de entorno. En nuestro caso, la de inserción. Si se encuentra una solución mejor, se vuelve a comenzar por la primera. Si no, se pasa a nuestra tercera estructura de entorno, la de eliminación. Si en ella se encuentra una solución mejor, se vuelve a comenzar el proceso por la primera estructura de entorno. Si no, se habrá llegado a un óptimo local con respecto a las tres estructuras de entorno.

Luego, si no se ha alcanzado el valor máximo de k, se aumenta este en uno y se vuelve a comenzar.

Para que el algoritmo no acabe haciendo punto de servicio por cada punto en la solución, se ha sumado un factor de penalización en el cálculo de nuestra función objetivo de *penalización * número de puntos de servicio*. La cantidad de *penalización*, se ha ido variando para ver cómo esta influía en las soluciones obtenidas.

2.2. Estructura de la solución

Nuestra solución estará compuesta por los puntos de servicio escogidos. A partir de ellos, se escogerá el grupo correspondiente a cada punto.

En el código, esta se verá representada por una clase *Solution*<*T*> que contará con un atributo *service_points_* constituido por un *clusters* donde cada *cluster* será un vector de vectores. De esta manera, se pondrán almacenar puntos de diferentes dimensiones sin problemas.

En la misma clase *Solution*, se encuentran los métodos correspondientes para manejar la solución, como la evaluación de la función objetivo. Dicha función objetivo será la fórmula de la p-mediana entre los puntos de servicio y los puntos que formen parte del *cluster* asignado a ese punto de servicio.

2.3. Código

En cuanto al código, se han desarrollado tres clases principales:

- *Problem*: Para almacenar toda la información sobre las instancias y trabajar con ellas. En ella se lee una instancia desde archivo, se almacena la nube de puntos y la cantidad de los mismos. Así como diversos métodos para trabajar con los mismos.

- *Solution*: Para almacenar y trabajar con la solución, tal y como se ha mencionado anteriormente.
- *Algorithm*: Clase abstracta que implementa un método virtual *run*, para ejecutar los diferentes algoritmos que heredan de ella.

Heredando de *Algorithm* se ha implementado las clases *Kmeans* para implementar dicho algoritmo y la clase *GRASP* que cuenta con atributos para controlar las estructuras de entorno empleadas y el límite de las mismas en la fase de *shaking*. Además, cuenta con numeros métodos para llevar a cabo la implementación de los algoritmos.

En este caso, se ha implementado varias veces el método *run*, para ser intercambiado según la ejecución que se desee. Uno para GRASP empleando su fase constructiva, otro para probar sólo la búsqueda local con una única estructura de entorno, partiendo de la fase constructiva de GRASP, y un último que implementa el algoritmo GVNS.

Además, se han implementado dos clases para facilitar y manejar la realización de varias ejecuciones, pudiendo almacenar los resultados de las mismas en un archivo *csv*. Dichas clases son *Generador* e *Iteración*.

Además, se ha implementado un programa sencillo en *python* para dada una única ejecución, se puedan representar, a partir del archivo de texto generado donde se almacenan los puntos de servicios y los *clusters* de puntos clientes asociados a los mismos, el agrupamiento de los mismos. De esta forma, se puede corroborar que el agrupamiento se está realizando de manera correcta.

Por último, todo está manejado por un *makefile* que permite ejecutar los tests creados para controlar las fórmulas de referencia empleadas y ejecutar, compilando antes o no, el proyecto.

3. Resultados

Para hacer un estudio sobre los resultados obtenidos, se han tenido en cuenta principalmente el valor de la función objetivo y el tiempo de ejecución del mismo.

3.1. kmeans

Si se hace un promedio del resultado de la función objetivo en el conjunto de ejecuciones incluido en el apéndice A, para cada instancia, se puede ver en la figura 1 un resultado promedio de entre 200 y 400 para el problema 1 y de entre 400 y 600 para el problema 2.

En cuanto al tiempo de ejecución, se puede observar en la figura 1 que para el problema 1 está en el orden de 20-40 microsegundos y de 40-60 microsegundos para el problema 2.

Conociendo el tamaño de la instancia (apéndice A) y observando la representación 1 se puede concluir que para instancias más grandes, con kmeans se obtiene un valor mayor para la función objetivo así como un tiempo mayor de ejecución.

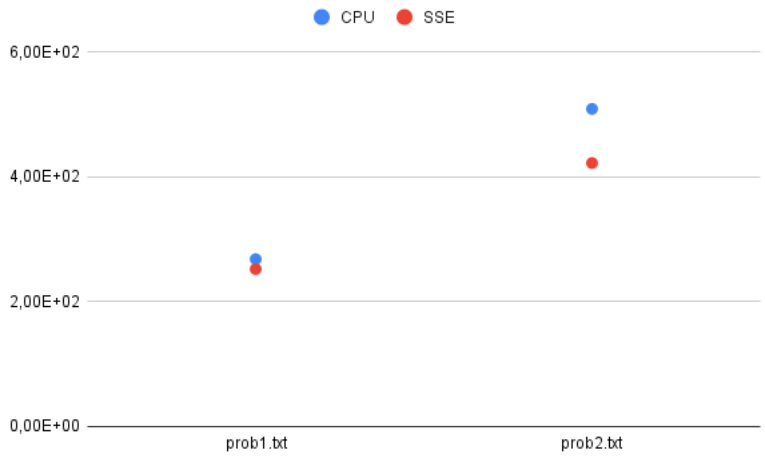

Figura 1: Promedio del valor de la función objetivo y del tiempo de ejecución para cada instancia

3.2. GRASP

Si representamos los datos obtenidos en el apéndice A del promedio del resultado de la función objetivo en función del número de elementos incluidos en la LRC, se puede observar que tanto para el problema 1 (figura 2) como para el dos (figura 3), estos influyen en el resultado obtenido. Se puede observar un valor similar cuando la LRC tiene tamaño 2 o 4 y un valor inferior cuando esta tiene tamaño 3.

Del mismo modo se observa en las figuras 2 y 3 una correspondencia directamente proporcional en el tiempo de ejecución y el resultado de la función objetivo para ambos problemas.

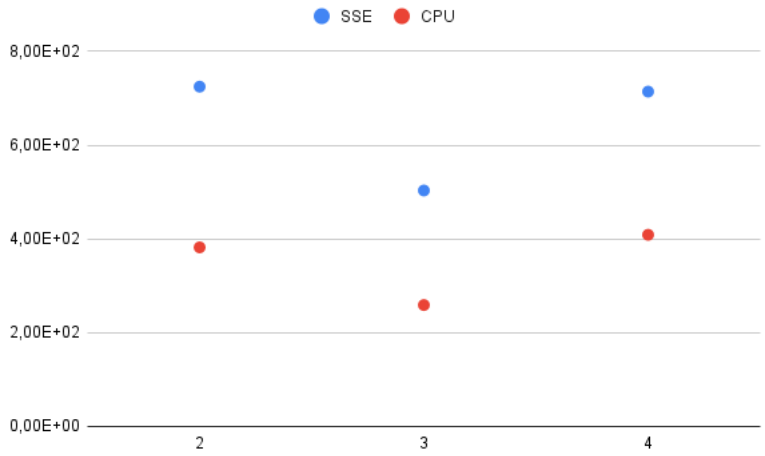

Figura 2: Problema 1. Valor de la función objetivo y Tiempo de ejecución en función del tamaño de la LRC

En cuanto a los órdenes de magnitud, se observa que el valor de la función objetivo para los diferentes tamaños de la LRC es superior en el problema 1 (figura 2) con respecto al problema 2 (figura 3).

Si nos fijamos en el tiempo de cómputo ocurre lo contrario, algo comprensible ya que el número de puntos en el problema 2 es mayor que en el problema 1 (20 puntos frente a 15).

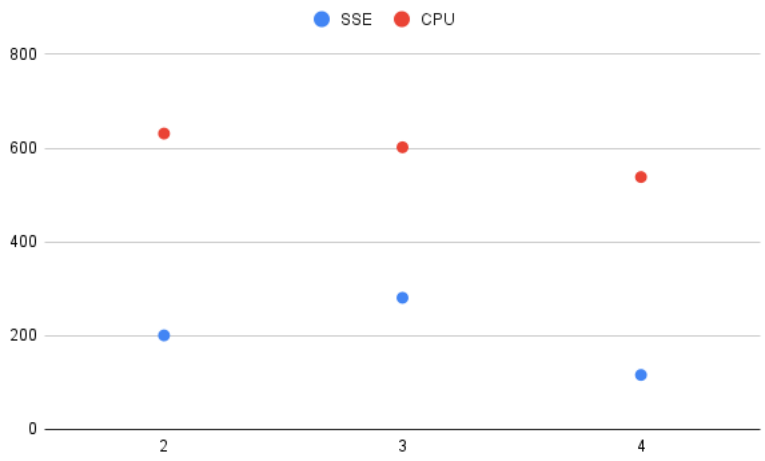

Figura 3: Problema 2. Valor de la función objetivo y Tiempo de ejecución en función del tamaño de la LRC

3.3. GVNS

Como se puede observar en los resultados de las figuras 4, 6, 5 y 7, el límite que establecemos en K_{max} no influye sustancialmente en la calidad de las soluciones que obtenemos, pero sí lo hace en el tiempo de ejecución.

Se puede suponer que esto se debe al tamaño de las instancias con las que trabajamos, y a que en la primera iteración, con las estructuras de entorno definidas, es capaz de explorar todo el espacio de soluciones.

En cuanto al factor de penalización, se puede observar en los apçendices C, D, E y F su influencia directa con el número de grupos formados por el algoritmo. Cuanto menor es, mayor será el número de puntos de servicio que añada a la solución.

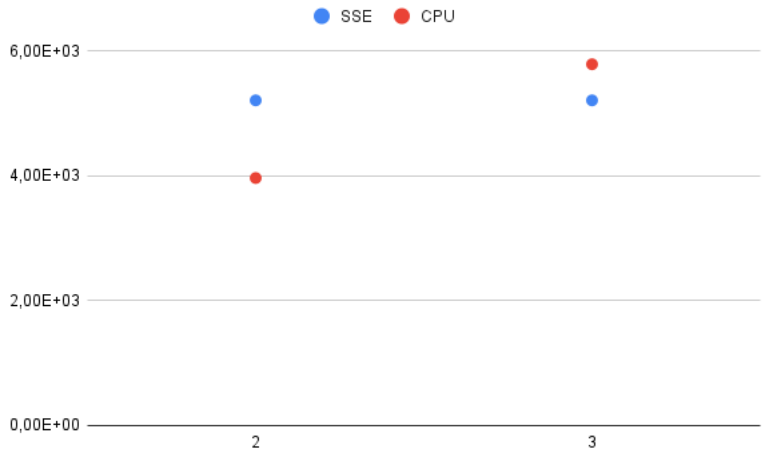

Figura 4: Problema 1. Tiempo de ejecución y valor de la función objetivo en función del valor de Kmax con factor de penalización de 4

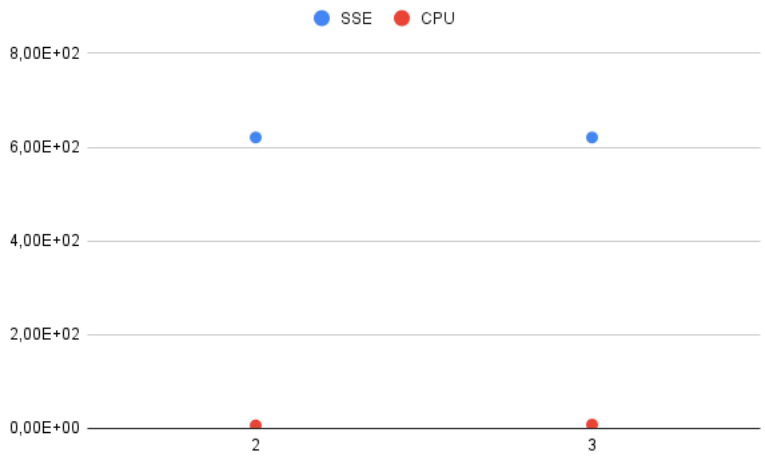

Figura 5: Problema 1. Tiempo de ejecución y valor de la función objetivo en función del valor de Kmax con factor de penalización de 6

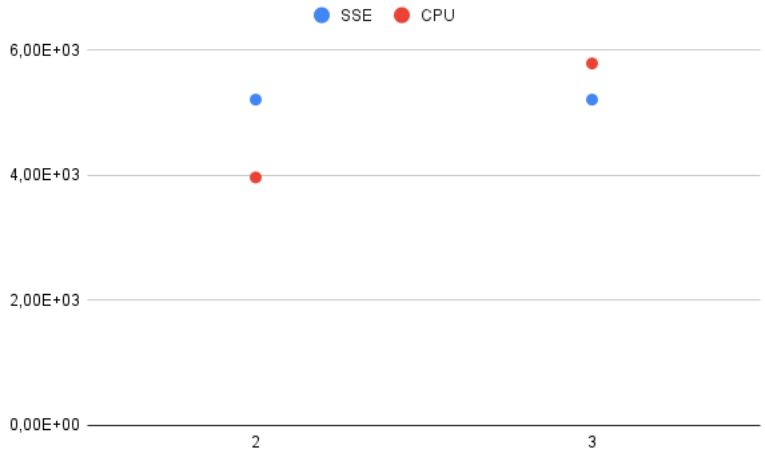

Figura 6: Problema 2. Tiempo de ejecución y valor de la función objetivo en función del valor de Kmax con factor de penalización de 4

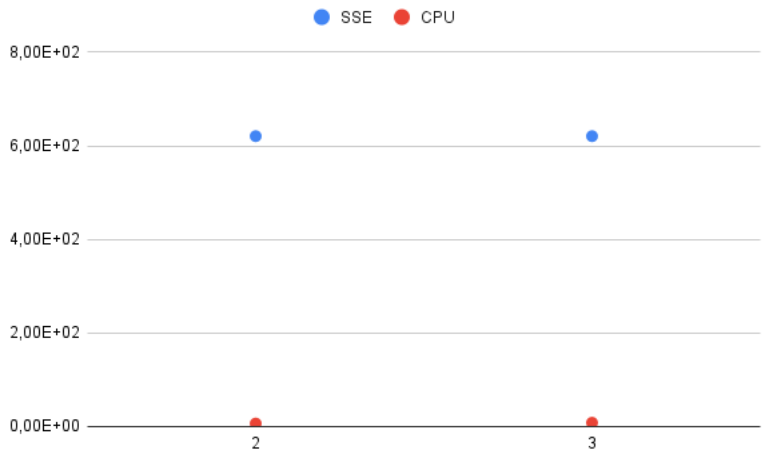

Figura 7: Problema 2. Tiempo de ejecución y valor de la función objetivo en función del valor de Kmax con factor de penalización de 6

4. Discusión

Si comparamos los resultados obtenidos por los diferentes algoritmos, se puede observar que kmeans obtiene un valor para la función objetivo similar a los de GRASP cuando el tamaño de la LRC es de 2 o 4, siendo peor comparado con GRASP empleando una LRC de tamaño 3 en el problema 1.

Sin embargo, para el problema 2 se obtienen valores mucho mejores empleando GRASP con cualquier tamaño de lista antes que con kmeans. En concreto, el tamaño de lista que mejor resultado da para esta instancia es el de 4.

Si comparamos ahora con el GVNS, se observa un cambio drástico en el descenso del valor de la función objetivo. Por ejemplo, para el problema 1, el promedio del mejor valor de la función objetivo usando GRASP es del orden de 500 (apéndice A), mientras que con GVNS con un factor de penalización de 4 y GVNS con un factor de penalización de 4, se disminuye este valor hasta el orden de 50 (apéndice C). Con un factor de penalización de 6 también disminuye hasta el orden de 60 (apéndice D), aunque no es tan bueno como el anterior.

5. Conclusiones

Conociendo el tamaño de la instancia (apéndice A) y observando la representación 1 se puede concluir que para instancias más grandes, con kmeans se obtiene un valor mayor para la función objetivo así como un tiempo mayor de ejecución.

En general, se obtienen mejores resultados para la función objetivo con el algoritmo GRASP con un tamaño de la LRC de 3 para el problema 1 y de 4 para el problema 2, antes que empleando el algoritmo kmeans.

Del mismo modo, se observa un aumento significativo en la calidad de las soluciones empleando GVNS antes que cualquiera de los dos algoritmos anteriores, ya que no sólo se buscan óptimos locales, sino que también se diversifica en el espacio de soluciones.

Sería interesante probar las ejecuciones con instancias mucho mayores, y llevar a los algoritmos al límite para ver como avanzan las curvas del tiempo de ejecución y del valor de la función objetivo cuando la cantidad de puntos es significativamente mayor.

Sería interesante también realizar un mayor número de representaciones gráficas comparando ambos algoritmos para ver de manera más clara una comparativa entre ellos, aunque con los mostrados en este informe se puede observar las diferencias entre ellos.

En conclusión, queda patente el poder de cómputo del GVNS, ya que se obtienen soluciones con un valor significativamente mejor para la función objetivo sin sacrificar tiempo de cómputo.

Referencias

[1] Subtitle with the maketitle page? (s. f.). TeX - LaTeX Stack Exchange. https://tex.stackexchange.com/questions/50182/subtitle-with-the-maketitle-page

[2] Campus Ingeniería y Tecnología 22/23: Entrar al sitio. (s. f.). https://campusingenieriaytecnologia2223.ull.es/mod/folder/view.php?id=28219

[3] Informe en Overleaf https://www.overleaf.com/read/yrxrkhwwjrbc

A.

Tabla de tiempo mínimo de procesamiento para cada instancia. Algoritmo k means

Problema	**m**	**K**	**Ejecución**	**SSE**	**CPU**
prob1.txt	15	2	1	247.17	2,47E+02
prob1.txt	15	2	2	240.477	2,67E+02
prob1.txt	15	2	3	244.593	3,03E+02
prob1.txt	15	2	4	274.153	2,25E+02
prob1.txt	15	2	5	260.826	2,85E+02
prob1.txt	15	2	6	266.539	2,10E+01
prob1.txt	15	2	7	264.774	4,72E+02
prob1.txt	15	2	8	240.477	2,33E+02
prob1.txt	15	2	9	244.593	3,07E+02
prob1.txt	15	2	10	240.477	3,23E+02
prob2.txt	20	2	1	411.14	5,40E+01
prob2.txt	20	2	2	419.484	9,64E+02
prob2.txt	20	2	3	421.838	3,65E+02
prob2.txt	20	2	4	432.108	7,04E+02
prob2.txt	20	2	5	421.838	5,99E+02
prob2.txt	20	2	6	432.108	4,55E+02
prob2.txt	20	2	7	419.484	5,19E+02
prob2.txt	20	2	8	419.484	6,49E+02
prob2.txt	20	2	9	418.17	3,77E+02
prob2.txt	20	2	10	428.465	4,06E+02

B. Tabla de tiempo mínimo de procesamiento para cada instancia del problema 1. Algoritmo GRASP

Problema	**m**	**K**	**|LRC|**	**Ejecución**	**SSE**	**CPU**
prob1.txt	15	2	2	1	836.974	4,54E+02
prob1.txt	15	2	2	2	705.142	5,15E+02
prob1.txt	15	2	2	3	705.643	4,15E+02
prob1.txt	15	2	2	4	731.665	3,89E+02
prob1.txt	15	2	2	5	731.665	3,92E+02
prob1.txt	15	2	2	6	773.289	4,09E+02
prob1.txt	15	2	2	7	705.643	4,29E+02
prob1.txt	15	2	2	8	668.362	3,94E+02
prob1.txt	15	2	2	9	658.395	4,10E+01
prob1.txt	15	2	2	10	731.102	3,82E+02
prob1.txt	15	2	3	1	710.883	4,42E+02
prob1.txt	15	2	3	2	85.042	4,00E+00
prob1.txt	15	2	3	3	635.517	4,60E+01
prob1.txt	15	2	3	4	694.912	4,10E+01
prob1.txt	15	2	3	5	85.042	3,99E+02
prob1.txt	15	2	3	6	85.042	3,85E+02
prob1.txt	15	2	3	7	731.665	4,22E+02
prob1.txt	15	2	3	8	710.883	4,15E+02
prob1.txt	15	2	3	9	635.517	3,95E+02
prob1.txt	15	2	3	10	658.395	4,10E+01
prob1.txt	15	2	4	1	770.666	4,12E+02
prob1.txt	15	2	4	2	710.883	4,05E+02
prob1.txt	15	2	4	3	694.912	4,12E+02
prob1.txt	15	2	4	4	710.567	3,95E+02
prob1.txt	15	2	4	5	765.667	3,89E+02
prob1.txt	15	2	4	6	668.362	4,03E+02
prob1.txt	15	2	4	7	728.925	4,03E+02
prob1.txt	15	2	4	8	705.643	4,07E+02
prob1.txt	15	2	4	9	770.666	4,05E+02
prob1.txt	15	2	4	10	616.164	4,57E+02

C. Tabla de tiempo mínimo de procesamiento para cada instancia del problema 2. Algoritmo GRASP

Problema	m	K	|LRC|	Ejecución	SSE	CPU
prob2.txt	20	2	2	1	118.672	7,41E+02
prob2.txt	20	2	2	2	114.044	6,40E+01
prob2.txt	20	2	2	3	124.244	7,71E+02
prob2.txt	20	2	2	4	109.944	6,93E+02
prob2.txt	20	2	2	5	119.529	6,34E+02
prob2.txt	20	2	2	6	110.773	7,82E+02
prob2.txt	20	2	2	7	124.244	6,88E+02
prob2.txt	20	2	2	8	116.739	6,55E+02
prob2.txt	20	2	2	9	115.554	6,46E+02
prob2.txt	20	2	2	10	952.498	6,38E+02
prob2.txt	20	2	3	1	9,52E+05	626
prob2.txt	20	2	3	2	1,02E+05	697
prob2.txt	20	2	3	3	1,19E+05	641
prob2.txt	20	2	3	4	1,34E+05	629
prob2.txt	20	2	3	5	1,14E+05	676
prob2.txt	20	2	3	6	9,47E+05	67
prob2.txt	20	2	3	7	1,16E+05	655
prob2.txt	20	2	3	8	1,08E+05	694
prob2.txt	20	2	3	9	1,24E+05	642
prob2.txt	20	2	3	10	9,22E+04	692
prob2.txt	20	2	4	1	1,09E+05	644
prob2.txt	20	2	4	2	1,16E+05	671
prob2.txt	20	2	4	3	108.37	681
prob2.txt	20	2	4	4	1,20E+05	688
prob2.txt	20	2	4	5	1,16E+05	634
prob2.txt	20	2	4	6	1,24E+05	7
prob2.txt	20	2	4	7	1,20E+05	667
prob2.txt	20	2	4	8	1,08E+05	67
prob2.txt	20	2	4	9	1,17E+05	682
prob2.txt	20	2	4	10	1,24E+05	643

D. Tabla de tiempo mínimo de procesamiento para cada instancia. Algoritmo GVNS. Penalización de 4

Problema	m	K	KmaxVNS	Ejecución	SSE	CPU
prob1.txt	15	9	2	1	5,21E+02	3,82E+02
prob1.txt	15	9	2	2	5,21E+02	4,95E+02
prob1.txt	15	9	2	3	5,21E+02	4,86E+02
prob1.txt	15	9	2	4	5,21E+02	3,91E+01
prob1.txt	15	9	2	5	5,21E+02	4,89E+02
prob1.txt	15	9	2	6	5,21E+02	4,05E+02
prob1.txt	15	9	2	7	5,21E+02	4,00E+02
prob1.txt	15	9	2	8	5,21E+02	3,83E+02
prob1.txt	15	9	2	9	5,21E+02	4,73E+02
prob1.txt	15	9	2	10	5,21E+02	4,16E+02
prob1.txt	15	9	3	1	5,21E+02	5,76E+02
prob1.txt	15	9	3	2	5,21E+02	5,72E+02
prob1.txt	15	9	3	3	5,21E+02	5,45E+02
prob1.txt	15	9	3	4	5,21E+02	5,72E+02
prob1.txt	15	9	3	5	5,21E+02	5,05E+02
prob1.txt	15	9	3	6	5,21E+02	5,41E+02
prob1.txt	15	9	3	7	5,21E+02	6,59E+02
prob1.txt	15	9	3	8	5,21E+02	5,34E+02
prob1.txt	15	9	3	9	5,21E+02	8,02E+02
prob1.txt	15	9	3	10	5,21E+02	4,86E+02

E. Tabla de tiempo mínimo de procesamiento para cada instancia. Algoritmo GVNS. Penalización de 6

Problema	m	K	KmaxVNS	Ejecución	SSE	CPU
prob1.txt	15	4	2	1	6,21E+01	6,77E-01
prob1.txt	15	4	2	2	6,21E+01	7,66E-01
prob1.txt	15	4	2	3	6,21E+01	5,71E-01
prob1.txt	15	4	2	4	6,21E+01	6,04E-01
prob1.txt	15	4	2	5	6,21E+01	6,02E-01
prob1.txt	15	4	2	6	6,21E+01	7,18E-01
prob1.txt	15	4	2	7	6,21E+01	6,45E-01
prob1.txt	15	4	2	8	6,21E+01	6,20E-01
prob1.txt	15	4	2	9	6,21E+01	7,65E-01
prob1.txt	15	4	2	10	6,21E+01	5,14E-01
prob1.txt	15	4	3	1	6,21E+01	7,65E-01
prob1.txt	15	4	3	2	6,21E+01	7,65E-01
prob1.txt	15	4	3	3	6,21E+01	8,75E-01
prob1.txt	15	4	3	4	6,21E+01	7,24E-01
prob1.txt	15	4	3	5	6,21E+01	8,23E-01
prob1.txt	15	4	3	6	6,21E+01	8,15E-01
prob1.txt	15	4	3	7	6,21E+01	7,86E-01
prob1.txt	15	4	3	8	6,21E+01	9,18E-01
prob1.txt	15	4	3	9	6,21E+01	8,40E-01
prob1.txt	15	4	3	10	6,21E+01	8,39E-01

F. Tabla de tiempo mínimo de procesamiento para cada instancia. Algoritmo GVNS. Penalización de 4

Problema	m	K	KmaxVNS	Ejecución	SSE	CPU
prob2.txt	20	13	2	1	7,12E+02	1,58E+02
prob2.txt	20	13	2	2	7,12E+02	1,78E+02
prob2.txt	20	13	2	3	7,12E+02	1,56E+02
prob2.txt	20	13	2	4	7,12E+02	1,50E+02
prob2.txt	20	13	2	5	7,12E+02	1,65E+02
prob2.txt	20	13	2	6	7,12E+02	1,93E+02
prob2.txt	20	13	2	7	7,12E+02	1,65E+02
prob2.txt	20	13	2	8	7,12E+02	1,68E+00
prob2.txt	20	13	2	9	7,12E+02	1,70E+02
prob2.txt	20	13	2	10	7,12E+02	1,52E+02
prob2.txt	20	9	3	1	7,12E+02	2,52E+02
prob2.txt	20	9	3	2	7,12E+02	2,34E+02
prob2.txt	20	9	3	3	7,12E+02	2,66E+00
prob2.txt	20	9	3	4	7,12E+02	2,42E+02
prob2.txt	20	9	3	5	7,12E+02	2,53E+02
prob2.txt	20	9	3	6	7,12E+02	2,52E+02
prob2.txt	20	9	3	7	7,12E+02	2,85E+02
prob2.txt	20	9	3	8	7,12E+02	2,25E+02
prob2.txt	20	9	3	9	7,12E+02	2,38E+02
prob2.txt	20	9	3	10	7,12E+02	3,28E+02

G. Tabla de tiempo mínimo de procesamiento para cada instancia. Algoritmo GVNS. Penalización de 6

Problema	m	K	KmaxVNS	Ejecución	SSE	CPU
prob2.txt	20	5	2	1	8,88E+01	1,16E+00
prob2.txt	20	5	2	2	8,88E+01	1,15E+00
prob2.txt	20	5	2	3	8,88E+01	2,17E+00
prob2.txt	20	5	2	4	8,88E+01	1,87E+00
prob2.txt	20	5	2	5	8,88E+01	1,60E+00
prob2.txt	20	5	2	6	8,88E+01	1,58E+00
prob2.txt	20	5	2	7	8,88E+01	1,39E+00
prob2.txt	20	5	2	8	8,88E+01	1,38E+00
prob2.txt	20	5	2	9	8,88E+01	1,74E+00
prob2.txt	20	5	2	10	8,88E+01	1,14E+00
prob2.txt	20	5	3	1	8,88E+01	1,52E+00
prob2.txt	20	5	3	2	8,88E+01	2,01E+00
prob2.txt	20	5	3	3	8,88E+01	1,62E+00
prob2.txt	20	5	3	4	8,88E+01	1,66E+00
prob2.txt	20	5	3	5	8,88E+01	1,82E+00
prob2.txt	20	5	3	6	8,88E+01	1,72E+00
prob2.txt	20	5	3	7	8,88E+01	1,80E+00
prob2.txt	20	5	3	8	8,88E+01	1,85E+00
prob2.txt	20	5	3	9	8,88E+01	1,78E+00
prob2.txt	20	5	3	10	8,88E+01	1,86E+00