TRABAJO FIN DE GRADO

CURSO ACADÉMICO 2022-2023

ALGORITMO GENÉTICO PARA EL CÁLCULO DE DENDROGRAMAS CON SELECCIÓN DE CARACTERÍSTICAS

UNIVERSIDAD MIGUEL HERNÁNDEZ

FACULTAD DE CIENCIAS SOCIALES Y JURÍDICAS DE ELCHE

Autor: Antonio Ramos Cánovas
Tutor: Mercedes Landete Ruiz y Marina Leal Palazón

Contenido

1. Introducción

En el ámbito de la optimización matemática y el análisis de datos, este trabajo aborda un problema multifacético: la clasificación de individuos mediante el empleo de técnicas de clustering jerárquico junto con la selección de características. Además, introduce una contribución innovadora en forma de un algoritmo genético diseñado específicamente para el cálculo eficiente de dendrogramas con selección de características.

La clasificación de individuos mediante el empleo de clustering jerárquico es un método ya conocido por los estudiantes de Estadística Empresarial pues se trabaja en asignaturas como Minería de datos o Técnicas estadísticas de análisis de mercados. Consiste en organizar los datos de manera significativa y funcional para facilitar la comprensión, la predicción, la personalización y la toma de decisiones sobre una base de datos de un problema. Por otro lado, el desarrollo de un algoritmo heurístico es algo en lo que no se profundiza en el grado por lo que puede ser considerado algo novedoso para el estudiante y, también, importante para conocer una nueva rama del árbol de la estadística.

La selección de características es un proceso fundamental para evitar el sobreajuste y reducir el tamaño de las bases de datos sin una pérdida significativa de información, por lo que puede resultar de utilidad para aplicarlo al clustering jerárquico Los dendrogramas son representaciones gráficas de algoritmos de clustering jerárquico que, en el caso del clustering de Single Linkage (enlace único), pueden interpretarse como árboles de expansión mínima en la red completa definida por la base de datos (Martine Labbé, Mercedes Landete, Marina Leal, 2022).

En este trabajo, presentamos el problema que determina simultáneamente un conjunto de características y un dendrograma con restricciones del problema de árbol de expansión mínima y de selección de características. Se ha llevado a cabo el desarrollo de un algoritmo heurístico de tipo genético con el cual ha sido posible encontrar soluciones razonablemente buenas y de forma rápida con las condiciones dadas. En la primera parte del trabajo se explicarán todos los conceptos necesarios para una correcta comprensión

de este. Después, se explicará cómo ha sido diseñado el algoritmo de tipo genético a la vez que se exponen ejemplos para facilitar la comprensión y, por último, veremos los resultados computacionales y qué quieren decir estos resultados.

2. Clustering jerárquico

El clustering jerárquico se refiere a un agrupamiento de datos que busca organizar elementos similares en grupos o clústers de manera jerárquica, formando así una estructura de árbol o dendrograma. Este método comienza agrupando cada observación como su propio clúster y luego continúa enlazando el resto hasta que se haya creado un único clúster con todos los datos. Esta representación jerárquica nos permite visualizar la estructura de similitud entre los elementos y facilita la exploración y análisis de los datos en diferentes niveles de granularidad. Existen numerosos métodos a la hora de agrupar datos, pero nosotros nos centraremos en el método *Single linkage.* (Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, 2021) (Jain, A. K., Murty, M. N., & Flynn, P. J. , 1999)

2.1. Single linkage

El single linkage (vinculación simple) es un método de clustering jerárquico utilizado para agrupar datos en dendrogramas. En este método, la distancia entre dos clústers se define como la distancia mínima entre cualquier par de puntos a cada uno de los clústers. (Martine Labbé, Mercedes Landete, Marina Leal, 2022) Funciona de la siguiente manera:

1. Comenzamos con cada elemento individual como su propio clúster.
2. Calculamos la distancia entre todos los pares de elementos y se registra la distancia más cercana entre cualquier par de elementos.
3. Fusionamos los dos grupos cuyos elementos tienen la distancia más cercana, formando un nuevo clúster que contiene todos los elementos de ambos grupos.

4. Actualizamos la matriz de distancias para reflejar la distancia entre el nuevo grupo y los grupos restantes.
5. Repetimos los pasos 2 y 4 hasta que todos los elementos estén agrupados en un solo clúster, formando así el dendrograma.

2.2. ¿Qué es un dendrograma?

El resultado de un clustering jerárquico es un árbol que representa las conexiones entre objetos en diferentes niveles. La representación gráfica de este árbol se llama dendrograma. En un dendrograma, los individuos están conectados en función de su similitud; cuanto más parecidos sean dos individuos, más cercana será la conexión de estos. Una vez obtenido el dendrograma, los grupos se determinan mediante cortes en el árbol: los individuos por debajo del nivel del corte permanecen en el mismo grupo. (Martine Labbé, Mercedes Landete, Marina Leal, 2022)

El dendrograma proporciona una visión clara de la estructura jerárquica de los datos. Los clusters más cercanos entre sí se fusionan antes, mientras que los clusters más distantes se fusionan en etapas posteriores. Por lo tanto, el dendrograma revela la similitud relativa entre grupos o clústeres en función de la distancia o similitud utilizada. Para determinar el número de clústeres en la solución final, se debe realizar un corte en el dendrograma en un nivel específico. Por lo general, al cortar el dendrograma en un nivel más alto, se obtienen menos clústeres pero más grandes, mientras que un corte en un nivel más bajo genera más clústeres pero más pequeños.

Existen diferentes métodos de enlace (también conocidos como criterios de vinculación o linkage methods) que se utilizan para construir dendrogramas en el contexto de algoritmos de clustering jerárquico. Estos métodos determinan cómo se calcula la similitud o distancia entre grupos de objetos y son fundamentales para la forma en que se fusionan los clusters. El método que se utiliza en el trabajo es, como ya se ha dicho, el single linkage.

3. Selección de características en árboles de expansión

La selección de características en árboles de expansión se refiere al proceso de elegir un conjunto óptimo de características de un conjunto de datos para construir un árbol de expansión. El objetivo principal de esta técnica es identificar las características más relevantes que contribuyen significativamente a la solución del problema en cuestión. Para poder entender mejor este concepto es necesario desarrollar las ideas básicas de este por separado.

3.1. ¿Qué es un árbol de expansión?

Un árbol de expansión es una estructura de grafo que se utiliza para conectar todos los nodos de un grafo conexo utilizando la menor cantidad de aristas posibles. Es decir, un árbol de expansión es un subconjunto de aristas que forman un árbol y conectan todos los nodos del grafo original sin formar ciclos. Existen principalmente dos tipos de árboles de expansión: árboles de expansión mínima (MST) por sus siglas en inglés, Minimum Spanning Tree, y árboles de expansión máxima. En este trabajo nos centraremos en el primer tipo.

Un árbol de expansión mínima es un tipo de árbol que se utiliza en grafos ponderados y conexos para conectar todos los nodos (vértices) del grafo con el menor peso total posible sin formar ciclos. En otras palabras, el MST es un subconjunto del conjunto de aristas del grafo original que cumple dos condiciones:

1. Contiene todos los nodos del grafo de modo que todos están conectados.
2. Tiene el menor peso total posible, es decir, la suma de los pesos de las aristas seleccionadas es mínima.

Para poder entender de manera más clara este concepto, se hará uso de un sencillo ejemplo:

Supongamos que debemos encontrar el árbol de expansión mínima del siguiente grafo de 4 nodos donde conocemos los costes entre vértices.

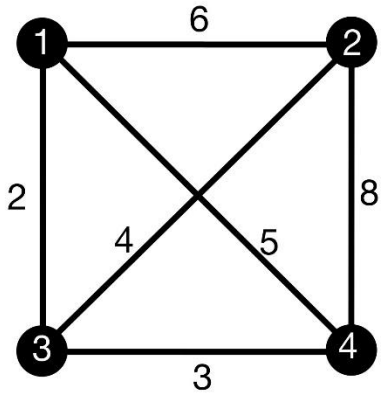

Figura 2.1.1 Grafo con costes entre vértices

Para calcular el árbol de expansión se conocen diferentes procedimientos. A continuación, explicamos uno de ellos sobre el grafo de la Figura 2.1.1. Comenzamos uniendo los vértices 1 y 3 pues son los que menor coste tienen. El siguiente vértice con menor coste que podemos unir al árbol es el vértice 4 que se une con el 3. Por último, para terminar de conectar todos los vértices, unimos el vértice 2 con el vértice 3. Con esto se consigue el siguiente árbol de expansión mínima:

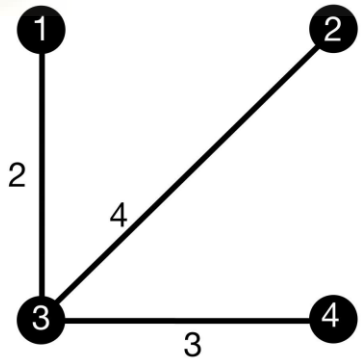

Figura 2.1.2 Representación del árbol de expansión mínima

El coste del árbol es de 9 (2 + 4 + 3).

Es de vital importancia destacar en este trabajo que toda la información necesaria para construir un dendograma con el single linkage se encuentra contenida en el MST. (Martine Labbé, Mercedes Landete, Marina Leal, 2022)

3.2. ¿Qué es la selección de características?

Se conoce la selección de características como el proceso de elegir un subconjunto relevante de características o variables de un conjunto de datos más grande con el objetivo de identificar aquellas características que contribuyen al problema de una manera más significativa.

Para nuestro problema buscaremos la combinación de características que mejor resultados nos dé, esto es la combinación que da lugar a un árbol de expansión mínima de menor coste. La combinación de dos características se da al sumar el coste de la misma unión de dos vértices entre las dos características. Para entenderlo mejor completaremos el ejemplo anterior:

Supongamos que debemos encontrar el árbol de expansión mínima de los siguientes grafos seleccionando dos características.

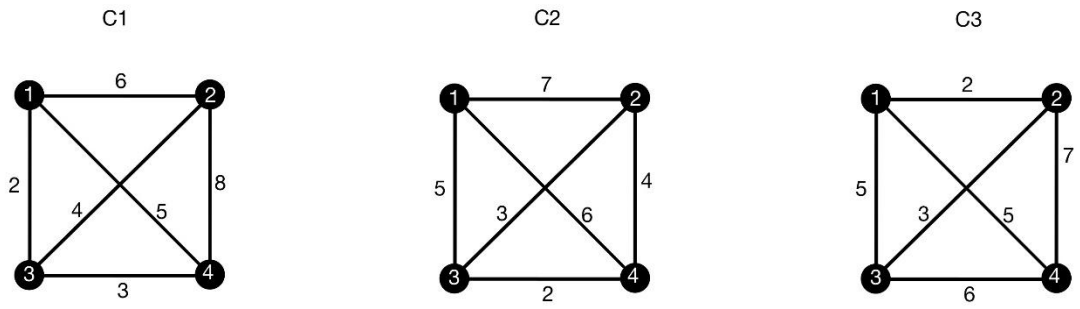

Figura 2.2.1 Distintos costes para el mismo grafo

En la Figura 2.2.1 se aprecian 3 costes distintos, cada uno sujeto a una característica. Como queremos encontrar los MST combinando dos características tenemos que combinar los tres grafos dos a dos lo que nos lleva a 3 posibles combinaciones (C1-C2, C1-C3, C2-C3).

Estas combinaciones se consiguen, como decíamos anteriormente, sumando todas las aristas de los dos grafos que se combinan.

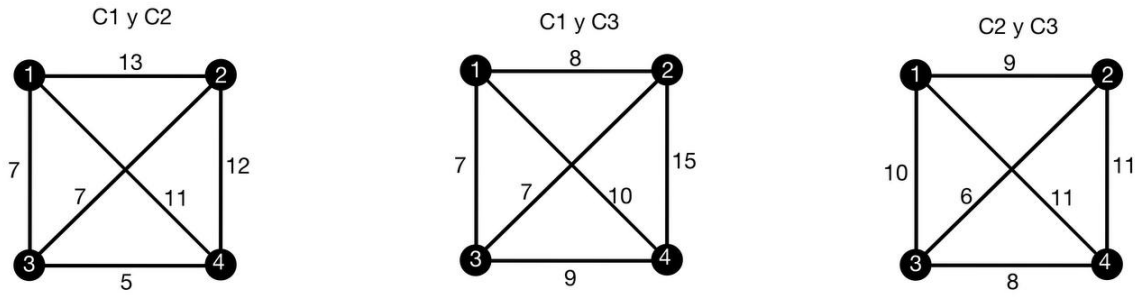

Figura 2.2.2 Combinaciones posibles con los costes por arista

El siguiente paso sería encontrar el árbol de expansión mínima de cada combinación con su coste asociado para ver cuál es la combinación óptima que, para nuestro problema, es la de menor coste.

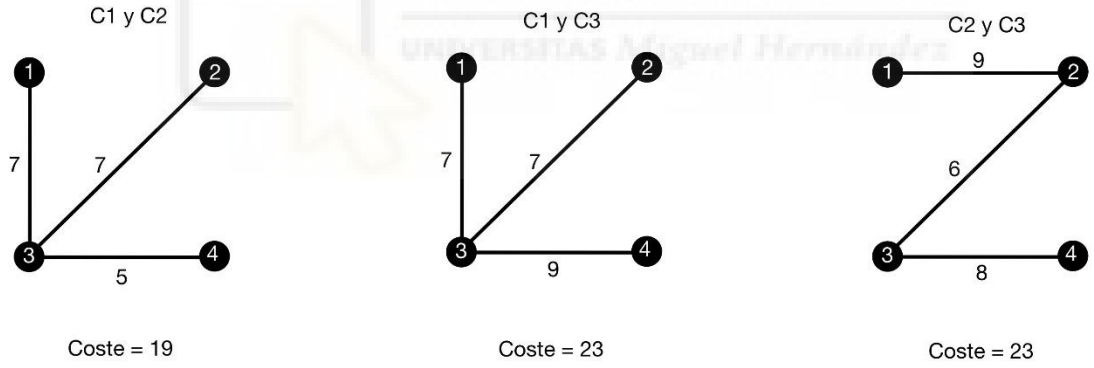

Figura 2.2.3 MST y costes para las 3 combinaciones de nuestro ejemplo

Como se observa, para este ejemplo y en el caso de que queramos combinar dos características, el menor coste de los árboles de expansión es de 19 unidades. Esto quiere decir que se seleccionan las características 1 y 2.

El ejemplo que se acaba de explicar ilustra de forma sencilla lo que se desea resolver en general: la elección de características que dan el MST de mínimo coste.

4. Resolución exacta del problema del cálculo de dendrogramas con selección de características

Cuando se trata de resolver problemas de optimización, encontrar la solución exacta del problema puede resultar muy costoso a nivel computacional y en término de tiempos, es por ello que a veces nos tenemos que conformar con encontrar una buena solución aproximada. Ahora, nos centraremos en la resolución exacta del problema para saber si la solución aproximada realmente lo es debemos conocer la exacta. Para la obtención de estas soluciones se aplican algoritmos rigurosos y deterministas. Estos algoritmos garantizan hallar la mejor solución posible dentro del espacio de búsqueda definido por el problema.

La resolución de nuestro problema de manera exacta ha consistido en programar en Python un algoritmo que calculase los costes de los árboles de expansión mínima para todas las combinaciones. Los árboles de expansión mínima y su coste asociado se han calculado con el algoritmo de Kruskal, el cual explicaremos en la Sección 4.1. Una vez calculados todos los costes, le hemos pedido al programa que nos muestre cuál es el menor de los valores (menor coste), con qué combinación de características se ha obtenido dicho valor y, también, cuánto ha tardado en conseguirlo. Se puede apreciar un ejemplo de los resultados de una instancia con el algoritmo exacto en la Figura 6.1.1 de este mismo trabajo.

4.1. Algoritmo de Kruskal

El algoritmo de Kruskal es un algoritmo de la teoría de grafos utilizado para encontrar el árbol de expansión mínima de un grafo no dirigido y conexo. Este algoritmo es de tipo voraz y construye el árbol paso a paso, seleccionando en cada paso la arista de menor peso que no forme un ciclo con las aristas previamente seleccionadas. (Wikipedia, 2020)

El algoritmo de Kruskal se puede resumir en los siguientes pasos:

1. Ordenar todas las aristas del grafo según sus pesos.

2. Inicializar un conjunto vacío para almacenar las aristas seleccionadas en el árbol de expansión mínima.
3. Recorrer las aristas en orden ascendente y, para cada arista, comprobar si su inclusión en el conjunto de aristas seleccionadas forma un ciclo.
4. Continuar hasta que se hayan añadido suficientes aristas al conjunto del MST para formar un árbol que conecte todos los vértices del grafo.

Al finalizar, el conjunto de aristas seleccionadas formará el árbol de expansión mínima.

4.2. Pseudocódigo del algoritmo exacto

Leemos los datos y creamos la población a estudiar (todas las combinaciones posibles)

Datos de entrada: Matriz de costes (M), número de características (K), número de nodos (N), número de características seleccionadas (p), número de combinaciones elegidas como población inicial (V)

01 Crear *'lista'* = {1, 2, …, k}

02 Crear conjunto '*combinaciones*' con todas las combinaciones posibles de p elementos de la lista 'lista'

03 Crear una lista vacía llamada *'objetivos'* para almacenar los costes totales

04 Para i = 1, …, N:

05 Crear una lista vacía llamada 'r'

06 Para k = 0, …, p:

07 Añadir el elemento combinaciones[i][k] a la lista *'r'*

Calculamos los costes del del árbol de expansión mínima para toda la población creada

08 Crear un diccionario vacío llamado *'ctree'* para almacenar los costes del árbol

09 Para i = 1, …, N:

10 Para j = 1, …, N:

11 Si i < j:

12 ctree[i,j] = 0

13 Para k = 1, …, P:

14 ctree[i,j] = ctree[i,j] + c[i,j,r[k]]

Calculamos el MST del grafo 'g' y obtenemos el valor objetivo

15 Crear un objeto grafo *'g'* con N nodos

16 Para i = 1, …, N:

17 Para j = 1, …, N:

18 Si i < j:

19 Añadir una arista entre los nodos i-1 y j-1 al grafo *'g'* con peso ctree[i, j]

20 Calcular el MST (Minimum Spanning Tree) del grafo *'g'* y obtener el valor objetivo MSTobj y la lista de aristas MSTx

21 Añadir el valor MSTobj a la lista *'objetivos'*

22 Ordenar la lista 'objetivos' en orden ascendente y guardarla en *'objetivos_ordenados'*

23 Obtener el mejor valor objetivo *'mejor_valor'* como el primer elemento de *'objetivos_ordenados'*

5. Algoritmo heurístico

Un algoritmo heurístico es un método para resolver un problema que no garantiza una solución óptima, pero que puede encontrar una solución rápida y aproximada. Los algoritmos heurísticos se utilizan a menudo para resolver problemas que son demasiado complejos o que requieren demasiado tiempo para resolverlos con métodos exactos (Emorís, 2013).

Existen métodos heurísticos de diversa naturaleza por lo que su clasificación es bastante complicada. Se sugiere la siguiente clasificación (Sarrió, 2022):

- *Métodos de descomposición.* Hacen servir la misma estructura de los problemas para solucionar problemas de menor tamaño de manera secuencial.
- *Métodos inductivos.* Pretenden extrapolar un problema pequeño a uno de dimensión mayor.

- *Métodos de reducción.* Tratan de determinar propiedades que cumplen las buenas soluciones para introducirlas como restricciones del problema.
- *Métodos constructivos.* Forman poco a poco una solución del problema y, en cada iteración se quedan con la mejor solución.
- Métodos de Búsqueda Local. En este caso, identifican una solución del problema y la van mejorando hasta que no encuentran un valor mejor.

Dentro de los algoritmos heurísticos situamos los metaheurísticos, que son aquellos que tratan de mejorar los primeros. Estos se clasifican en: *Métodos Constructivos, Métodos Evolutivos y Métodos de Búsqueda.*

En este estudio se ha creado un algoritmo genético, incluido dentro de los métodos evolutivos, para el cálculo de dendrogramas con selección de características en problemas de optimización.

5.1. Algoritmo heurístico de tipo genético

Un algoritmo genético es un tipo de algoritmo heurístico inspirado en la evolución natural de poblaciones, en la teoría de la selección natural de Darwin y en la teoría de la transferencia de material genético de Mendel. Un algoritmo genético emula el comportamiento de una población de individuos que representan soluciones y que evoluciona en base a los principios de la evolución natural: reproducción mediante operadores genéticos y selección de los mejores individuos, correspondiendo estos a las mejores soluciones del problema a optimizar. Estos algoritmos son frecuentemente utilizados para la resolución de problemas complejos de optimización. (Algoritmo de Búsqueda Heurística, s.f.)

La versión simple o canónica del algoritmo genético trabaja siguiendo los siguientes pasos:

1. Generar una población inicial de soluciones
2. Seleccionar aleatoriamente de la población actual algunas de las soluciones

3. Cruzar esas soluciones para obtener descendencia
4. Mutar algunas soluciones para obtener soluciones mutadas
5. Elegir las soluciones que sobreviven y formarán la nueva generación
6. Realizar una búsqueda local para encontrar el mejor de los genes de uno de los que tenemos en la población
7. Volver al paso 2 hasta que se alcance el criterio de parada o el número de iteraciones establecidas

5.1.1. Población

En un algoritmo heurístico de tipo genético, la población inicial es el primer conjunto de soluciones candidatas con las que comienza el proceso de optimización. La población es una colección de individuos, y cada individuo representa una posible solución al problema que estamos intentando resolver. Los genes representan las partes individuales de una solución candidata o las características que forman parte de la solución al problema que se está abordando. El gen puede tener toda la información que el programador considere oportuna. En nuestro caso, el gen tendrá longitud p, es decir, se compondrá de tantas características como se seleccionan para la resolución del problema.

Supongamos que vamos a resolver una instancia con 100 nodos, 15 características y 3 características seleccionadas. A la hora de obtener una población inicial, crearemos el número de genes que consideremos de longitud 3. Por ejemplo, para la resolución de dicha instancia, la población se inició con 4 genes (o vectores) aleatorios de longitud 3 como estos: (2, 3, 11), (6, 14, 15), (4, 11, 13), (10, 13, 15). En cada uno de los genes todas las características deben ser distintas pues, si se repitiese alguna, no estaríamos combinando 3 de ellas sino menos y no es lo que queremos.

Cada uno de los vectores anteriores son una combinación de características del problema con un coste de los árboles de expansión mínima. Los costes asociados a los árboles de expansión de dichas combinaciones son:

Combinación	Coste
(2, 3, 11)	5799
(6, 14, 15)	6934
(4, 11, 13)	6204
(10, 13, 15)	6954

Figura 5.1.1.1 Costes de los vectores de población inicial

Estos costes están bastante lejos de ser una buena solución al problema. Ahora que ya tenemos una población inicial por la que empezar, podemos empezar a cruzar los genes.

5.1.2. Cruce

Para realizar el cruce se han escogido dos genes al azar (genes padre) y se han unido. Del cruce de los genes padre surge un nuevo gen, el gen hijo. Este gen hijo, que es una combinación de características aleatorias escogidas de los genes padres, tiene también asociado un coste del árbol de expansión mínima calculado para sus características. Si por ejemplo se cruzasen los vectores (4, 11, 13) y (10, 13, 15), un posible hijo sería el (13, 10, 15) el cual su árbol de expansión mínima tiene un coste asociado de 6954.

Una vez conocido el valor del coste asociado al hijo pueden suceder dos cosas:

1. El valor del coste es menor que alguno de los genes de la población y entonces se sustituye el peor de los genes de la población por el gen hijo.
2. El valor del coste no es menor que ninguno de los genes de la población y entonces no se sustituye, se sigue iterando.

En el ejemplo que se exponía, el valor del coste asociado al gen hijo es igual que el mayor de los ya existentes en la población (6954), por lo que, al no ser estrictamente menor, este vector no entra en la población.

5.1.3. Mutación

La mutación en un algoritmo genético es una operación fundamental que permite introducir variabilidad en la población y explorar nuevas soluciones en el espacio de búsqueda. Su principal propósito es mantener la diversidad genética y evitar que el algoritmo quede atrapado en óptimos locales, lo que podría limitar su capacidad para encontrar soluciones óptimas o cercanas a esta.

Por lo general, la mutación se trata de escoger un gen al azar y cambiar un elemento (característica en nuestro caso), también al azar, por otro y calcular el coste asociado al árbol de expansión mínima con esa combinación. En este caso, no importa si el resultado es mejor o peor, pues el gen mutado se incorporará de todos modos a la población reemplazando al peor de esta. Lo que si que hay que tener en cuenta es que la característica del gen que cambiemos no debe repetirse, es decir, no puede estar ya en el gen.

Siguiendo con el ejemplo anterior, si por ejemplo tuviésemos que mutar el gen (4, 11, 13) comenzaríamos por elegir una de sus características aleatoriamente. Si, por ejemplo, se elige aleatoriamente la característica 13, esta se cambiaría por otra de todas las características posibles. Esto es todas las características exceptuando las que ya posee el vector y la misma característica. Una vez obtenido el vector mutado, por ejemplo podría ser el (4, 11, 12), se calcula el coste asociado al árbol de expansión mínima con estas características y se sustituye sí o sí por el vector elegido para ser mutado. En nuestro ejemplo se ha realizado la mutación en el 10% de las iteraciones.

5.1.4 Búsqueda Local

La búsqueda local es una estrategia que se utiliza para mejorar las soluciones del algoritmo una vez que este ya ha realizado cruces y mutaciones. Esta estrategia busca explorar de manera más detallada el entorno de una solución para encontrar soluciones óptimas o más cercanas a la óptima.

En este caso se trata de elegir un gen al azar y un elemento de dicho gen también al azar y comprobar cuál es la mejor de las combinaciones posibles entre los elementos restantes y el elemento a cambiar. Para entender mejor en qué consiste imaginemos que tenemos el gen (6, 7, 14). Ahora debemos elegir una característica aleatoria de este vector; supongamos que es la característica 6. El vector que quedaría entonces sería el (7, 14) y es al que le iremos añadiendo una a una todas las características posibles para comprobar cuál es el mejor de los valores de la siguiente forma:

Combinación	Valor
(7, 14, 1)	4437
(7, 14, 2)	3595
(7, 14, 3)	4894
(7, 14, 4)	3773
(7, 14, 5)	5049
(7, 14, 6)	5842
(7, 14, 8)	5373
(7, 14, 9)	5823
(7, 14, 10)	4227
(7, 14, 11)	4450
(7, 14, 12)	5448
(7, 14, 13)	4642
(7, 14, 15)	4917

Figura 5.1.4.1 Valores obtenidos con Búsqueda Local para un vector

Una vez realizada la búsqueda local nos damos cuenta de que la mejor combinación de características es (2, 7, 14), pues es la que menor coste asociado tiene su árbol de expansión mínima.

Una vez encontrada la mejor combinación de características, esta se sustituye en la población por la combinación elegida al azar.

5.2. Pseudocódigo del algoritmo heurístico

Antes de comenzar con el desarrollo del pseudocódigo del algoritmo heurístico, cabe destacar que este es considerablemente más extenso que el del algoritmo exacto. A nivel computacional es más sencillo el heurístico pues, no es necesario que haga todos los cálculos que sí necesita el exacto para la resolución exacta del problema, sino que iterará tantas veces como queramos. Para que la solución que nos ofrece el algoritmo heurístico sea razonablemente buena, es necesario una programación más compleja de dicho algoritmo lo que nos lleva a que este sea, como comentábamos, más extenso.

Leemos los datos y creamos una población inicial

Datos de entrada: Matriz de costes (M), número de características (K), número de nodos (N), número de características seleccionadas (p), número de combinaciones elegidas como población inicial (V)

01 Crear *'lista'* = {1, 2, …, K}

02 Generar todas las combinaciones posibles de p elementos de *'lista'* y guardarlas en *'combinaciones'*

03 Seleccionar aleatoriamente V combinaciones y guardarlas en *'vectores_elegidos'*

04 Crear una lista vacía llamada *'objetivos'* para almacenar los objetivos

05 Crear una lista vacía llamada *'vectores_objetivos_pequenos'* para almacenar los vectores correspondientes a los objetivos

06 Para i = 1, …, V:

07 Crear una lista vacía llamada 'r'

08 Para K = 1, …, p:

09 Añadir el elemento vectores_elegidos[i][k] a la lista *'r'*

Calculamos los costes del del árbol de expansión mínima para la población inicial

10 Crear un diccionario vacío llamado *'ctree'* para almacenar los costes del árbol

11 Para i = 1, …, N:

12 Para j = 1, …, N:

13 Si i < j:

14 ctree[i,j] = 0

15 Para k = 1, …, P:

16 ctree[i,j] = ctree[i,j] + c[i,j,r[k]]

Creamos el grafo 'g' con N nodos correspondiente al árbol de expansión mínima

17 Crear un objeto grafo *'g'* con N nodos

18 Para i = 1, …, N:

19 Para j = 1, …, N:

20 Si i < j:

21 Añadir una arista entre los nodos i-1 y j-1 al grafo *'g'* con peso ctree[i, j]

22 Calcular el MST (Minimum Spanning Tree) del grafo *'g'* y obtener el valor objetivo MSTobj y la lista de aristas MSTx

23 Añadir los valores de MSTobj a *'objetivos'*

24 Añadir los vectores obtenidos *'r'* a *'vectores_objetivos_pequenos'*

25 Para i = 1, …, número iteraciones:

Creamos la mutación

26 Si i ≠ 0 y el valor elegido para la mutación es múltiplo de las iteraciones:

27 Seleccionar aleatoriamente 1 elemento de *'vectores_objetivos_pequenos'* y guardarlo en *'vector_aleatorio'*

28 Eliminar una de las componentes de *'vector_aleatorio'* y guardarlo como *'vector mutado'*

29 Creamos la lista 'características' = {1, 2, …, K + 1}

30 Para i en vector_mutado:

31 Eliminar característica i

32 Añadir la *'característica_anadida'* a *'vector_mutado'*

Volvemos a calcular los costes del árbol de expansión mínima

33 Crear un diccionario vacío llamado *'ctree'* para almacenar los costes del árbol

…

Volvemos a crear calcular el MST del grafo ‘g’ y su valor objetivo

34 Crear un objeto grafo *'g'* con N nodos

…

35 Eliminar de *'objetivos'* el objetivo que le corresponde al vector elegido aleatoriamente para la mutación

36 Añadir el objetivo calculado *'MSTobj'*

37 Ordenamos los objetivos y los guardamos en *'objetivos*_ordenados’

38 Eliminar de *'vectores_objetivos_pequenos'* el vector que se ha elegido aleatoriamente

39 Añadir a *'vectores_objetivos_pequenos'* el vector *'vector_mutado'*

40 Seleccionar el más pequeño de los objetivos ordenados y guardarlo en *'index_peor'*

Creamos la búsqueda local

41 O si i ≠ 0 y el valor elegido para la Búsqueda Local es múltiplo de las iteraciones:

42 Seleccionar aleatoriamente 1 elemento de *'vectores_objetivos_pequenos'* y guardarlo en *'vector_aleatorio'*

43 Seleccionar aleatoriamente 1 elemento de *'vector_aleatorio'* y guardarlo en *'caracteristica_aleatoria'*

44 Copiar *'vector_aleatorio'* en *'vector_mutado'*

45 Eliminar la característica aleatoria de *‘vector_mutado’*

46 Crear una lista vacía 'objetivos_mutados'

47 Crear una lista vacía ‘vectores_mutados'

48 Crear una lista 'caracteristicas_disponibles' = {1, 2, …, K + 1}

49 Para *'caracteristica_anadida'* en el rango de *'caracteristicas_disponibles'*:

50 Copiar *'vector_aleatorio'* en *'vector_mutado'*

51 Eliminar *'característica_aleatoria'* de *'vector_mutado'*

52 Si 'caracteristica_anadida' no está en 'vector_mutado'

53 Añadir una característica a 'vector_mutado'

54 Añadir el vector mutado a 'vectores_mutados'

Volvemos a calcular los costes del árbol de expansión mínima

55 Crear un diccionario vacío llamado *'ctree'* para almacenar los costes del árbol

…

Volvemos a crear calcular el MST del grafo 'g' y su valor objetivo

56 Crear un objeto grafo *'g'* con N nodos

…

57 Añadir el objetivo mutado a la lista de objetivos *'objetivos_mutados'*

Seleccionar el más pequeño de los objetivos ordenados y guardarlo en *'index_peor'*

58 Eliminar de 'objetivos' el objetivo que le corresponde al vector elegido aleatoriamente la búsqueda local

Añadir el objetivo calculado *'MSTObj'*

59 Ordenar los objetivos y guardarlos en *'objetivos_ordenados'*

Eliminar de *'vectores_objetivos_pequenos'* el vector que se ha elegido aleatoriamente

60 Añadir a *'vectores_objetivos_pequenos'* el vector *'mejor_vector_mutado'*

En las iteraciones en las que no hay búsqueda local o mutación se realiza el cruce

61 Si no:

62 Crear una lista vacía llamada *'índices'* para almacenar la posición de los vectores padre para el cruce

62 Mientras la longitud de la lista *'índices'* < 2:

63 Elegir un número aleatorio desde 0 hasta V y guardarlo como *'índice'*

64 Si *'índice'* no está en *'índices'* entonces:

65 Añadir *'índice'* a *'índices'*

66 Crear una lista vacía llamada 'objetivos_elegidos' para almacenar los valores de los vectores elegidos para el cruce

67 Para i = 0, 1:

68 Añadir a *'vectores_objetivos'* el vector de *'vectores_objetivos_pequenos'* de índice i

69 Crear una lista vacía llamada *'vector_hijo'*

70 Añadir a *'vector_hijo'* un vector aleatorio del primero de los *'vectores_objetivos'*

71 Mientras la longitud de *'vector_hijo'* < p:

72 Elegir un vector aleatorio de *'vectores_objetivos'* y guardarlo en *'car'*

73 Si *'car'* no se encuentra entre los elementos de *'vector_hijo'*:

74 Añadir *'car'* a *'vector_hijo'*

75 Si *'vector_hijo'* no se encuentra entre *'vectores_objetivos_pequenos'*:

76 Para i = 0, …, p:

Crear una lista vacía llamada 'r'

77 Para k = 0, …, p:

78 Añadir el elemento vector_hijo[k] a la lista 'r'

Volvemos a calcular los costes del árbol de expansión mínima

79 Crear un diccionario vacío llamado *'ctree'* para almacenar los costes del árbol

…

Volvemos a crear calcular el MST del grafo 'g' y su valor objetivo

80 Crear un objeto grafo *'g'* con N nodos

…

81 Guardar como *'index_peor'* mayor de los valores de *'objetivos_ordenados'*

82 Si *'MSTobj'* < que el mayor de los valores de *'objetivos_ordenados'*:

83 Sustituir el peor objetivo por el del vector hijo

84 Sustituir el vector correspondiente al peor objetivo por el del vector hijo

85 Ordenar los objetivos

6. Resultados computacionales

En esta sección presentamos los resultados obtenidos mediante la implementación del algoritmo genético y su comparación con el algoritmo exacto. Para una mayor comprensión se utilizará una de las instancias a modo de ejemplo explicativo.

6.1. Ejemplo explicativo resolviendo una única instancia

La primera instancia por resolver posee 100 nodos, 15 características y se seleccionan 3 de las características. Previamente a la resolución de la instancia con el algoritmo heurístico se ha resuelto con el algoritmo exacto, lo que nos ha dejado los siguientes datos:

VALOR	CARACTERÍSTICAS	TIEMPO
3595	2 7 14	16.95

Figura 6.1.1 Resultados con algoritmo exacto

Esto quiere decir que, el mejor valor de todos los árboles de expansión mínima es 3595, conseguido con la combinación de características 2, 7 y 14. La resolución de esta instancia le ha llevado al algoritmo exacto 16.95 segundos.

Ahora lanzaremos 10 veces el algoritmo heurístico para ver cómo de bien funciona. Como población inicial utilizaremos 4 vectores (combinaciones de características) y el número de iteraciones será de 100. El porcentaje de mutación se fija en el 10% y el de búsqueda local en el 29% de las iteraciones.

INTENTO	MEJOR VALOR	CARACTERÍSTICAS	TIEMPO
1	3595	2 7 14	6.99
2	3595	2 7 14	6.37
3	4018	2 11 14	6.52
4	3595	2 7 14	6.22
5	3595	2 7 14	6.08
6	3745	2 4 14	6.25

7	3745	2 5 14	6.76
8	3595	2 7 14	6.55
9	3595	2 7 14	6.15
10	3595	2 7 14	6.26
Media	3667.3	-	6.41

Figura 6.1.2 Tabla de resultados instancia de ejemplo

Como se puede observar, el algoritmo heurístico ha encontrado la solución óptima 7 de las 10 veces que lo hemos lanzado. Las 3 veces que no ha encontrado la solución óptima el valor ha sido próximo.

Es interesante comentar cómo ha mejorado el algoritmo al haber añadido la búsqueda local. Cuando se realizó el mismo estudio con la misma instancia sin haber incluido la búsqueda local los resultaros fueron los siguientes:

INTENTO	MEJOR VALOR	CARACTERÍSTICAS	TIEMPO
1	3773	7 14 4	5.230146408081055
2	4018	14 11 2	5.080327272415161
3	3595	2 14 7	6.008177757263184
4	4582	14 4 13	6.65901517868042
5	4592	14 1 2	4.696047782897949
6	4018	14 11 2	6.421388864517212
7	3595	14 2 7	6.329698324203491
8	3745	4 2 14	5.277873992919922
9	3595	7 2 14	5.58092188835144
10	3595	14 7 2	6.086152076721191
Media	3910.8	-	5.943

Figura 6.1.3 Tabla de resultados instancia de ejemplo sin búsqueda local

Aunque ahora el algoritmo es algo más lento, la media de los valores obtenidos tras 10 intentos es significantemente menor por lo que parece que la búsqueda local es efectiva.

Por último, cabe destacar que el tiempo que tarda el algoritmo genético en dar una solución aproximada es notablemente menor que el tiempo que tarda el algoritmo exacto, concretamente en un 62.2%.

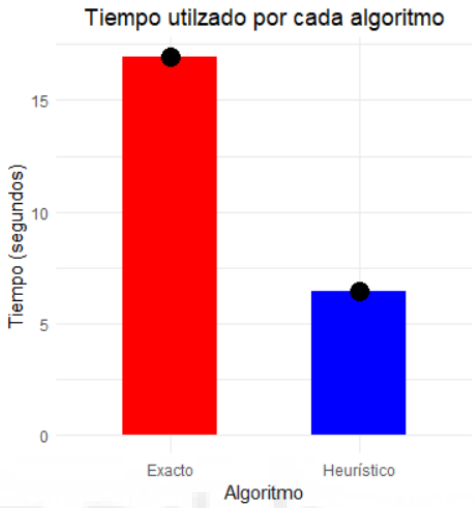

Figura 6.1.4 Tiempos utilizados por cada algoritmo

Es interesante también mostrar gráficamente uno de los árboles obtenidos con las ejecuciones del ejemplo:

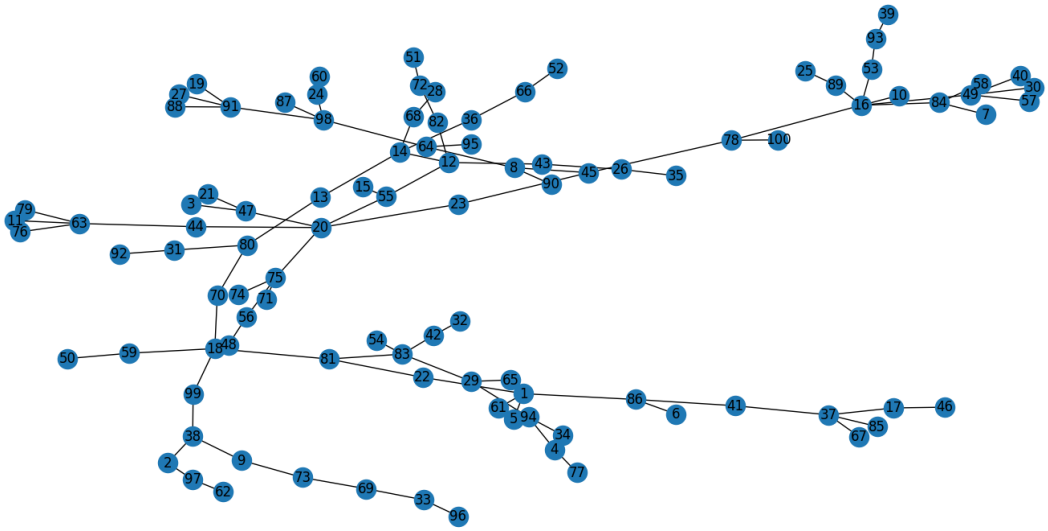

Figura 6.1.5 Representación gráfica del MST de una de las ejecuciones

6.2. Resolución de múltiples instancias

Tras resolver el problema tanto de manera exacta como utilizando el algoritmo heurístico, reuniremos los resultados en una tabla para su posterior análisis e interpretación.

Esta tabla contendrá los resultados de 15 instancias en total de 3 tamaños distintos: 5 instancias con 50 nodos (N), 5 instancias con 40 nodos y 5 instancias con 20 nodos. Todas las instancias con N = 50 tienen 9 características y se seleccionan 4 de ellas, todas las instancias con N = 40 tienen 10 características y se seleccionan 5 de ellas y todas las instancias con N = 20 tienen 12 características y se seleccionan 5 de ellas.

Según el número de nodos que posea la instancia se realizará un número de iteraciones concreto. Para las instancias con 50 nodos se ha decido realizar 100 iteraciones, para las instancias con 40 nodos se realizarán 80 iteraciones y, por último, para las instancias con 20 nodos se realizarán 60 iteraciones. Para todas ellas se realizará una mutación en el 5% de las iteraciones y una búsqueda local en el 29% de las ocasiones.

En la Figura *6.2.1*, vemos que la primera columna posee la información de los tres parámetros que definen las instancias: N es el número de nodos, K es el número de características y P es el número de características que seleccionamos para la resolución de ese problema. Luego, tanto para la segunda como tercera columna, se tienen los 3 datos más relevantes para la interpretación del problema. El valor óptimo en el caso del método exacto y el que se obtiene con el método heurístico, el tiempo que emplea cada uno de los métodos y las características con las que se obtiene el valor de la solución óptima en el caso del método exacto y de la solución en el caso del heurístico.

DATOS			**Método exacto**			**Método heurístico**			**-**
N	**K**	**P**	**Valor**	**Tiempo**	**Características**	**Valor**	**Tiempo**	**Características**	**GAP**
50	9	4	325.13	1.34	4 7 8 9	325.13	1.87	4 7 8 9	0
50	9	4	327.20	1.34	4 6 8 9	327.20	1.80	4 6 8 9	0
50	9	4	314.12	1.25	4 6 7 8	314.12	1.78	4 6 7 8	0
50	9	4	323.68	1.26	4 6 7 9	324.12	1.74	4 7 8 9	0.13
50	9	4	330.73	1.32	4 6 8 9	330.73	1.75	4 6 8 9	0
40	10	5	377.18	1.69	1 7 8 9 10	377.18	1.59	1 7 8 9 10	0
40	10	5	380.18	1.71	1 6 8 9 10	380.18	1.85	1 6 8 9 10	0

40	10	5	377.92	1.81	1 7 8 9 10	378.61	1.77	1 5 8 9 10	0,18
40	10	5	371.70	1.73	1 6 8 9 19	371.70	1.67	1 6 8 9 10	0
40	10	5	387.34	1.70	1 5 8 9 10	396.05	1.43	1 5 7 8 10	2,19
20	12	5	183.42	1.47	3 6 8 9 11	183.42	0.23	3 6 8 9 11	0
20	12	5	184.71	1.44	3 6 8 10 11	184.71	0.24	3 6 8 10 11	0
20	12	5	172.94	1.43	6 8 9 10 11	172.94	0.24	6 8 9 10 11	0
20	12	5	180.90	1.44	3 6 7 10 11	183.24	0.23	6 7 9 10 11	1,27
20	12	5	186.48	1.48	3 8 9 10 11	186.48	0.24	3 8 9 10 11	0

Figura 6.2.1 Tabla de resultados de 15 instancias

Al observar los resultados nos damos cuenta de varias cosas. Para comenzar, el algoritmo heurístico parece funcionar bien pues son pocas las veces que se obtiene un GAP distinto de 0 y cuando se obtiene es pequeño. Cuando hablamos de GAP nos referimos al % de desviación entre los valores obtenidos con ambos métodos. La forma de calcular el GAP ha sido la siguiente:

$$\frac{valor\ heurístico - valor\ exacto}{valor\ heurístico}\ x\ 100$$

Cuando el GAP es igual a 0 quiere decir que hemos encontrado la solución óptima, y cuando estos son muy pequeños pero distintos de cero, nos encontramos ante una solución muy próxima a la óptima. Por tanto, como decíamos, parece que el algoritmo cumple bien su función de encontrar una buena solución.

Con las instancias estudiadas es algo difícil observar que realmente el algoritmo heurístico está cumpliendo su función en cuanto a tiempo y esto es porque las instancias no son de gran tamaño. En las instancias con N = 50 y N = 40 el número de características son 9 y 10 respectivamente de las cuales se seleccionan 4 y 5 de esas características. Esto hace que el número de combinaciones de características posibles no sea demasiado elevado. Eso sumado a que el número de nodos no es demasiado elevado, hace que el algoritmo exacto sea rápido de calcular todos los valores. Si nos fijamos en los tiempos entre el algoritmo exacto y heurístico en estas instancias vemos que son bastante parejos.

Por otro lado, tenemos las instancias con N = 20. Cuando calculamos los costes de estas instancias, el algoritmo exacto necesita un tiempo parecido al que ya venía necesitando

con las instancias de distinta N pues, aunque N sea menor, tenemos más características que combinar. Por otro lado, cuando calculamos los costes con el algoritmo heurístico, vemos como los tiempos se reducen considerablemente. Parece ser que al algoritmo heurístico le afecta más tener más nodos que características que combinar.

Bajo estas evidencias y sobre todo a las expuestas en la sección *6.1.* de este mismo trabajo, podemos decir que el algoritmo heurístico de tipo genético creado sí es efectivo, pues llega al óptimo en la mayoría de las ocasiones y lo hace en un tiempo menor que el algoritmo exacto.

6.3. Conclusiones

Tras haber realizado todo el estudio, podemos extraer varias conclusiones interesantes sobre este trabajo. En primer lugar, hay que destacar que se ha conseguido desarrollar un algoritmo heurístico de tipo genético que cumple con las condiciones o restricciones del problema: dar soluciones razonablemente buenas en tiempos no tan costosos. Por ejemplo, si retrocedemos al ejemplo de la sección 6.1, vemos como la mejora de tiempo en ambos algoritmos era de un 62%, reduciendo el tiempo utilizado por el algoritmo exacto de 16 segundos, a un tiempo de tan solo 6 segundos con el algoritmo heurístico creado.

Algo que se ha de valorar también del algoritmo creado es que, a mayor tamaño de instancias a resolver, mayor eficacia tiene. Cuando hemos resuelto instancias no tan grandes como las de la sección 6.2, no hemos encontrado diferencias de tiempo tan grandes entre ambos algoritmos.

Como conclusión de lo anterior, podemos estar contentos de garantizar el correcto de un funcionamiento de un algoritmo que, en una línea futura de trabajo, resolverá de manera efectiva instancias de gran tamaño en la que las diferencias de tiempo serán realmente importantes para nosotros. Además, es interesante entender que el algoritmo puede ser modificado para mejorar añadiendo, por ejemplo, más procesos de búsqueda local, consiguiendo así potenciar el algoritmo.

7. Bibliografía

Algoritmo de Búsqueda Heurística. (s.f.). Obtenido de Capítulo 8: https://biblus.us.es/bibing/proyectos/abreproy/70238/fichero/Capitulo+8.pdf

Emorís, A. V. (2013). *Algoritmos heurísticos en optimización.* Trabajo fin de master en Técnicas estadísticas. Universidad de Santiago de Compostela.

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2021). *An Introduction to Statistical*. Obtenido de https://hastie.su.domains/ISLR2/ISLRv2_website.pdf.

Jain, A. K., Murty, M. N., & Flynn, P. J. . (1999). *Data clustering: A review. ACM Computing Surveys.*

Martine Labbé, Mercedes Landete, Marina Leal. (2022). Dendrograms, Minimum Spanning Trees and Feature.)Departamento de Estadística, Matemáticas e Informática, Universidad Miguel Hernández.

Sarrió, L. N. (2022). *Mejora de conexiones en problemas de localización.* Trabajo Fin de Grado. Universidad MIguel Hernández.

Wikipedia. (2020). Obtenido de Algoritmo de Kruskal: https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal