Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Clustering basado en la modificación de Grafos mediante Algoritmos Genéticos

Autor(a): Jose Ignacio Alba Rodríguez
Tutor(a): Alfonso Mateos Caballero y Eloy Vicente Cestero

Madrid, Julio de 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Clustering basado en la modificación de Grafos mediante Algoritmos Genéticos

Julio de 2024

Autor(a): Jose Ignacio Alba Rodríguez
Tutor(a): Alfonso Mateos Caballero y Eloy Vicente Cestero
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Debido a la gran importancia del aprendizaje no supervisado en el futuro de la ciencia, es necesario desarrollar de nuevos algoritmos que mejoren las técnicas previas. Aunque los algoritmos de clustering suelen clasificarse según su filosofía, las contribuciones más recientes suelen combinar elementos de diferentes categorías para obtener lo mejor de cada enfoque.

Siguiendo esa línea, en este trabajo se presenta el innovador algoritmo GPGAC. En este caso, un algoritmo genético de longitud variable es empleado para decidir la secuencia de modificaciones basadas en densidad que, tras ser aplicadas sobre el grafo de los k_0 vecinos más cercanos, obtendrá la mejor clasificación. Gracias a este nuevo enfoque, se obtiene un algoritmo que está a la altura del estado del arte, con algunas ventajas importantes como la independencia de los parámetros en el resultado final, la adaptabilidad al conjunto de datos y la flexibilidad que le otorga poder optimizar una medida de calidad deseada.

Abstract

Given the significant relevance of unsupervised learning for the future of sciences, newer algorithms are needed that improve the existing techniques. Although clustering algorithms are typically classified based on their philosophy, the most recent contributions tend to combine elements from different categories in order to obtain the best from each approach.

Following this trend, this thesis presents the novel algorithm GPGAC. In this case, a variable-length genetic algorithm is employed to decide the sequence of density-based modifications which, once applied over the graph k_0-nearest neighbors graph, results in the best clustering given a quality measure. Thanks to this new approach, a new algorithm is obtained. Not only is this algorithm able to compete with other state-of-the-art approaches, but it also includes several advantages, such as an independance of the solution quality from the initial parameters, the adaptability to a given dataset and the flexibility that arises from being able to choose which quality measure to optimize.

Agradecimientos

Me gustaría dedicar todo el trabajo dedicado durante la realización de este proyecto a las personas que me han querido, apoyado y cuidado a lo largo de esta etapa, ya que, sin ellas, esto no habría sido posible. Prometo devolveros toda esta dedicación hacia mí, que no ha tenido precio.

Por otro lado, me gustaría agradecer a mis tutores Alfonso y Eloy por la paciencia y la comprensión. Finalmente, quiero agradecer al Ministerio de Ciencia e Innovación la financiación del proyecto PID2021-122209OB-C31.

Tabla de contenidos

Capítulo 1

Introducción

La capacidad de diferenciar objetos a simple vista es una de las primeras habilidades que adquirimos los seres humanos durante nuestra infancia, ya que nos permite tanto interactuar con nuestro entorno como reconocer a nuestros seres queridos y poder socializar. Esta cualidad que adquirimos de forma inconsciente resulta vital para nuestro desarrollo cognitivo. Por tanto, no es de extrañar que en el mundo de la inteligencia artificial resulte de gran importancia replicar esta característica propia de los seres humanos para el desarrollo de sistemas complejos. La rama del Aprendizaje Automático busca crear modelos que reflejen estas cualidades de forma automática, sin necesidad de seguir unas reglas prefijadas. Esto se consigue mediante el desarrollo de algoritmos que son capaces de adaptarse al entorno o a un conjunto de datos para poder identificar patrones, diferenciar objetos o tomar decisiones de forma autónoma. Este tipo de modelos están en auge, ya que tienen infinitas aplicaciones en diferentes áreas, como el reconocimiento de patrones, el análisis de datos o el procesamiento de imágenes.

Dentro del Aprendizaje Automático, se pueden distinguir dos grandes ramas: el aprendizaje supervisado y el aprendizaje no supervisado. En el primer caso, se parte de un conjunto de datos de entrenamiento, donde previamente se conocen las clases reales de cada elemento. En este, el modelo es entrenado prediciendo la clase de estos datos para aprender iterativamente de sus errores, adaptándose poco a poco hasta que sea capaz de clasificar con éxito nuevas instancias de datos que no han sido nunca vistas durante el entrenamiento. En cambio, en el aprendizaje no supervisado se parte de un conjunto donde estas clases reales no son conocidas. Aquí, el objetivo pasa a ser el crear modelos capaces de extraer información a partir de los datos. Dentro de esta rama, se encuentra el *clustering*, el cual consiste en, dado un conjunto de datos, arbitrario, analizar las características similares de los datos para poder encontrar agrupaciones de estos. Estas agrupaciones, o clusters, deben verificar que todos los elementos dentro de un mismo cluster compartan muchas cualidades, mientras que son fácilmente diferenciables de los elementos del resto de clusters [23]. El aprendizaje no supervisado presenta una serie de dificultades añadidas frente al aprendizaje supervisado, ya que en este no se dispone de más información que la de los propios datos. Por esta razón, suele basarse en medidas de similaridad, para detectar los elementos más semejantes, y medidas de calidad interna, que en lugar de utilizar las etiquetas reales para medir la calidad, se basan en la compacidad y la separación de los clusters [50].

Los algoritmos de clustering suelen clasificarse según la idea de la que parten. Existen hasta un total de 6 categorías diferenciadas:

- Los algoritmos *jerárquicos* crean una estructura de clusters anidados que puede ser presentada mediante un *dendrograma*. Según la forma en la que obtienen la jerarquía, pueden ser *aglomerativos* o *divisivos*. Los aglomerativos parten de una clasificación donde cada elemento es un cluster diferente y, según una medida de distancia entre clusters, combinan iterativamente el par de clusters más cercano hasta obtener un único cluster. Algunos ejemplos de algoritmos aglomerativos son SLINK [58], BIRCH [67] o CURE [28]. En cambio, los algoritmos divisivos parten de un cluster que agrupa todos los elementos y deciden en cada paso qué particion escoger, generando nuevos clusters a cada vez. Algunos de estos algoritmos son DIANA [37] o DIVFRP [69]. El esquema jerárquico es utilizado a menudo para acompañar algoritmos de otras categorías, como HDBSCAN [15], FPDC [24] o Chamaleon [36].

- Los algoritmos *particionales* o *basados en centroides* crean una clasificación a partir de k centroides diferentes. La clase de un punto corresponderá a la clase asociada al centroide más cercano. Estos algoritmos se basan en desplazar los centroides, iterativamente, tratando de minimizar una función objetivo. Dentro de esta categoría se encuentran algoritmos como k-*means*++ [6], PAM [38] o CLARANS [61], entre otros.

- Los algoritmos *basados en rejillas* dividen el espacio en distintas regiones, calculando la densidad de cada una de ellas según el número de elementos del conjunto de datos que quedan dentro de dicha reegión. Estos algoritmos posteriormente determinan la clasificación uniendo aquellas regiones adyacentes con mayor densidad de puntos. Gracias a que no tratan cada uno de los puntos independientemente, estos algoritmos son muy eficientes para grandes conjuntos de datos. De entre ellos, destacan CLIQUE [2] y STING [64].

- Los algoritmos *basados en grafos* representan los datos mediante grafos, donde la clasificación vendrá dada por componentes conexas de este. Estos algoritmos aprovechan la teoría de grafos para poder modificarlos, combinarlos o podarlos, alterando la información de estos para obtener mejores clasificaciones. Dentro de esta categoría pueden encontrarse algoritmos como DBSGRAPH [30], GDL [68], Chamaleon [36] o AMOEBA [22].

- Los algoritmos *basados en densidad* determinan las zonas donde se encuentra una mayor cantidad de puntos en un menor espacio para determinar los clusters. Estos algoritmos tienen la ventaja de distinguir los puntos entre "ruido", "frontera" o "núcleo" según la densidad de elementos del mismo cluster que los rodean. El algoritmo que dió lugar a esta categoría fue DBSCAN [21], aunque posteriormente otros algoritmos fueron construidos sobre sus ideas, como HDBSCAN [15] o OPTICS [5].

- Los algoritmos de clustering *basados en metaheurísticas* llevan a cabo esta clasificación no supervisada empleando una metaheurística como estrategia de alto nivel para resolver el problema de clustering, que pertenece a la categoría NP-Duro. Las metaheurísticas son algoritmos inspirados generalmente en la naturaleza que tratan de encontrar soluciones de buena calidad para los problemas NP-Duro, implementando técnicas de exploración, para evitar caer en mínimos

locales, e intensificación, para mejorar iterativamente las soluciones encontradas. Existen diversas técnicas para aprovechar estas metaheurísticas dentro del clustering. Por ejemplo, algunos algoritmos se apoyan en una metaheurística para encontrar la mejor partición en un esquema divisivo, como FDPC [24]. Otra estrategia común dentro de esta categoría consiste en emplear una metaheurística para desplazar las posiciones de los centroides en un esquema particional, usando la metaheurística PSO [4] *(Particle Swarm Optimization)* o la metaheurística EPC [31]*(Emperor Penguins Colony)*. Finalmente, también se consideran los algoritmos que emplean la metaheurística directamente para obtener la clasificación, por ejemplo en CGA [33] se emplea el algoritmo genético VGA para encontrar el cromosoma que codifica la mejor clasificación, donde cada clasificación viene dada por introducir el i-ésimo elemento en la clase numerada por el i-ésimo gen del cromosoma. Debido a que la rama de las metaheurísticas es tan amplia, existen muchas posibilidades a la hora de combinar estos esquemas para resolver problemas de clustering.

Dada la complejidad de esta tarea, es necesario que se sigan desarrollando nuevos algoritmos de forma continua. A menudo, las aportaciones más recentes suelen combinar técnicas de diferentes categorías para obtener mejores resultados [1]. Por estas razones, en este trabajo se presenta el nuevo algoritmo GPGAC *(Graph Prunning based Genetic Algorithm for Clustering)*, donde se implementan algoritmos genéticos de longitud variable para identificar una secuencia de modificaciones que, tras aplicarlas sobre un grafo dirigido obtenido uniendo los k_0 vecinos más cercanos mediante arcos, generen una mejor clasificación según una medida de calidad prefijada. Cada una de estas posibles modificaciones permite denotar puntos como frontera basándose en medidas de densidad o centralidad en un grafo, podando todos los arcos salientes de este nodo. Mediante este enfoque, se consigue un algoritmo capaz de competir con el estado del arte, con ciertas ventajas añadidas, como la independencia frente a los parámetros o la flexibilidad que le otorga el poder decidir qué función objetivo se desea optimizar.

A continuación se presentan los objetivos que se buscan alcanzar en este trabajo:

Objetivos

El objetivo principal de este trabajo consiste presentar el algoritmo GPGAC, denotando cuales han sido las inspiraciones y el proceso del desarrollo y, finalmente, ponerlo a prueba frente a diferentes algoritmos del estado del arte para comprobar el éxito de este nuevo algoritmo. Los pasos que se van a seguir son los siguientes:

- Estudiar el estado del arte sobre el clustering, prestando especial atención a las nuevas ideas que incorporan cada uno de los nuevos algoritmos.
- Profundizar en aquellos conceptos de la teoría de grafos que puedan ayudar a crear un nuevo algoritmo basado en estos.
- Introducir el concepto de algoritmos genéticos en detalle y combinarlos con la teoría de grafos para el nuevo algoritmo.
- Desarrollar y definir un nuevo algoritmo cuyos resultados no dependan en gran medida de los parámetros introducidos y sea capaz de adaptarse a cada conjunto de datos.

- Comparar este con algunos de los algoritmos del estado del arte para validar los resultados y extraer conclusiones sobre la calidad del nuevo algoritmo.

Este trabajo está estructurado de la siguiente manera. En el Capítulo 2 se presenta el estado del arte, introduciendo algunos de los algoritmos más populares de clustering de cada categoría y explicando la idea de la que parten, así como las medidas de similaridad y calidad de referencia empleadas en ellos. El Capítulo 3, está dedicado a explicar el nuevo algoritmo GPGAC. En el Capítulo 4 se presentan los resultados obtenidos tras emplear este con diferentes parámetros, para posteriormente compararlo con BIRCH, HDBSCAN, k-*means*++ y DBSGRAPH, comprobando si el algoritmo es suficientemente bueno. Finalmente, en el Capítulo 5, se presentan las conclusiones obtenidas tras este trabajo y se presentan posibles futuras líneas de investigación.

Capítulo 2

Estado del Arte

El *clustering* es una rama del aprendizaje no supervisado que consiste en agrupar los elementos de un conjunto extrayendo la información a partir de los datos. Más concretamente, dado un conjunto de datos X, se busca encontrar k clases o clusters $C_1, C_2, \ldots, C_k \subset X$ de forma que cada elemento $x \in X$ sea asignado a alguna de las clases C_i. El objetivo es conseguir que cada uno de los clusters C_i tengan una gran cohesión interna, es decir, que agrupe elementos muy similares, mientras que a la vez que se diferencie del resto de clases. Una buena clasificación será aquella en la que los grupos encontrados permitan diferenciar las características intrínsecas de los elementos de cada clase.

A diferencia del clustering, en la clasificación supervisada cada dato tiene asociada una etiqueta con su clase real. Por tanto, el objetivo es definir un modelo capaz de aprender de los datos para predecir la clase de futuras instancias. En cambio, el clustering se enmarca dentro del aprendizaje no supervisado, por lo que no disponemos de la clasificación real. Por lo tanto, solo podemos emplear las propiedades que pueden extraerse de las relaciones entre los datos para obtener la clasificación. La forma más común de extraer información de los datos es mediante una *medida de similaridad*, es decir, una función que compare qué tan parecidos son dos elementos diferentes del conjunto X. De esta forma, los clusters estarán formados por grupos de elementos que presenten una gran similaridad entre ellos, pero una mayor diferencia con respecto a los elementos de otras clases.

Además, debido al carácter no supervisado, no podemos comparar las clases obtenidas con las originales, lo que provoca la necesidad de definir diferentes métricas para poder comparar diferentes clasificaciones obtenidas, llamadas *medidas de calidad*. Estas nos permitirán tener una idea acerca de la calidad de la clasificación, para poder comparar y para poder modificar los parámetros utilizados en cada algoritmo tratando de mejorar la solución final.

En esta sección se introducen algunas de las medidas de similaridad y calidad más populares, así como diferentes algoritmos de clustering, organizados según la idea de la que parten, para poder tener un trasfondo completo sobre las diferentes técnicas que se han ido usando a lo largo de la literatura hasta el día de hoy.

2.1. Medidas de Similaridad

Las medidas de similaridad son empleadas para extraer relaciones entre $x = (x_1, \ldots, x_n) \in X$ e $y = (y_1, \ldots, y_n) \in X$ cualesquiera de nuestro conjunto de datos. En general, el objetivo del clustering es agrupar los elementos que presenten una mayor similaridad entre ellos. Emplear el mismo algoritmo con diferentes medidas de similaridad puede resultar en clasificaciones diferentes. Por esta razón, existen diversas medidas y en cada caso debemos elegir la que más se adapte al conjunto de datos que queremos clasificar. A menudo, los datos toman valores reales y pueden ser representados es un espacio n-dimensional. Para estos casos, las métricas de distancia han sido muy utilizadas, debido a que representan con exactitud el concepto de cercanía que buscamos. Sin embargo, existen casos en los que se toman variables categóricas o discretas y es necesario emplear distintas medidas. A continuación se presentan algunas de las medidas más utilizadas.

- *Distancia de Minkowski:* También llamada la p-distancia, es una métrica que generaliza varias distancias como la distancia de Manhattan o métrica del taxi cuando $p = 1$, que solo considera los desplazamientos en las direcciones de los ejes, o la métrica euclídea cuando $p = 2$, que es la distancia más natural y la más utilizada en la literatura. Su fórmula viene dada por:

$$L_p(x, y) = \left(\sum_{i=1}^{n} \omega_i |x_i - y_i|^p \right)^{\frac{1}{p}},$$

  donde ω_i son los pesos asociados a la i-ésima coordenada. Estos pesos pueden ser ajustados de acuerdo a la importancia que se le da una variable sobre el resto o a la escala de esta variable. Un ejemplo para ajustar los pesos automáticamente es usando el inverso de la varianza de dicha variable dentro del dataset.

- *Distancia de Chebyshev:* Esta es la métrica que se obtiene a partir de la distancia de Minkowski en el límite cuando $p \to \infty$. Solo considera la máxima diferencia entre dos variables, dando lugar a clusters cuadrados en lugar de esféricos:

$$L_\infty(x, y) = \max_{1 \leq i \leq n} \left(|x_i - y_i| \right).$$

- *Distancia del Coseno:* Viene dada por el ángulo entre dos elementos tratados como vectores y es aplicada con éxito en conjuntos de datos que son representados mediante vectores embebidos. Un ejemplo de datos que normalmente son representados de esta forma son los problemas de clasificación de documentos [51], para los cuales se suele utilizar esta métrica. Su fórmula es:

$$\theta(x, y) = \frac{arccos(x \cdot y)}{\|x\| \, \|y\|}.$$

- *Distancia de Chord:* Esta distancia se puede aplicar cuando se tiene un conjunto de datos normalizada dentro de una hiperesfera de radio 1. Al contrario que las anteriores, la similaridad entre dos elementos es mayor cuanto mayor es la distancia de Chord. Su fórmula viene dada por:

$$d_{chord}(x, y) = \left(2 - 2 \frac{\sum_{i=1}^{n} x_i y_i}{\|x\| \, \|y\|} \right)^{\frac{1}{2}}.$$

- *Distancia de Mahalanobis:* Esta métrica es similar a la métrica euclidea donde se pondera por los coeficientes de la covarianza del dataset. Sea S la matriz de covarianza del conjunto de datos X, la distancia de Mahalanobis viene dada por la forma bilineal asociada a S:

$$d_{Mah}(x,y) = \sqrt{(x-y)S^{-1}(x-y)^T}.$$

  Esta métrica permite encontrar clusters con forma hiperelipsoidal y reduce los problemas de la correlación, adaptándose mejor a conjuntos de datos donde unas variables tienen más importancia que otra [12].

- *Distancia de Hamming:* Esta es una distancia empleada para variables categóricas donde de mide el número de posiciones en las que los elementos son diferentes. Se emplea con éxito cuando cada coordenada x_i puede tomar una cantidad discreta de valores. Por ejemplo, si la variable x_i puede tomar 5 valores y queremos que sea indiferente la distancia entre $x_i = 1$ y $x_i = 2$ que entre $x_i = 1$ y $x_i = 5$. La ecuación viene dada por:

$$d_{hamming}(x,y) = \sum_{i=1}^{n} \delta(x_i, y_i) \quad \text{donde } \delta(x_i, y_i) = \begin{cases} 0 & \text{si } x_i = y_i. \\ 1 & \text{en otro caso.} \end{cases}$$

- *Distancia de Jaccard:* Mide la similaridad entre dos datos a través del número de elementos que tienen en común, relativo al número de elementos distintos que contienen. Este coeficiente es utilizado en espacios con variables categóricas donde los datos $A, B \in X$ representan conjuntos de decisiones. Por ejemplo, si buscamos clasificar el tipo de clientes de un supermercado según el tipo de producto que compran, utilizaremos esta métrica [29].

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

2.2. Medidas de Calidad

A la hora de determinar qué tan buena ha sido la solución encontrada por el algoritmo nos encontramos con una dificultad añadida. En el caso del aprendizaje supervisado, conocemos las etiquetas reales, por tanto podemos medir la calidad utilizando medidas como la *precisión*, el *F1-Score*, el *índice de Rand* o la *matriz de confusión*. Sin embargo, en el clustering, no se conocen a priori estas etiquetas reales y, por tanto, necesitamos establecer medidas que dependan únicamente de la información de los datos y de los clusters definidos. Para ello, necesitamos medidas que valoren positivamente la cohesión dentro de cada cluster y la distancia entre clusters diferentes, así como la separación del ruido. Algunas de las medidas de calidad más utilizadas son las siguientes:

- *Suma del error cuadrático:* Esta medida se suele usar para algoritmos particionales donde se tiene un centroide y es la medida que trata de optimizar el algoritmo *k-means* [25]. Viene dada por la distancia al cuadrado con respecto de la media del cluster. Si tenemos k clusters y μ_i es la media o centroide del cluster C_i, esta medida viene dada por:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2.$$

- *Índice de Calinski-Harabasz:* Mide la compacidad del cluster mediante la distancia de los puntos a su centroide y la separación entre clusters midiendo la distancia entre los centroides con el centroide global μ_G, ambas ponderadas por sus grados de libertad [14]. Para ello, utiliza esta fórmula:

$$CH = \frac{\frac{BCSS}{k-1}}{\frac{SSE}{n-k}} \quad \text{donde} \quad BCSS = \sum_{j=1}^{k} n_i \cdot \|\mu_j - \mu_G\|^2.$$

  donde *BCSS (Between-Cluster Sum of Squares)* mide el grado de separación entre los clusters como la suma de los cuadrados entre los centroides de cada cluster con el centroide global, ponderado por $n_i = |C_i|$, y *SSE* es la suma del error cuadrático definida anteriormente.

- *Índice de Davies-Bouldin:* evalúa la separación entre clusters calculando la media de la similaridad de cada cluster con su más cercano [16]. Para determinar esta similaridad entre clusters, suma la cohesión S_i de cada uno de los clusters y divide entre las distancias entre sus centroides. Seleccionando el máximo de esta cantidad, se obtiene la similaridad con el cluster más cercano a C_i. Cuanto menor sea este índice, mayor será la calidad de la clasificación.

$$DB = \frac{1}{k}\sum_{i=1}^{k} \max_{i \neq j} \left\{ \frac{S_i + S_j}{d(\mu_i, \mu_j)} \right\} \quad \text{donde} \quad S_i = \frac{1}{n_i}\left(\sum_{x \in C_i} d(x, \mu_i) \right).$$

- *Coeficiente de Silhouette:* Establece una medida que compara qué tan similar es un objeto a los elementos de su cluster comparándolo con los del cluster más cercano. Puede tomar valores en el intervalo $[-1, 1]$ donde un valor cercano a 1 indica una buena clasificación. Para esto, compara la media de las distancias de x a cada uno de los elementos de su cluster $a(x)$ con la media de las distancias al cluster más cercano $b(x)$ empleando las siguientes fórmulas [54]:

$$a(x) = \frac{1}{n_i - 1} \cdot \sum_{y \in C_i, y \neq x} d(x, y)$$

$$b(x) = \min_{j \neq i} \frac{1}{n_j} \cdot \sum_{y \in C_j} d(x, y)$$

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}} = \begin{cases} b(x)/a(x) - 1 & \text{si } a(x) \geq b(x) \\ 1 - a(x)/b(x) & \text{si } a(x) < b(x) \end{cases}$$

  Así, $s(x)$ nos da una noción de cómo de bien clasificado está x. Para obtener una medida de calidad completa, podemos tomar la media o la mediana de los coeficientes de Silhouette de todos los elementos, descartando los considerados como ruido.

- *Índice de Dunn:* Este coeficiente mide la calidad de la clasificación basándose en el ratio de la distancia entre los dos clusters más cercanos, dividido entre la cohesión del cluster más disperso. De esta forma, observa el ratio solo en los peores casos [20]. Un mayor coeficiente de Dunn indica una mejor clasificación, y viene dado por la siguiente fórmula:

$$D = \frac{\min\limits_{1 \leq i < j \leq k} d(C_i, C_j)}{\max\limits_{1 \leq p \leq k} d'(C_p)}.$$

La distancia entre los grupos C_i y C_j puede medirse de diferentes formas, como la distancia entre sus centroides $d(\mu_i, \mu_j)$ o la distancia entre los dos elementos más cercanos $\min_{x \in C_i, y \in C_j} d(x, y)$. La distancia $d'(C_p)$, a su vez, puede ser cualquier medida de distancia de cohesión del cluster c_p, como por ejemplo, la media de las distancias con su centroide $\frac{1}{n_p} \sum_{x \in C_p} d(x, \mu_p)$ o como el diámetro del cluster $\max_{x,y \in C_p} d(x, y)$.

2.3. Algoritmos de Clustering

Dentro de los algoritmos de clustering, existen diferentes clasificaciones posibles. La principal manera de clasificar estos algoritmos es según la estrategia o idea principal que toman para encontrar la clasificación. De esta forma, tendremos los algoritmos jerárquicos, particionales, los basados en rejillas, en grafos o en densidad, y finalmente los que emplean metaheurísticas para llevar a cabo la clasificación [23]. Más adelante, se explican cada una de estas categorías, introduciendo algunos de los ejemplos más populares.

No obstante, también tiene interés tener en cuenta las siguientes definiciones acerca de este tipo de algoritmos:

- Un algoritmo de clustering se dice que lleva a cabo una clasificación *sólida* (*hard clustering*) si cada elemento $x \in X$ es clasificado dentro de un único cluster C_i, de forma que los clusters son disjuntos dos a dos, $C_i \cap C_j = \emptyset \quad \forall i \neq j$.

- En cambio, un clustering *solapados* (*overlapping clustering*) permite que un elemento pertenezca a diferentes clases. Esto será de utilidad, por ejemplo, a la hora de clasificar usuarios en redes sociales, pues buscas etiquetar a cada usuario en base a diferentes categorías.

- Una clasificación se dirá que es *parcial* (*partial clustering*) cuando es capaz de descartar elementos ruidosos, es decir, determinar puntos que no son asignados a ningún cluster. De esta forma, se pueden conseguir algoritmos más robustos al ruido e implementar mecanismos que nos permitan descartar elementos atípicos que perjudican la información extraible del conjunto de datos.

- Por último, una clasificación *difusa* (*fuzzy clustering*) será aquella en la que los clusters estén definidos mediante *conjuntos borrosos*. En este tipo de conjuntos, no se tiene una noción de pertenencia binaria. En lugar de ello, la pertenencia de cada elemento al conjunto borroso viene caracterizada por una función de pertenencia $f_A : X \to [0, 1]$. Esta función asigna la posibilidad de que un elemento de X pertenezca a dicho conjunto difuso A asignándole un valor en el intervalo $[0, 1]$, donde $f_A(x_1) = 0$ indica que el elemento x_1 no pertenece a A y $f_A(x_2) = 1$ indica que, con total seguridad, el elemento x_2 está dentro de A . Mediante estos conjuntos se puede modelizar una lógica borrosa, donde las definiciones no son tan estrictas y se tiene una mayor flexibilidad. Los algoritmos difusos permiten obtener clusters borrosos, donde esta posibilidad puede modelizar qué tan bien está caracterizado un elemento dentro de un cluster, diferenciando a los puntos más centrales, con un valor más alto, de aquellos que hacen de frontera con otros clusters. Así, resultan más flexibles y aportan más información. Además,

son aptos para compatibilizarlos con sistemas de lógica borrosa, aunque son más difíciles de interpretar.

A continuación, se presentan las categorías descritas anteriormente, explicando la idea que toman como base y algunos de los algoritmos más reconocidos dentro de cada una de ellas.

2.3.1. Algoritmos Jerárquicos

Los algoritmos de clustering jerárquicos se caracterizan por partir de una clasificación inicial que es iterativamente modificada combinando o dividiendo los clusters. Se pueden distinguir dos grandes categorías: los algoritmos aglomerativos, que parten de una clasificación muy granulada y van combinando los clusters con mayor similaridad hasta que se cumple un criterio de parada; y los divisivos, que hacen el proceso contrario, dividiendo un cluster inicial hasta obtener una partición mas granulada.

Estos procedimientos generan un conjunto de clasificaciones anidadas que pueden ser representadas mediante un dendrograma. En la Figura 2.1 se presenta un ejemplo. En cada nivel del dendrograma se presentan los subclusters que son unidos para cierto valor de similaridad. A partir de este, es posible obtener una clasificación de diferentes formas. La más sencilla consiste en seleccionar un valor de similaridad y extrayer la clasificación que subyace a este corte. Cada algoritmo debe incorporar una forma de elegir una entre las múltiples clasificaciones del dendrograma o determinar un criterio de parada para poder obtener una clasificación final, a menudo denominada *"flat clustering"*. Los algoritmos jerárquicos tienden a ser deterministas, por lo que no exploran todas las posibles clasificaciones, pudiendo quedar atrapados en óptimos locales. Además, son muy sensibles a la clasificación inicial y a las decisiones tomadas en pasos anteriores, así como al ruido o a valores atípicos. Por estas razones, estas ideas suelen partir de soluciones encontradas por otros algoritmos de diferente naturaleza, combinándolos para obtener mejores resultados.

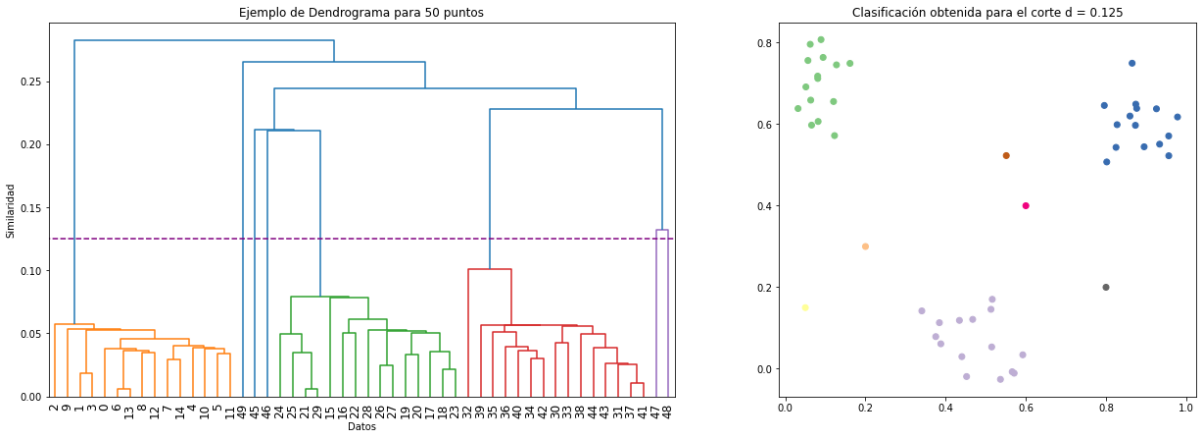


Figura 2.1: Ejemplo de dendrograma para 50 datos obtenido mediante SLINK. Fuente Propia.

El algoritmo aglomerativo más representativo es SLINK [58]. Este parte de una clasificación donde todos los puntos forman un cluster individual. Posteriormente, une los dos clusters más cercanos y los combina. La forma en la que mide la distancia

entre clusters es mediante la regla de *single-linkage*, tomando la distancia entre los dos elementos más cercanos, $d(C_i, C_j) = \min_{p \in C_i, q \in C_j} (d(p, q))$.

Más adelante surgieron otras variaciones que emplean reglas diferentes para determinar la distancia entre dos clusters. Mediante la regla *average-linkage*, esta distancia viene dada por la distancia media entre elementos de cada cluster [62], $d(C_i, C_j) = \frac{1}{n_i n_j} \cdot \sum_{p \in C_i, q \in C_j} d(p, q)$ donde $n_i = |C_i|$.

En cambio, el algoritmo CLINK [17] aplica la regla *complete-linkage*, según la cual la distancia viene dada por el diámetro que tendrá el cluster resultante tras combinar C_i y C_j. Esto se calcula como la distancia entre los elementos más lejanos, $d(C_i, C_j) = \max_{p \in C_i, q \in C_j} (d(p, q))$, y termina generando clusters más compactos.

Otros métodos más modernos toman esta distancia a través de la distancia entre los centroides μ_i como $d(C_i, C_j) = d(\mu_i, \mu_j)$. Estos centroides pueden ser calculados como el punto medio del cluster $\mu_i = \frac{1}{|C_i|} \sum_{p \in C_i} p$ [9], aunque otras variaciones calculan este centroide usando la mediana [48], siendo más robusto frente a outliers.

Los algoritmos divisivos llevan a cabo el proceso contrario. En general, se parte de un único cluster, que contiene todos los puntos, y este es iterativamente dividido en clusters más pequeños hasta alcanzar un número máximo de clusters o un tamaño mínimo de cada cluster. A diferencia de los aglomerativos, donde se busca la mejor pareja a unir, los algoritmos divisivos buscan la mejor partición posible. Debido a que el número de posibles particiones de un cluster es mucho mayor, esta técnica tiene un coste computacional mucho mayor. Por esta razón, los algoritmos divisivos generalmente incorporan estrategias para reducir esta búsqueda. El algoritmo DIANA [37] (Divisive Analisis) elige en cada paso el cluster de mayor diámetro C_i que va a ser dividido. Para ello, construye un nuevo cluster C_{k+1} extrayendo elementos de C_i. En cada iteración, para cada punto $x \in C_i$, se calcula la distancia media entre este y el resto de puntos de C_i. A esta cantidad le restamos la distancia media con los puntos de C_{k+1}, es decir:

$$t(x) = \frac{\sum_{y \in C_i \setminus \{x\}} d(x, y)}{|C_i| - 1} - \frac{\sum_{y \in C_{n+1}} d(x, y)}{|C_{n+1}|} \quad \forall x \in C_i$$

Si $\exists x \in C_i$ tal que $t(x) > 0$, escogemos el x que maximice esta cantidad y actualizamos $C_i \leftarrow C_i \setminus \{x\},\ C_{n+1} \leftarrow C_{n+1} \cup \{x\}$. En caso contrario, hemos encontrado la partición deseada, incorporamos los cambios al dendrograma y buscamos el próximo cluster a dividir.

Otro algoritmo divisivo es DIVFRP [69]. En este caso, el cluster C_i que va a ser particionado se elige como aquel con mayor desviación típica. Para elegir la partición, se elige como representante $x_r \in C_i$ al punto más alejado con respecto al centroide y se calculan las distancias del resto de puntos con respecto x_r. Posteriormente, se ordenan estas distancias de menor a mayor. Denotamos $d0 = d(x_r, x_r) = 0,\ d1 = d(x_r, x_1)$ donde $x_1 = \arg \min_{y \in C_i \setminus \{x_r\}} d(x_r, y)$, $\ldots, d_{n_i - 1}$. Definimos como frontera t al punto medio entre las dos distancias consecutivas con mayor diferencia, es decir, $t = \frac{d_{t+1} - d_t}{2}$ donde $d_t = \arg \max_{0 \leq h < n_i - 1} (d_{h+1} - d_h)$. A partir de t, podemos generar los nuevos clusters como $C_i^1 = \{x_i \in C_i \mid d_i < t,\ 1 \leq i \leq n_i - 1\} \cup \{x_r\}$ y $C_i^2 = \{x_i \in C_i \mid d_i > t,\ 1 \leq i \leq n_i - 1\}$. Este

método de partición tiene la ventaja de que es capaz de eliminar los valores atípicos de forma automática. Esto se debe a que estos puntos serán, en general, los mas alejados del centroide y por tanto se tomarán como representante x_r. Al estar alejado de los demás, la distancia que se toma como frontera $d_t = d_0$, por lo que finalmente $C_l^1 = \{x_r\}$, con un único elemento, que es reconocido como ruido.

Algoritmos posteriores tratan de solventar nuevas problemáticas. Por ejemplo, CURE *(Clustering Using Representatives)* [28] busca implementar un algoritmo capaz de trabajar con conjuntos de datos grandes de forma eficiente. CURE toma un conjunto de c representantes de cada cluster de forma que queden bien repartidos, capturando la forma del cluster. Posteriormente, estos puntos se reducen al centroide, multiplicando el vector que los separa por un factor α. La distancia entre clusters se medirá como la mínima distancia entre cada par de representantes de cada cluster. Además, incorpora el uso de estructuras de datos como montículos, que ordenan los pares de clusteres más cercanos, y *k-d trees*, que contiene los representantes de cada cluster, para hacer más eficiente la búsqueda de los clusters más cercanos. En cada paso del algoritmo, se toma el par de clusters más cercanos (almacenados en la cima del montículo) y los combina. El nuevo cluster tomará los $2c$ representantes de los clusters anteriores, los ampliará a sus puntos originales multiplicando por α^{-1} para finalmente escoger c nuevos representantes de entre ellos y reducirlos con respecto el nuevo centroide. Finalmente actualiza la distancia al resto de clusters apoyándose en el *k-d tree* para encontrar el representante más cercano y actualizar las estructuras de datos. Una vez se hayan clasificado los representantes, el resto de puntos son asignados al cluster para el cual la distancia media con sus representantes sea menor. De esta forma, este algoritmo es capaz de trabajar con bases de datos muy grandes de forma más eficiente que el resto de algoritmos.

Otro algoritmo adaptado para grandes conjuntos de datos es el algoritmo BIRCH [67]. Este es capaz de trabajar con grandes conjuntos de datos gracias a su estructura eficiente en memoria. BIRCH conserva los *"Clustering Features"* de cada cluster como una tupla $CF = (N, LS, SS)$ donde N es el número de elementos del cluster, LS es la suma de todos sus elementos y SS es la suma de sus cuadrados. Estas son todas las magnitudes que necesita el algoritmo para calcular las distancias entre clusters. Las tuplas CF de cada cluster son conservadas en un árbol equilibrado llamado *CF-tree*, donde cada nodo tiene la información CF propia y de sus descendientes. Cada descendiente representa un subcluster. Los nodos intermedios tienen B descendientes y los nodos finales tienen L clusters finales con la restricción de que su diámetro debe ser menor que un parámetro T. En cada paso del algoritmo, se inserta un elemento en el árbol, eligiendo el cluster más cercano calculando la distancia a partir de los calores de CF. Una vez se encuentra el cluster más adecuado para este punto, se actualizan los valores de CF para dicha rama y se procede a clasificar el siguiente punto. La condición de que el *CF-Tree* sea equilibrado asegura que se cumplen los límites de elementos en cada hoja. De esta forma se obtiene un algoritmo muy eficiente que obtiene buenos resultados. En este caso, la jerarquía viene determinada por los niveles del *CF-Tree*. Sin embargo, la calidad de la clasificación es muy dependiente de los parámetros y esta sesgado hacia clusters no circulares debido a que el díametro determina los límites de los subclusters [23].

Sin embargo, los algoritmos basados en distancias no siempre se adaptan bien cuando las variables son categóricas o binarias. A raíz de esta observación, surge el algoritmo ROCK [29], que está más adaptado a trabajar con variables categóricas. Para

ello, propone una medida de similaridad basada en el Coeficiente de Jaccard (véase la sección 2.1). A partir de ella, se define la noción de vecindad: dos puntos son vecinos si la función de similaridad excede un parámetro θ. El objetivo del algoritmo es encontrar clusters de forma que se maximice la suma de vecinos comunes entre elementos de un mismo cluster, minimizando el número de vecinos comunes con elementos de otros clusters. Para evitar que todos los elementos se junten bajo el mismo cluster, se divide el número de vecinos comunes por la media esperada de vecinos comunes, dada por $n_i^{1+2f(\theta)}$ [29], donde n_i es el tamaño del cluster y $f(\theta) = \frac{1-\theta}{1+\theta}$. De forma similar a CURE, este algoritmo empieza tomando un conjunto de puntos al azar donde cada punto forma un cluster unitario y los combina de forma jerárquica. La regla que sigue dicta que los clusters C_i y C_j que se van a combinar son aquellos que maximicen $g(C_i, C_j) = \frac{\#vecinosComunes(C_i,C_j)}{(n_i+n_j)^{1+2f(\theta)}-n_i^{1+2f(\theta)}-n_j^{1+2f(\theta)}}$. Esta busca maximizar el número de vecinos comunes entre elementos de ambos clusters, ponderado por el número esperado de vecinos comunes de cada cluster así como del que resultaría tras combinarlos. Finalmente, el resto de puntos son clasificados. Cada punto será clasificado al cluster C_i con el que tenga un mayor número de vecinos en común, ponderando por el número esperado de vecinos.

Debido a la flexibilidad que ofrecen los algoritmos jerárquicos, muchos algoritmos optan por incorporar técnicas que combinen o dividen los clusters obtenidos para mejorar los resultados. Por eso, también se pueden incluir en esta categoría algoritmos que son mencionados en las sucesivas secciones, por ejemplo Chameleon [36], que combina técnicas de grafos para obtener pequeños clusteres iniciales y posteriormente recombinarlos; HDBSCAN [15], que emplea una estructura de grafos basada en DBSCAN y MST para obtener una jerarquía; o FPDC [24], el cual define un algoritmo divisivo que se apoya en la metaheurística IDPSO para encontrar la mejor partición en cada iteración, entre otros.

Uno de los algoritmos que más destacan dentro de los que combinan jerarquías con otras técnicas es el algoritmo SWIFT [49] *(Scalable Weighted Iterative Flow-clustering Technique)*. Este se basa en el algoritmo EM [19] *(Expectation–Maximization)*. EM es un algoritmo de clustering probabilístico que trata de ajustar un número de distribuciones normales a los datos mediante dos etapas. En la etapa de esperanza se calcula la verosimilitud, incluyendo variables latentes, y en la etapa de maximización se trata de encontrar los estimadores de máxima verosimilitud. Iterando estas etapas se obtiene una mixtura gaussiana, es decir, un conjunto de distribuciones normales multidimensionales que se ajustan a los datos. El algoritmo SWIFT parte de estas distribuciones para formar clusters. Estos clusters van a tener una forma redondeada y a menudo se solapan, pues nacen de las mixturas gausianas encontradas en el paso anterior. Por esta razón, SWIFT combina estos clusters de forma jerárquica hasta obtener una mejor aproximación del dataset real. Este algoritmo es capaz de encontrar clusters de formas no convexas con muy buena precisión.

2.3.2. Algoritmos Particionales

Los algoritmos particionales o basados en centroides buscan cubrir el conjunto de datos X mediante una partición de k clusters, determinados a partir de un centroide c_i de forma que $C_i = \{x \in X \mid d(x, c_i) \leq d(x, c_j)\ \forall j \neq i\}$. Es decir, cada $x \in X$ pertenece al cluster cuyo centroide es el más cercano. Estos algoritmos, por lo general, toman como parámetro k el número de clases. Para empezar, generan k centroides iniciales

que se irán actualizando iterativamente con el objetivo de minimizar una medida de calidad. Esta distancia suele tomarse como la suma de errores cuadráticos SSE, definida en la Sección 2.1. Además, estas clasificaciones se pueden representar de forma visual mediante diagramas de Voronoi [63]. Un ejemplo de este puede verse en la Figura 2.2. Aquí, cada región representa la zona del espacio más cercana a un centroide y por tanto, los puntos que pertenezcan a dicha región son asignados al mismo cluster.

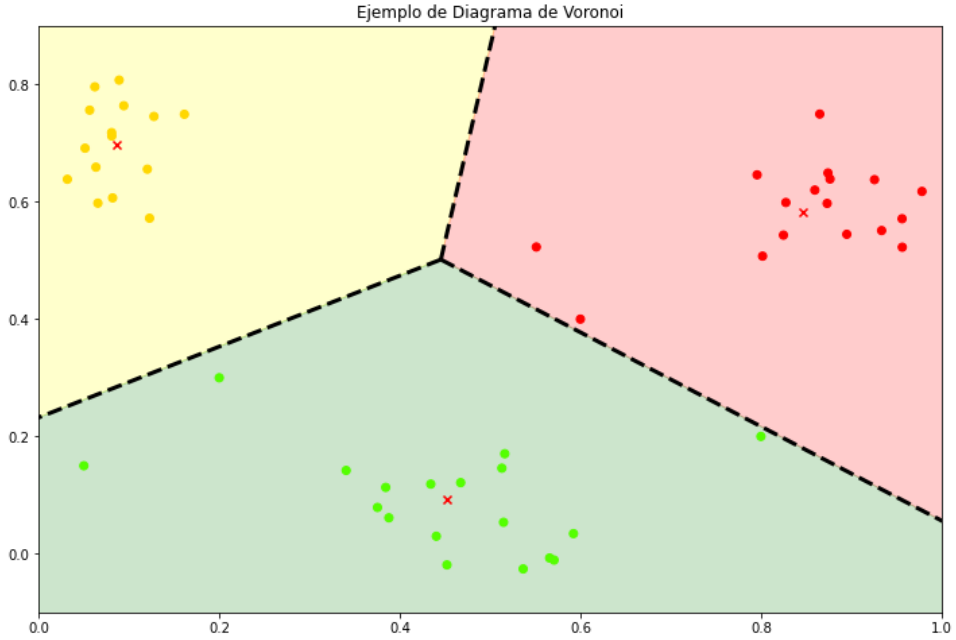


Figura 2.2: Ejemplo de diagrama de Voronoi para 50 datos obtenido mediante k-*means*. Las cruces rojas representan los centroides. Fuente Propia.

Sin embargo, estos algoritmos cuentan con una serie de inconvenientes. Dependen del parámetro k, que es crítico para obtener una buena clasificación y, en lugar de ser el algoritmo quien lo determina, debe ser determinado a priori de forma arbitraria. Además, estos algoritmos siempre generan clusters convexos de tamaños similares, por lo que no siempre se adaptan bien a todos los conjuntos de datos pues presentan fronteras lineales, que no son necesariamente realistas. Por último, la clasificación final es muy dependiente de la inicialización. Diferentes inicializaciones pueden generar clasificaciones distintas, quedando atrapados en mínimos locales. Esto se debe a que no disponen de estrategias no deterministas para aumentar el espacio de búsqueda. Esta sensibilidad puede hacer que no siempre aporten los mejores resultados, aunque son muy eficientes a la hora de determinar conjuntos linealmente separables.

Dentro de esta categoría, el algoritmo más conocido es el algoritmo k-*means* [25]. En este, se escogen aleatoriamente k elementos $x_1, x_2, \dots x_k \in X$ que serán empleados como semillas o centroides iniciales $c_1^0, c_2^0, \dots c_k^0$. A partir de este punto, se lleva a cabo un proceso iterativo basado en las siguientes etapas. En la iteración p-ésima, se calcula la distancia de cada punto $x \in X$ con cada centroide y se asigna x a la clase asociada al centroide más cercano. De esta forma, se obtienen las clases $C_1^p, \dots, C_k^p$.

Posteriormente, se calculan los nuevos centroides como la media de cada clase:

$$c_i^{p+1} = \frac{1}{|C_i^p|} \sum_{x \in C_i^p} x.$$

Este proceso se itera hasta que los centroides no cambien o se cumpla un criterio de parada. Existen dos versiones de este algoritmo, la primera versión de Forgy [25] solo actualiza los centroides una vez todos los puntos han sido clasificados en el paso p. Esto puede provocar que haya puntos que estén muy alejados del centroide c_i^{p+1} de la siguiente iteración, por lo que este no captura correctamente la información del cluster. Esta razón llevó a definir el método de MacQueen [44], el cual actualiza los centroides cada vez que un punto es clasificado. Es decir, en el paso p, se clasifica cada elemento x buscando el centroide más cercano c_i. Entonces, x se introduce en C_i^p y se recalcula el centroide recalculando la media. Solo se continua con el paso $p+1$ cuando se hayan clasificado todos los elementos. De esta forma, los centroides se desplazan múltiples veces durante la etapa de clasificación, evitando el problema anterior.

Para intentar tratar de mejorar el problema de la inicialización de este algoritmo, surge *k-means++* [6]. En esta versión, se trata de seleccionar las semillas iniciales de forma más cuidadosa para evitar caer en mínimos locales debido a la inicialización aleatoria. En este caso, se escoge un centroide c_1^0 aleatoriamente de X. El resto de centroides iniciales se seleccionará de la siguiente forma: supongameos que ya tenemos i centroides determinados, entonces para cada punto $x \in X$ se calcula $D(x) = \min_{1 \leq j \leq i}(d(x, c_j^0))$. El siguiente centroide c_{i+1}^0 se elegirá aleatoriamente siguiendo las probabilidades dadas por $p_x = \frac{D(x)^2}{\sum_{x \in X} D(x)^2}$. Este proceso se repite hasta haber seleccionado las k semillas iniciales para emplear el algoritmo mencionado anteriormente. De esta forma, se consigue que las semillas estén repartidas por el espacio, dando lugar a un menor número de clasificaciones erráticas.

Debido a la popularidad de *k-means*, surgen muchos algoritmos inspirados por este que tratan de incorporar funcionalidades nuevas. Este es el caso de *fuzzy c-means* [10], el cual combina el algoritmo *k-means* con la lógica difusa. En una clasificación difusa, cada cluster tiene asociada una "función de posibilidad", que denota la posibilidad o fuerza con la que un elemento pertenece a un cluster. El concepto de posibilidad es diferente al de probabilidad. La probabilidad modeliza la incertidumbre, no sabemos a qué cluster pertenece, pero podemos estimar esta certidumbre, además, la suma de todas las probabilidades debe sumar 1, pues pertenece a al menos un cluster. En cambio, la posibilidad modeliza la indeterminación, en la lógica difusa la pertenencia no determinista pues se considera que los conjuntos difusos tienen unas barreras más abstractas que no se pueden definir con exactitud, por esta razón la suma de posibilidades puede ser mayor que 1. Esto da lugar a que un elemento pueda pertenecer a varios clusters. El algoritmo *fuzzy c-means* determina esta pertenencia en relación a la distancia con el centroide, relativa al resto de elementos del cluster. A menor distancia, mayor es la pertenencia. El interés de esta ampliación de *k-means* reside en que, mediante la lógica difusa, pueden crearse sistemas de inferencia de reglas para, entre otras cosas, crear sistemas de control. Los algoritmos de clustering difuso pueden ser una herramienta útil para esta clase de sistemas, además de que dan una mayor información sobre los elementos del cluster que puede ser interpretada para aprender más acerca de la clasificación.

Por otro lado, algunas variaciones de k-*means* modifican la fórmula empleada para la obtención del centroide. El algoritmo k-*median* [35] es uno de ellos, donde los centroides de cada cluster se calculan mediante la mediana en vez de la media. De esta forma se puede obtener una mayor variación en el tamaño de los clusters y una mayor robustez frente a valores atípicos.

Otro popular algoritmo particional es el algoritmo PAM [38], el cual consta de dos fases. En la primera fase, denominada BUILD, se seleccionan las k semillas de la siguiente forma: Supongamos que tenemos las semillas $S = \{x_1^0, \ldots, x_i^0\}$, para decidir la siguiente semilla x_{i+1}^0, para cada $y \in X \setminus X$, calculamos $g_y = \sum_{u \in X \setminus S} \max\{D_u - d(u, y), 0\}$ donde $D_u = \min_{s \in S}(d(u, s))$. Elegiremos como nueva semilla el elemento que maximice g_y. De esta forma, si u está más cerca de y que de la semilla más cercana, entonces u contribuye a elegir a la semilla candidata y. Una vez se han seleccionado las k semillas, se lleva a cabo la fase de SWAP, donde se intercambian los elementos de S para mejorar la clasificación. Para cada paso de esta fase, se evalúa la mejora provocada por intercambiar la semilla s por el elemento h para cada par $(s, h) \in S \times (X \setminus S)$. El par que maximice la mejora es seleccionado y se realiza el intercambio. Este método tiene una particularidad con respecto a k-*means* y es que, en este caso, los centroides siempre son elementos del conjunto de datos que buscamos clasificar.

Posteriormente, el algoritmo CLARA [39] adapta las ideas de PAM para conjuntos de datos muy grandes, de forma análoga a CURE. Para ello, escoge $40 + 2k$ de elementos aleatorios del dataset y los clasifica mediante PAM. Luego emplea los centroides obtenidos para clasificar todo el dataset. Este proceso es repetido varias veces, eligiendo elementos diferentes, para quedarse con la mejor clasificación obtenida. Otra mejora vino acompañada de CLARANS [61] *(Clustering for Large Application based on Randomized Search)*, haciendo el algoritmo más eficiente y más escalable [60]. Esta versión se basa en una *Random Local Search*, donde el espacio de búsqueda son todas las posibles elecciones de semillas $S = \{x_1^0, \ldots, x_k^0\}$ y toma los parámetros $max_{neighbor}$ y num_{local}. Decimos que los conjuntos de semillas S_1 y S_2 son vecinos si solo tienen un elemento distinto, es decir, $|S_1 \cap S_2| = k - 1$. De esta forma, el algoritmo CLARANS realiza la búsqueda para cada i en el rango $0 \leq i < num_{local}$, de la siguiente manera. Parte de un conjunto S_i aleatorio y evalúa la clasificación. Después, elige un vecino aleatorio, (es decir, cambia una semilla aleatoriamente) y calcula la mejora que produce este cambio empleando la misma fórmula que PAM. Si produce mejora, se actualiza el conjunto S_i, de lo contrario, se elige un vecino aleatorio. Una vez se hayan elegido $max_{neighbor}$ vecinos sin que ninguno haya ofrecido ninguna mejora, se pasa al siguiente i. Evaluados todos los S_i, se elige la mejor clasificación. De esta forma, se realizan un total de num_{local} búsquedas locales aleatorias para quedarse con la mejor. Si el parámetro $max_{neighbor}$ fuese muy alto, el algoritmo se aproximaría más a PAM, pues se evaluarían todos los vecinos en cada paso (equivalente a evaluar todos los pares). Mientras que, tomando un valor de $max_{neighbor}$ más pequeño, podemos realizar una búsqueda más heurística pero más eficiente.

2.3.3. Algoritmos basados en Rejilla

Los algoritmos basados en rejillas o subespacios seccionan el espacio donde se encuentran los datos, creando celdas o rejillas, que son comparadas en conjunto. De esta forma, se obtienen algoritmos muy eficientes para conjuntos de datos muy gran-

des o de mucha dimensionalidad. Sin embargo, no son los más precisos, pues no incorporan información de los elementos individuales, en lugar de ello, emplean la información de la rejilla en su conjunto.

Uno de los algoritmos mas sencillos dentro de esta categoría es CLIQUE [2]. Este divide el espacio en celdas del mismo tamaño, discretizándolo para su posterior análisis. Se dice que una celda es densa si tiene más de n_0 elementos. Se empieza escogiendo una celda al azar no evaluada. Si esta es densa, formará un cluster incorporando recursivamente las celdas adyacentes que también sean densas. Combinando las celdas adyacentes se completan los clusters, obteniendo una clasificación final. Este algoritmo es muy eficiente para conjuntos de datos enormes, con muchas dimensiones y que sean separables, sin embargo la calidad de la clasificación depende del tamaño de la rejilla y casi siempre quedará atrapado en mínimos locales.

Otro popular algoritmo de esta categoría STING [64] *(Statistical Information Grid)*. En STING, la división del espacio se lleva a cabo mediante una jerarquía de celdas, donde cada nivel tiene un tamaño menor. Cada celda tiene precomputada una serie de medidas estadísticas, como el número de elementos, la media o la desviación típica, que serán empleados durante la clasificación. En este caso, la clasificación comienza en un nivel de la jerarquía y la recorre hacia abajo seleccionando las celdas que serán relevantes para la clasificación. Una vez llega al final, combina las celdas relevantes encontradas, subiendo en la jerarquía hasta alcanzar el nivel inicial. De esta forma, el algoritmo STING es más eficiente, pues no explora las celdas no relevantes, y obtiene clusters más adaptados, ya que los formar con celdas de diferentes tamaños, aunque mantiene los problemas de los algoritmos basados en rejilla.

2.3.4. Algoritmos basados en Grafos

Los algoritmos basados en grafos representan los elementos del conjunto de datos como nodos y las relaciones de similaridad entre ellos son representadas mediante las aristas. Estos algoritmos se apoyan en esta representación para extraer conocimiento a partir de la teoría de grafos para generar clasificaciones. Debido a la amplia gama de técnicas recogidas en la teoría de grafos, los algoritmos de esta categoría son muy variados, combinando a menudo técnicas de las mencionadas anteriormente. El esquema general consiste en inicializar el grafo de forma que modele los datos, aplicar alguna técnica sobre este y finalmente determinar los clusters a través de las componentes conexas. Existen diferentes formas de inicializar las aristas del grafo, las más populares son mediante un *grafo de proximidad* o conectando los k *vecinos más cercanos*. En ambas, cada elemento del conjunto X da lugar a un nodo del grafo. En el *grafo de proximidad*, se toma un parámetro α y se unen mediante aristas los vértices u, v tales que $d(u, v) < \alpha$, con un peso igual a esta distancia. Por otro lado, en el *grafo asociado a los* k *vecinos más cercanos*, cada nodo u está unido mediante una arista con cada uno de sus k vecinos más cercanos. Esta segunda forma es más eficiente ya que es posible encontrar los k vecinos más cercanos en tiempo $O(log\ n)$ empleando *KD-Trees* [8] y el número de aristas que contiene el grafo está mas controlado, ya que tiene como máximo $k \cdot |X|$ aristas. Además, como esta propiedad es no simétrica, mediante esta inicialización es posible obtener grafos dirigidos. Sin embargo, los métodos basados en grafos suelen ser menos robustos al ruido y a los valores atípicos. A menudo es interesante reducir la información del grafo, por lo que estas técnicas suelen acompañar a algoritmos jerárquicos.

El algoritmo MST [34] genera un grafo de proximidad inicial, donde cada par de nodos está unido mediante una arista con peso igual a la distancia entre los puntos. Posteriormente, este algoritmo calcula el árbol de mínimo coste usando el algoritmo de Prim. La estructura de árbol verifica que, siempre que una arista es eliminada, se obtiene una nueva componente conexa. A partir de este árbol es fácil obtener una jerarquía de clasificaciones: eliminando las aristas más pesadas y analizando las componentes conexas resultantes se obtiene una clasificación jerárquica equivalente a SLINK. Esto se debe a que, en cada paso, estamos uniendo las componentes conexas más cercanas. En cambio, es más eficiente en su ejecución. Por esta razón, es común verlo aplicado con diferentes métricas y acompañado de otras técnicas dentro de otros algoritmos, como es el caso de HDBSCAN [15].

En el caso del algoritmo CLICK [57] *(Cluster Identification via Connectivity Kernels)*, se lleva a cabo un algoritmo divisivo basado en cortes de mínimo coste y estadística. Los datos son preprocesados, calculando la media y desviación típica de la medida de similaridad entre elementos que pertenecen a un mismo cluster μ_T y σ_T, las análogas para los elementos de clusters distintos μ_F y σ_F y la probabilidad de que dos elementos aleatorios pertenezcan al mismo cluster p_T. Esto se puede hacer a partir de las etiquetas en el caso de aprendizaje supervisado, pero para el caso de no supervisado, se recomienda emplear el algoritmo EM para estimarlo. Posteriormente, cada par de nodos $u, v \in X$ son unidos por una arista cuyo peso viene dado por:

$$w(u,v) = \log\left(\frac{p_T\sigma_F}{(1-p_T)\sigma_T}\right) + \frac{(S(u,v)-\mu_F)^2}{2\sigma_F^2} - \frac{(S(u,v)-\mu_T)^2}{2\sigma_T^2},$$

donde $S(u,v)$ es la medida de similaridad entre ambos. De esta forma, los pesos de las aristas capturan las probabilidades tomadas durante el preprocesamiento. Posteriormente, se busca el corte de mínimo coste, generando dos subgrafos H^1 y H^2. Si alguno de estos grafos tiene un único elemento, se descarta y en caso contrario se procede a dividirlos recursivamente hasta que se alcance una medida de pureza medida estadísticamente. De esta forma se obtiene un algoritmo que, aunque no es el más eficiente, tiene un alto rigor estadístico y es capaz de encontrar clasificaciones con éxito en grupos de datos altamente complicados como pueden ser el de las expresiones genéticas.

Otro popular algoritmo jerárquico es Chameleon [36]. La estructura de este se presenta en la Figura 2.3. En primer lugar, se inicializa un grafo simétrico G empleando los k vecinos más cercanos. Este grafo tendrá más relaciones de las deseadas, por lo que se llevan a cabo particiones iterativamente hasta que el mayor cluster tenga un número de elementos menor que un parámetro n_min. En general, se recomienda que n_min tome valores entre un $1\,\%$ y un $5\,\%$ de los datos del conjunto inicial. Cada partición de un cluster C_i se realiza seleccionando el corte de mínimo coste que, al generar los subclusters resultantes C_i^1 y C_i^2, asegure que cada uno de los subclusters tenga al menos el $25\,\%$ de los elementos de C_i. El coste de un corte consiste en la suma de los costes asociados a las aristas que van a ser eliminadas para generar C_i^1 y C_i^2, es decir, $\sum_{u\in C_i^1, v\in C_i^2} w_G(u,v)$. Los subclusters obtenidos mediante estas particiones formarán las hojas del dendrograma. Estos se irán combinando de forma jerárquica ateniéndose a las medidas de interconectividad dentro de cada clusters y de distancia relativa entre ellos. La interconectividad entre los clusters C_i y C_j se mide como el coste del corte que genera estos clusters a partir del grafo inicial G. Esta medida será

ponderada por la media de la interconectividad propia de cada cluster. La interconectividad propia de cada cluster se mide como el coste asociado al corte que los separa en partes iguales. Para la cercanía relativa entre dos clusters se utiliza la media de los pesos que conectan los clusters. Estas dos medidas son empleadas para detectar cual es la pareja de clusters a combinar, formando así una jerarquía. De esta forma, Chameleon es capaz de encontrar clusters con formas no convexas y de diferentes tamaños, ya que la decisión para combinar los clusters se adapta dinámicamente.

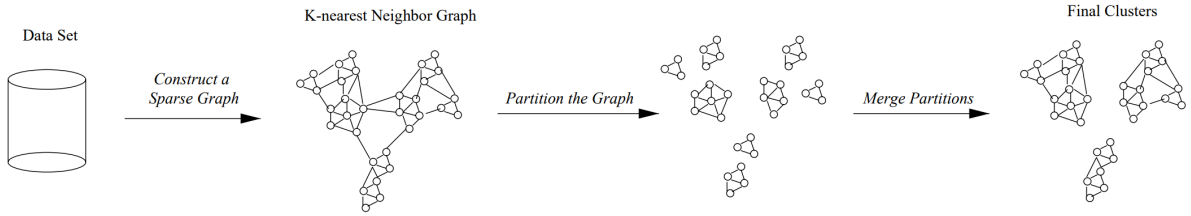


Figura 2.3: Estructura general de Chameleon. Genera un grafo con los k vecinos más cercanos, para posteriormente particionarlo en pequeños subclusters y recombinarlos de forma jerárquica. Fuente: [36].

En el caso de GDL [68] *(Graph Degree Linkage)*, se utilizan dos grafos dirigidos: El grafo G_k formado uniendo los k vecinos más cercanos y el grafo G_0, usando solo los k_0 vecinos más cercanos, donde $k_0 < k$, generalmente $k_0 \in \{1, 2\}$. El grafo G_0 es empleado para tomar una clasificación inicial que será combinada de forma aglomerativa. La clasificación inicial se obtiene a partir de las componentes débilmente conexas del grafo G_0. Para decidir qué clusters C_i y C_j van a ser combinados, se emplea una medida de afinidad basada en el grado entrante y saliente sobre el grafo G_k. De esta forma, se define el grado entrante de un vértice v con respecto al cluster C_j como la media de los pesos de las aristas de G_k que parten de un vértice $u \in C_j$ y acaban en v, es decir: $deg^-_{G_k}(v, C_j) = \frac{1}{|C_j|} \sum_{u \in C_j} w_{G_k}(u, v)$. Análogamente, el grado entrante se define como $deg^+_{G_k}(v, C_j) = \frac{1}{|C_j|} \sum_{u \in C_j} w_{G_k}(v, u)$, donde cambia la dirección de las aristas.

Así, la afinidad entre un vértice v y el cluster C_j se define como: $A_{v \to C_j} = deg^-_{G_k}(v, C_j) \cdot deg^+_{G_k}(v, C_j)$. Finalmente, podemos tomar como distancia entre dos clusters la suma de afinidades de los vértices con cada cluster dada por $d(C_i, C_j) = \sum_{v \in C_i} A_{v \to C_j} + \sum_{u \in C_j} A_{u \to C_i}$. El algoritmo GDL combina los clusters C_i y C_j que minimicen esta distancia. De esta forma, se emplea la correlación entre el grado entrante y saliente para determinar la fuerza con la que dos clusters están conectados. Además, se presenta una fórmula que permite realizar los cálculos para la afinidad de forma matricial, haciendo el cálculo muy eficiente [68]. Este sencillo algoritmo presenta unos buenos resultados y su interés reside en que presenta una forma de aprovechar los grafos dirigidos, los cuales son habitualmente ignorados en los algoritmos de clustering.

Cabe destacar que diferentes inicializaciones de grafo pueden dar lugar a algoritmos muy distintos. En el algoritmo AMOEBA [22], esta inicialización se hace a través de una triangulación de Delaunay [18]. Aquí, tres puntos forman un triángulo de Delaunay si en el interior de la circunferencia que los une no hay ningún otro punto. Mediante esta idea, se puede obtener un diagrama de Delaunay, ejemplificado en la

figura 2.4, que es posteriormente explorado por el algoritmo. Gracias a este diagrama, podemos identificar los clusters como aquellos puntos cuyas aristas son más cortas. Por eso, este algoritmo irá eliminando las aristas que tengan una distancia mayor que un límite que va siendo modificado. Primero, se calcula la media de los pesos asociados a las aristas de cada nodo x , llamada media local l_x. Posteriormente, se calcula la media global m y la desviación típica global s. Para cada nodo del grafo se establece un nivel de tolerancia $T_x = s * \frac{m}{l_x}$. Las aristas incidentes con x cuyo peso sea mayor que el límite de tolerancia $m + T_x$ serán eliminadas del grafo. Tras evaluar todos los nodos, se obtiene un subgrafo de donde podemos eliminar los nodos con grado menor o igual que 1, considerados elementos ruidosos. Finalmente, calculamos las componentes conexas para obtener los clusters. Este método obtiene clusters de tamaños similares, por lo que, para afinar más, se aplica recursivamente AMOEBA a cada uno de los subclusters hasta que este no produzca ninguna partición o se alcance un número mínimo de elementos por cluster. Este algoritmo consigue una clasificación jerárquica eficiente y de buena calidad que es capaz de encontrar clusters de formas arbitrarias gracias a basarse en la triangulación de Delaunay.

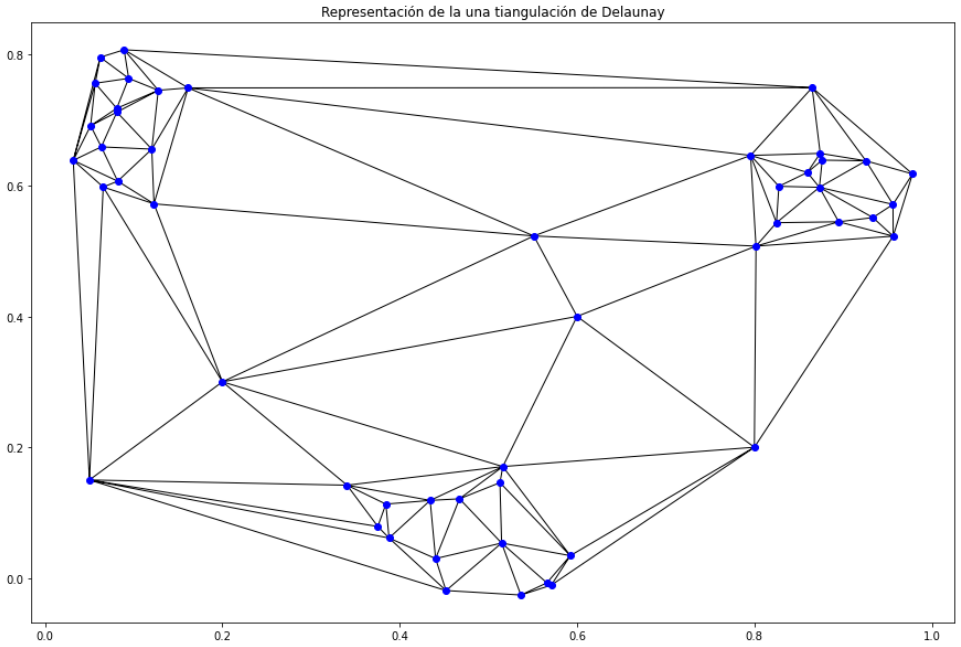


Figura 2.4: Ejemplo de triangulación de Delaunay. Las aristas más cortas indican una mayor densidad. Fuente Propia.

Los algoritmos basados en grafos también sirven para clasificar datos que son comunmente representados mediante grafos, como son los datos asociados a las redes sociales. En general, estas son representados por grados con millones de nodos. Los algoritmos vistos anteriormente no son capaces de trabajar con cantidades tan grandes de información. Por esta razón, se define el algoritmo GEM [66]. Este lleva a cabo un proceso de extracción de un esqueleto del grafo, dando mayor prioridad a los nodos que tienen un mayor grado. El esqueleto es clasificado usando una versión de k-*means* llamada *Online Weighted Kernel* k-*means*, donde la medida de similaridad está ponderada por unos pesos que se actualizan en cada iteración de forma automática, con el objetivo de adaptarse a una función objetivo. Una vez el esqueleto es clasificado, la clasificación se extrapola al resto de puntos del grafo de partida. De

esta forma se obtiene un algoritmo muy eficiente para poder clasificar grandes grafos de datos, como pueden ser los de las redes sociales. Además, este algoritmo cuenta con la ventaja de que es paralelizable. La versión paralelizada, denominada PGEM, permite encontrar clasificaciones de redes tan grandes como Twitter en apenas 3 minutos [66], empleando múltiples procesadores.

Entre los algoritmos de esta categoría, se destaca el algoritmo DBSGRAPH [30] *(Density Based Spatial Clustering for Graphs)*, el cual ha servido como punto de partida para este trabajo. Este algoritmo busca apoyarse en la teoría de grafos y las nociones de densidad con el objetivo de obtener un algoritmo que combine las ventajas de DBSCAN y *fuzzy c-means*, a la vez que reduce el número de parámetros necesarios a un único umbral α [30]. El resultado es un algoritmo que, aprovechando la medida de centralidad por cercanía en grafos (véase Sección 3.1), es capaz de encontrar una clasificación difusa, apta para conjuntos no convexos y que necesite el refinamiento de un único parámetro. Para la clasifición del conjunto X, DBSGRAPH genera el grafo de proximidad G_α para un parámetro α seleccionado, de forma que las aristas vienen dadas por aquellos elementos que, tras normalizar, disten menos que α. El peso asociado a cada arista sera igual a esta distancia normalizada. Es decir, $E = \{(x, y) \mid x, y \in X, \dot{d}(x, y) < \alpha\}$ con pesos $w(x, y) = \dot{d}(x, y)\ \forall (x, y) \in E$ donde $\dot{d}(x, y) = \frac{d(x,y)}{\max\limits_{a,b \in X} d(a,b)}$ denota la distancia normalizada. A partir de este grafo, extrae la clasificación a partir de las componentes conexas, donde todos los elementos que pertenezcan a la misma componente están en el mismo cluster. Posteriormente, podemos definir el valor de pertenencia a la clase a partir de la medida de centralidad por cercanía normalizada, es decir,

$$Clos(u) = \frac{C(u) - \min\limits_{v \in V} C(v)}{\max\limits_{v \in V} C(v) - \min\limits_{v \in V} C(v)},$$

donde $C(u)$ denota dicha medida de *closeness centrality*, dando lugar a la clasificación difusa. Este algoritmo es iterado sobre distintos valores de $\alpha \in [0, 1]$ hasta dar con la clasificación que asegure que todos los clusters tienen una calidad mayor. Para no verse tan afectado por el ruido, los elementos pertenecientes a clusters con menos de cinco elementos son considerados como ruido. Este algoritmo tiene la ventaja de encontrar conjuntos no convexos, estableciendo además un clustering difuso a partir de la información del grafo. Sin embargo, la búsqueda iterativa del mejor α^* resulta bastante ineficiente, pues dos valores de α distintos, pero suficientemente cercanos entre sí, pueden generar la misma clasificación. A pesar de ello, solo es capaz de generar una pequeña fracción de todas las posibles clasificaciones, debido a su carácter determinista. Esto lo hace propenso a quedar atrapado en mínimos locales, no siendo capaz de adaptarse a cualquier conjunto de datos y viéndose muy perjudicado cuando hay una gran presencia de ruido.

2.3.5. Algoritmos basados en Densidad

Los algoritmos particionales funcionan muy bien cuando los datos son linealmente separables pero fallan al tratar de clasificar conjuntos de datos donde las fronteras no son convexas. Por esta razón, surgen los algoritmos basados en densidad, los cuales tienden a favorecer las regiones donde hay un mayor concentración de puntos. Además, estos algoritmos son capaces de diferenciar entre los puntos núcleo frente

a los puntos frontera en cada cluster, donde los primeros son los que aglutinan una mayor densidad mientras que los frontera son los más cercanos a otros clusters. Más aún, estos son capaces de distinguir el ruido, por lo que son más robustos y habitualmente empleados para limpiar los datasets determinando estos elementos ruidosos.

El algoritmo más representativo de esta categoría es DBSCAN [21]. Este algoritmo toma 2 parámetros: un radio ϵ que delimita la vecindad y un número n_0 mínimo de vecinos para que un punto sea considerado núcelo. Es decir, un punto x es considerado un núcleo si su vecindad $V_x = \{y \in X | d(x,y) < \epsilon\}$ verifica que $|V_x| \geq n_0$. Diremos entonces que el punto núcleo x forma el cluster C_x dado por todos los puntos "densamente alcanzables" a partir de x. Se dice que un punto a es "densamente alcanzable" por x si a pertenece al vecindario de un punto núcleo z, es decir, $a \in V_z$ y existe una sucesión $y_1, ..., y_n$ de núcleos vecinos entre x y z, es decir, $y_1 \in V_x$, $y_i + 1 \in V_{y_i}$ y $z \in V_{y_n}$. Así, el cluster C_x viene dado por los puntos que están en el vecindario de algún núcleo al que se puede llegar desde x a través de otros núcleos. Junto a este algoritmo, se presenta un teorema que demuestra que independientemente de por qué elemento núcleo del cluster se empiece, se obtiene el mismo cluster [21]. Por eso, para encontrar la clasificación, el algoritmo DBSCAN procede de la siguiente manera. En primer lugar, selecciona un punto $x \in X$ al azar. Si x es núcleo, entonces se determina el cluster C_x. Este cluster estará formado inicialmente por todos los vecinos $y \in V_x$. Si $|V_y| < n_0$, entonces se añade y al cluster C_x como punto frontera. En caso de que $|V_y| \geq n_0$, entonces y se añade a C_x como núcleo, y por tanto se procede recursivamente a añadir los puntos alcanzables por y. En el caso de que x no fuese núcleo, este queda clasificado como ruido, a no ser que más tarde se detecte que pertenece al vecindario de otro núcleo. Una vez finalizado el análisis de x, se selecciona otro punto que no pertenezca a ningún cluster hasta que no queden más puntos por clasificar. Este algoritmo ha cosechado muy buenos resultados, siendo ampliamente utilizado en diversas áreas del análisis de datos, incluso suele ser utilizado para descartar los elementos ruidosos debido a la fiabilidad con la que los determina. Además, es capaz de determinar el número de clusters por sí mismo y de encontrar clusters de diferentes tamaños y formas, ya que no necesariamente deben ser convexos. Sin embargo, es propenso a dividir clusters y es muy dependiente de los parámetros introducidos: pequeños cambios en n_0 o ϵ pueden producir resultados muy diferentes. Por esta razón, se necesita mucho refinamiento a la hora de seleccionar los parámetros. Además, su mayor debilidad es que el parámetro ϵ actúa como un límite de densidad global, no permitiendo definir correctamente clusters de densidades diferentes.

Para tratar de disminuir esta dependencia de los parámetros, surge OPTICS [5]. Esta actualización dependerá únicamente del parámetro n_0, aunque opcionalmente se añade el parámetro ϵ como parámetro máximo de búsqueda. Si p es un punto núcleo, se define su "distancia de núcleo" $d_{core}(p)$ como la distancia entre p y su n_0-ésimo vecino más cercano. Posteriormente, se define la distancia de alcanzabilidad entre un punto núcleo p_c y otro cualquiera q como el $d_{reach}(p_c, q) = \max(d_{core}(p_c), d(p_c, q))$, que determina lo fácil que es alcanzar q desde p_c. El algoritmo empieza por un punto cualquiera y genera una lista de prioridad con todos los elementos de su cluster, ordenados según su distancia de accesibilidad con p. Para cada nuevo punto núcleo q añadido a la lista, se actualiza la distancia de accesibilidad de cada elemento o como $\min(d_{reach}(p,o), d_{reach}(q,o))$. De esta forma, se obtiene un "gráfico de accesibilidad" (véase la Figura 2.5 para ver un ejemplo). Este gráfico es análogo a un dendrogra-

ma, donde se pueden ver los puntos ordenados según su distancia de accesibilidad. Mediante este gráfico, se pueden seleccionar los clusters observando los valles en el gráfico. Se puede determinar hasta qué altura lo denotamos como valle, seleccionando así una clasificación jerarquíca en base a esta distancia de accesibilidad. Seleccionando una altura fija, obtenemos la clasificación que da DBSCAN para ese parámetro ϵ, mientras que si seleccionamos diferentes alturas para cada valle, podemos obtener clusters con densidad variable.

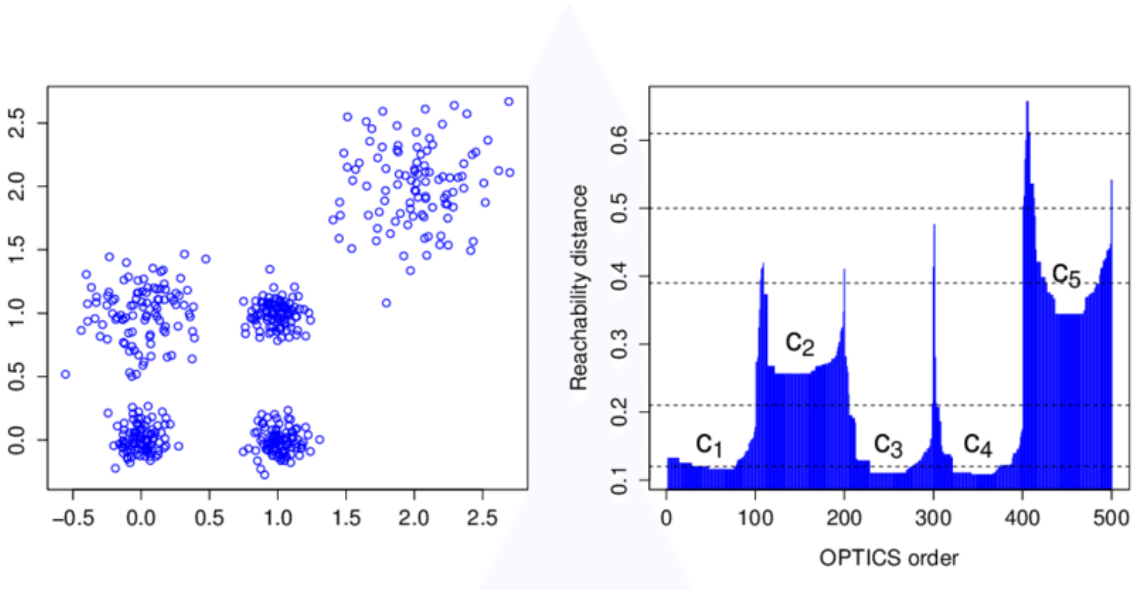
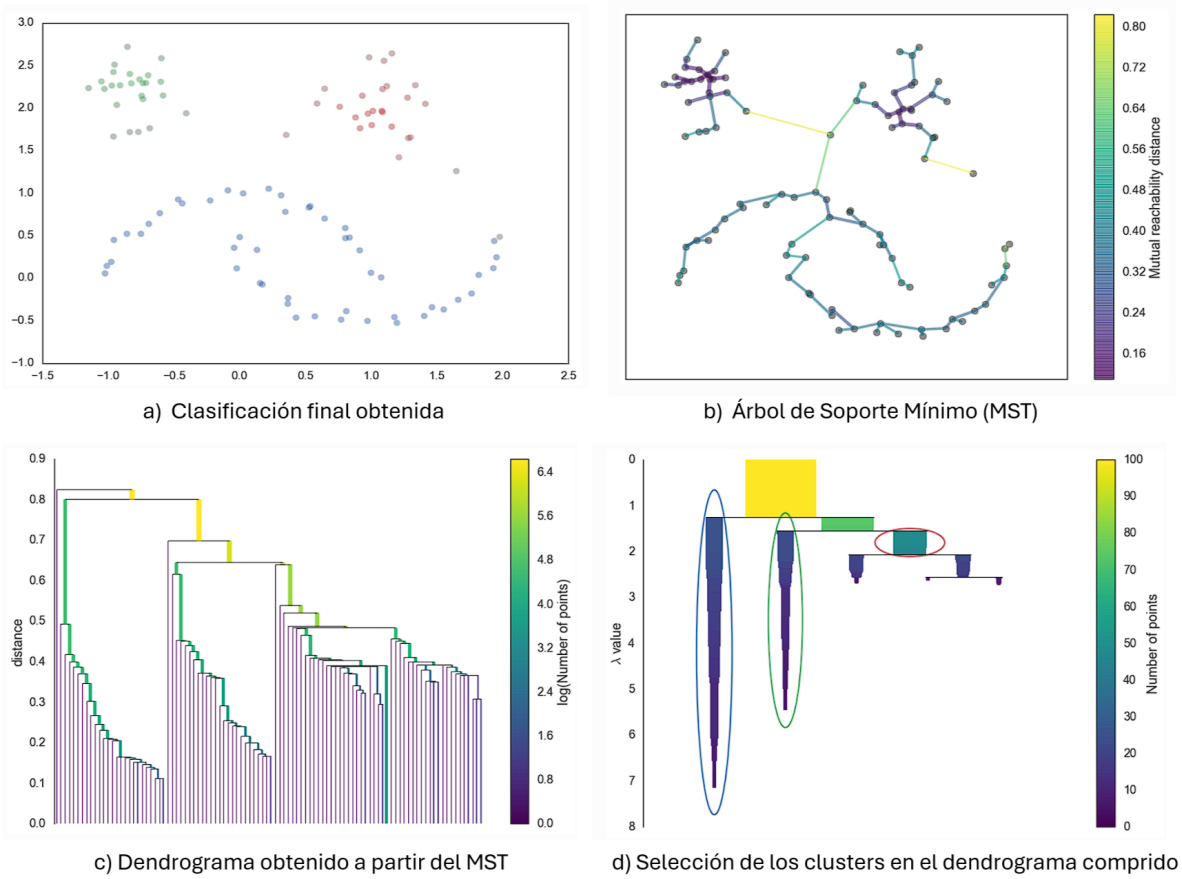


Figura 2.5: Gráfico de Accesibilidad obtenido mediante OPTICS. Los valles del gráfico identifican cada uno de los clusters, siendo aquellos con una densidad más alta los que tienen una menor altura. Fuente: [26].

Si bien OPTICS reduce la dependencia con respecto a ϵ, la selección de los clusters es ambigua y no permite discernir clusters de distinta densidad de forma precisa. El algoritmo HDBSCAN [15] establece una clasificación jerárquica basada en DBSCAN diferente, que consigue obtener clusters con diferentes densidades de manera correcta. En este caso, la distancia de accesibilidad mutua se toma como $d_{mr}(p,q) = \text{máx}(d_{core}(p), d_{core}(q), d(p,q)$, donde la distancia núcleo está definida para todos los puntos. A partir de esta medida, se genera un grafo donde las aristas tienen el peso asociado a la distancia de accesibilidad mutua. Posteriormente, se procede a realizar una clasificación jerárquica mediante MST, equivalente a *single-linkage* para esta distancia. Una vez obtenido el dendrograma, se realiza compresión de este, podando los clusters con menos de n_0 elementos. En este nuevo dendrograma más compacto, consideramos la distancia a la que cada punto es eliminado del cluster. Finalmente, necesitamos un método que permita seleccionar la clasificación a partir del dendrograma. En vez de partir por una distancia, el algoritmo HDBSCAN selecciona aquellos clusters que más perduran en la jerarquía usando una medida de estabilidad dada por $\sum_{p \in C} (\lambda_p - \lambda_C)$, donde $\lambda_p = \frac{1}{d_p}$, siendo d_p la distancia en la que el elemento p abandona el cluster y λ_C es el inverso de la distancia a la cual el cluster es generado a partir de la partición de los anteriores. De esta forma, se obtienen clusters donde la distancia empleada para determinar la densidad es diferente, solucionando este problema de DBSCAN. Además, normalizando los valores de λ_p para cada punto,

podemos obtener una medida de la posibilidad de que cada elemento pertenezca al cluster, obteniendo una clasificación difusa que permita distinguir el ruido. Un ejemplo de cada uno de los diagramas obtenidos mediante el algoritmo se puede ver en la Figura 2.6.



a) Clasificación final obtenida

b) Árbol de Soporte Mínimo (MST)

c) Dendrograma obtenido a partir del MST

d) Selección de los clusters en el dendrograma comprido

Figura 2.6: Diferentes diagramas obtenidos durante la ejecución de HDBSCAN. Obtenemos el dendrograma a partir del MST para posteriormente comprimirlo. En d), el ancho representa el número de elementos que contiene el cluster para dicho valor de similaridad. Se seleccionan los clusters más estables. Finalmente, se obtiene la clasificación vista en a), donde la trasparencia denota el grado de pertenencia. Fuente: [46].

Como hemos visto, HDBSCAN es independiente del parámetro ϵ. Sin embargo, se ha observado que establecer un parámetro $\hat{\epsilon}$ que actúa como un límite de densidad puede favorecer de cara mejorar selección de la clasificación [45]. En los casos en los que existen zonas de alta densidad que aglutinan gran parte de los elementos, puede ocurrir que algún subcluster persista durante demasiado tiempo. Si esto ocurre, al seleccionar la clasificación final, los clusters más estables serán más pequeños de lo que se buscaba inicialmente. Por esta razón, se desarrolla el algoritmo HDBSCAN($\hat{\epsilon}$) [45], donde se añade este parámetro. Se dice que un cluster C del dendrograma es epsilon-estable si la distancia a la que se produce el corte que genera C es mayor que el parámetro $\hat{\epsilon}$. Los clusters que no sean ϵ-estable se les asignará una medida de estabilidad igual a 0, por lo que no podrán ser elegidos en el proceso final. Así, aumentando el valor de $\hat{\epsilon}$, se evitan estas.

2.3.6. Algoritmos basados en Metaheurísticas

Las metaheurísticas son algoritmos de optimización generalizados que se inspiran en la naturaleza para resolver problemas de la clase NP-Duro. Para este tipo de problemas es imposible encontrar un algoritmo eficiente que encuentre la solución óptima en tiempo polinómico. Por esta razón se emplean metaheurísticas, las cuales son algoritmos, generalmente no deterministas, que implementan estrategias para explorar el espacio de búsqueda, tratando de acercarse la mejor solución del problema sin quedar atrapado en mínimos locales [55]. Estas sacrifican la seguridad de alcanzar el óptimo a cambio de una mayor eficiencia.

Debido a que el problema de encontrar la mejor clasificación pertenece a la clase NP-Duro [1], diversos autores han tratado de aproximarse a este problema mediante el uso de metaheurísticas. Para poder aplicar este tipo de algoritmos, se necesita una forma de codificar cada posible solución y una función objetivo que mida la calidad de esta, la cual vamos a tratar de maximizar. En el caso del clustering, esta función objetivo suele tomarse como una de las medidas de calidad descritas en la sección 2.2. Los algoritmos de clustering basados en metaheurísticas a menudo combinan diferentes técnicas de las vistas anteriormente. A continuación, se introducen algunos ejemplos, explicando cómo combinan estas ideas para obtener mejores clusterings.

El primer grupo de combinaciones consistiría en emplear metaheurísticas para hacer más eficiente la selección de los clusters a combinar o la partición en un algoritmo jerárquico. Este es el caso del algoritmo FPDC [24] *(Fast IDPSO-based divisive clustering)*. Este algoritmo toma inspiración del esquema de DIVFRP y busca mejorar la selección de la partición del cluster a dividir. Comienza asignando todos los elementos a un único cluster y, en cada paso, determina el cluster que va a ser dividido como aquel que tiene una mayor varianza con respecto a su centroide. De esta forma, los cortes que se realizarán serán mediante el hiperplano que genere la máxima reducción en la varianza de los dos nuevos clusters resultantes. Para determinar este hiperplano, se emplea una versión discretizada mejorada de PSO llamada IDPSO [43]. El algoritmo PSO tiene un número n_0 de partículas que se desplazan por el espacio tratando de aproximarse al óptimo global. Para ello, reciben información tanto de su mejor posición encontrada hasta la fecha como la del resto de partículas vecinas asociadas para determinar la nueva trayectoria [40]. En este caso, sea C el cluster que va a ser dividido en C^0 y C^1 y sea $N_C = |C|$, cada partícula p^j tendrá dimensión N_C. Cada partícula representa una posible división, donde $p^j = (p^j_1, p^j_2, \ldots, p^j_{N_C})$ determina que el i-ésimo elemento x_i de C estará incluido en el subcluster C^0 si $p^j_i = 0$ o en C^1 en otro caso. Las partículas se desplazarán a lo largo del plano tratando de maximizar la función objetivo asociada a la reducción de la varianza, así, taras varias iteraciones, habremos encontrado una partición cercana a la óptima. Este algoritmo se apoya en una metaheurística para decidir una partición de calidad de forma eficiente de entre todo el conjunto de posibles particiones para mejorar el algoritmo divisivo.

Otro ejemplo que combina estrategias jerárquicas con metaheurísticas se puede ver en [7]. En este caso, se divide el conjunto de datos usando la estrategia de los k vecinos más cercanos para generar un número de subclusters. Posteriormente se emplean algoritmos genéticos (GA) para combinarlos hasta obtener una clasificación final. Los algoritmos genéticos cuentan con una población de individuos, denominados cromosomas, donde cada uno representa una solución. Estos cromosomas son seleccionados de forma que se simulan los procesos evolutivos: en cada iteración, lla-

madas generaciones, del algoritmo, se seleccionan los cromosomas más "aptos" para producir una nueva descendencia aplicando una recombinación. Para determinar los cromosomas más aptos, se toma una función objetivo, y mediante la recombinación, el nuevo cromosoma creado toma características de ambos de sus progenitores, tratando de intensificar la búsqueda. Además, con una probabilidad p, tendrá lugar una mutación, esto es un cambio en el nuevo cromosoma. Este mecanismo permite explorar el espacio de búsqueda en su total y escapar de mínimos locales [59]. En este caso, se emplean algoritmos genéticos de longitud variable, donde cada cromosoma representa una posible jerarquía obtenida a partir de los subclusters. A través de la recombinación se obtiene una población que maximiza una función objetivo que coincide con una medida de calidad. Normalmente se usa el índice de Dunn o el de Davies-Bouldin [7].

Otro grupo de algoritmos basados en metaheurísticas se basan en el paradigma particional. En estos, la metaheurística es empleada para determinar la posición de los centroides y los elementos son asignados al cluster asociado al centroide más cercanos. Existen múltiples variaciones de esta estrategia usando diferentes algoritmos. Uno de estos algoritmos se puede encontrar en [31]. Este se apoya sobre la metaheurística EPC [32] *(Emperor Penguins Colony)* para determinar los centrodies. La metaheurística EPC simula en el movimiento en espiral que llevan a cabo los pingüinos emperador para distribuir el calor alrededor de la colonia. Este algoritmo es empleado para determinar una matriz de tamaño $k \times (d+1)$, donde k es el número máximo de centroides a considerar y d es la dimensión del conjunto de datos X. En este caso, se generan k centroides, uno en cada fila, de dimensión d junto con una columna extra de activación la cual puede desactivar el cluster asociado a dicha fila en caso de que no se cumpla un criterio [31]. Esta columna extra permite tener una cantidad de clusters variable. De esta forma, el algoritmo es capaz de encontrar clusters separables de forma eficiente y es más flexible que otros algoritmos particionales, pues es capaz de determinar el número de clusters de forma autónoma.

Existen más algoritmos que parten de esta idea. En [4], se emplea la metaheurística PSO para hallar estos centroides. En este caso, cada elemento de la población representa un conjunto de centroides, que son inicializados siguiendo una distribución aleatoria uniforme. Posteriormente, el algoritmo PSO es empleado para desplazar los centroides y encontrar la mejor posición para estos. A diferencia de k-*means*, este algoritmo es menos propenso a acabar atrapado en óptimos locales, ya que el no determinismo le otorga una mayor exploración, por lo que es capaz de encontrar mejores clasificaciones.

Además de los que combinan técnicas distintas, también se puede atajar directamente el problema de clasificación mediante metaheurísticas. Un ejemplo es CGA [33]. Aquí se emplea el algoritmo genético VGA para obtener directamente la clasificación. Cada uno de los cromosomas tiene dimensión $|X|+1$, donde la última coordenada representa el número máximo de clusters y el resto de coordenadas toman un valor entero entre 1 y dicho máximo, indicando a qué cluster pertenece el i-esimo elemento en la clasificación representada por dicho cromosoma. Mediante la recombinación y mutación, estos cromosomas exploran todo el espacio de búsqueda, tratando de hallar la mejor clasificación de acuerdo al coeficiente de Silhouette, empleado como función objetivo. Este algoritmo consige explorar el enorme espacio de búsqueda de todas las posibles clasificaciones de forma eficiente, introduciendo además la posibilidad de tener un número variable de clusters.

De forma similar, en [41] se aplica la metaheurística DE *(Diferential Evolution)* para explorar este espacio. Los algoritmos de DE son muy parecidos a los algoritmos genéticos y siguen el mismo tipo de idea, ya que también se inspiran en los principios de la evolución de Darwin. En este caso, los cromosomas son inicializados aleatoriamente. Cada uno de estos representa a qué cluster es asignado el elemento que corresponde con su coordenada. Este algoritmo incorpora un paso local tras la recombinación basado en una iteración de k-*means* con el objetivo de mejorar la nueva población de cara a las futuras generaciones. En este caso, el algoritmo se aplica para resolver el problema asociado a las WSN *(Wireless Sensor Networks)*, donde se busca conectar múltiples sensores para cubrir todas las necesidades minimizando el consumo de energía.

La rama de las metaheurísticas es una rama muy amplia, por lo que existen numerosas formas de combinar estos algoritmos para aplicarlos al clustering. En esta sección, hemos visto algunos de los paradigmas más populares. Estos consisten en emplear la metaheurística dentro de un esquema jerárquico para mejorar la selección de los clusters, usarlas para desplazar los centroides para obtener un clustering particional o atajando el problema directamente, codificando soluciones completas e iterándolas para explorar el espacio de búsqueda de posibles clasificaciones.

Capítulo 3

GPGAC: Algoritmo Genético para Clustering basado en Poda de Grafos

La motivación para la realización de este trabajo surge a partir de analizar algunas de las limitaciones que tiene el algoritmo DBSGRAPH. Este busca definir un algoritmo que dependa de un número mínimo de parámetros, combinando nociones de densidad y grafos para obtener una clasificación difusa [30]. Mediante la unión de estas técnicas, este algoritmo trata de resolver la dependencia frente a los parámetros de DBSCAN [21], el cual cuenta con dos parámetros ϵ y n_0 muy sensibles, donde pequeñas fluctuaciones en ellos pueden dar lugar a clasificaciones de peor calidad. Sin embargo, DBSGRAPH solo aprovecha las técnicas asociadas a la teoría de grafos a la hora de obtener la pertenencia difusa para los clusters. Además, si bien este algoritmo ha conseguido reducir el número de parámetros a uno único, α, este sigue siendo muy dependiente de él y, para algunos conjuntos de datos, puede que no haya ningún valor de α que le permita escapar de mínimos locales.

Por esta razón, se propone el nuevo algoritmo GPGAC *(Graph Prunning based Genetic Algorithm for Clustering)*, donde se busca aprovechar la teoría de grafos de forma más completa, reduciendo la dependencia de los parámetros y aprovechando la exploración de los algoritmos genéticos para poder escapar de mínimos locales con mayor frecuencia. Este algoritmo se apoya en la metaheurística de los algoritmos genéticos para seleccionar la mejor secuencia de *modificaciones* sobre un grafo inicial dado por los k_0 vecinos más cercanos, obteniendo una clasificación que optimice una función objetivo, donde dicha función es una medida de calidad. Las modificaciones sobre el grafo están inspiradas en la forma en la que el algoritmo AMOEBA [22] poda las aristas, mejorando la clasificación. En este caso, la poda de aristas se realiza en base a las medidas de centralidad introducidas a continuación. Estas medidas son las empleadas por DBSGRAPH para obtener la pertenencia difusa, en cambio, en este algoritmo son aprovechadas para determinar qué nodos pueden ser considerados como frontera, descartando las aristas que partan de ellos.

En este capítulo se van a introducir los conceptos de la teoría de grafos para el algoritmo, en la Sección 3.1. A continuación, se describen en detalle los algoritmos genéticos en la Sección 3.2. Finalmente, la Sección 3.3 está dedicada a definir de

forma completa el algoritmo GPGAC.

3.1. Nociones previas sobre grafos

Un *grafo no dirigido* es un par $G = (V, E)$ donde V es un conjunto de vértices o nodos y E es el conjunto de aristas, cada *arista* viene dada por un par no ordenado de dos vértices (u, v). Dos nodos $u, v \in V$ se dicen que son *adyacentes* si la arista (u, v) que los une está en E. Denotamos por $Adj(v) = \{w \in V \mid (v, w) \in E\}$ al conjunto de nodos que son adyacentes a v y denotamos como *grado* de v al cardinal de este conjunto, $g(v) = |Adj(v)|$. En el caso de los *grafos dirigidos*, consideramos el orden de los pares en E, denominando a estos como *arcos* en lugar de *aristas*. Es decir, en un grafo dirigido $G_d = (V_d, E_d)$, un *arco* $(u, v) \in E_d$ es un par ordenado que tiene como origen u y como destino v. Denotaremos por $Adj^+(v) = \{w \in V_d \mid (v, w) \in E_d\}$ y $g^+(v) = |Adj^+(v)|$ al conjunto de *sucesores* y al *grado saliente* de v respectivamente, donde consideramos únicamente los arcos que tienen a v como origen. Análogamente, el conjunto de *predecesores* de v y su *grado entrante* son denotados por $|Adj^-(v)|$ y $g^-(v)$, donde consideramos las aristas que tienen a v como destino.

A cada una de las aristas se le puede asignar un *peso*, denotado como $w(u, v)$. La *matriz de adyacencia* de un grafo G viene dada por $A = (a_{i,j})_{i,j=1}^{N}$ donde $N = |V|$ y $a_{i,j} = w(v_i, v_j)$. Finalmente, un *triángulo* viene dado por una terna de nodos $u, v, w \in V$ que están unidos en el grafo dos a dos, es decir, $(u, v), (v, w), (w, u) \in E$. Denotamos el número de triángulos a los que está conectado v como $T(v)$. Este se calcula sumando el número de vecinos comunes que tiene con cada nodo adyacente entre dos, ya que cada triángulo es sumado dos veces: $T(v) = \frac{1}{2} \sum_{w \in Adj(v)} |Adj(v) \cap Adj(w)|$.

Un *camino* entre dos vértices $u, v \in V$ en un grafo es una sucesión de aristas $(a_1, \ldots, a_k)$ concatenadas, es decir, $a_1 = (u, w_1),\ a_2 = (w_1, w_2),\ \ldots\ ,\ a_k = (w_{k-1}, v)$, y denotamos el *coste del camino* como $c(a_1, \ldots, a_k) = \sum_{i=1}^{k} w(a_i)$. La *distancia en* G entre dos nodos $u, v \in V$, denotada por $d_G(u, v)$ viene dada por el camino de mínimo coste entre estos nodos. En un *camino binario*, se considera el coste de todas las aristas igual a 1, por tanto, su coste es igual al número de aristas k que forman el camino. Un grafo se dice que es *conexo* si existe un camino para cada par de puntos. Una *componente conexa* C_i de G es un subgrafo maximalmente conexo de G. Es decir, $C_i = (V_i, E_i)$, donde $V_i \subset V$ y $E_i = \{(u, v) \in E \mid u, v \in V_i\}$, es una componente conexa si el subgrafo C_i es conexo y si al añadir cualquier otro nodo de V, no existe ninguna arista en E que permita hallar un camino con este nuevo nodo.

Dentro de la teoría de grafos, existen diferentes medidas para determinar la importancia de los nodos a partir de la información de este. Estas pueden ser empleadas para determinar la densidad en torno a un nodo. A continuación, se presentan algunas de las más populares:

- *Centralidad de grado:* Determina la densidad en torno a un nodo como la fracción de los nodos a los que está conectado:

$$C_{degree}(v) = \frac{g(v)}{|V|}.$$

- *Centralidad por cercanía:* Mide la centralidad de un nodo v como el inverso de

la media de la distancia del camino más corto entre v y cada uno de los nodos de su componente conexa:

$$c_{closeness\ centrality}(v) = \frac{n-1}{|V|-1} \frac{n-1}{\sum_{w \in CC(v)} d_G(v,w)},$$

donde $n-1 = |CC(v)|$ y el factor $\frac{n-1}{|V|-1}$ lleva a cabo una corrección para darle un menor valor a aquellos nodos de componentes conexas más pequeñas [65].

- *Centralidad por autovector:* Mide la centralidad del nodo x_i como la i-ésima componente del autovector x asociado al autovalor λ de máximo módulo de la matriz de adyacencia A del grafo. Este verifica que la centralidad de x_i es la suma de la centralidad de sus nodos adyacentes [11]. El autovector x verifica que

$$A \cdot x = \lambda \cdot x.$$

- *Centralidad por caminos:* También llamada *betweenness centrality*, determina la centralidad de un nodo $v \in V$ a través del número de caminos binarios más cortos entre cada par de nodos $s, t \in V \setminus \{v\}$ [13]. Para ello, suma la fracción del número de caminos más cortos que pasan por v entre el número total de estos:

$$c_B(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma(s,t|v)}{\sigma(s,t)},$$

donde $\sigma(s,t)$ es el número de caminos binarios de mínimo coste entre s y t mientras que $\sigma(s,t|v)$ es el número de estos caminos que pasan por v.

- *Coeficiente de Clustering:* Viene dado por la fracción de triángulos de los que forma v con sus vecinos de entre todos los posibles triángulos que puede formar según su grado. Un mayor valor de este coeficiente indica que los vecinos de v están muy interconectados. Viene dada por:

$$c_{clustering} = \frac{2 * T(u)}{g(v) \cdot (g(v)-1)},$$

donde el máximo número de triángulos de los que puede formar parte u es $\binom{g(v)}{2} = \frac{g(v) \cdot (g(v)-1)}{2}$.

Por otro lado, existen variantes de este coeficiente de clustering que tienen en cuenta el peso de las aristas que forman los triángulos [56]. De esta forma, se puede definir a través de la media geométrica de los pesos del triángulo, recibiendo el nombre de "intensidad":

$$c_{weighted\ clustering} = \frac{1}{g(v_i) \cdot (g(v)-1)} \sum_{j,k} (w_{ij} w_{jk} w_{ki})^{\frac{1}{3}},$$

donde $w_{ij} = w(v_i, v_j)$ y los vértices v_i, v_j, v_k forman un triángulo.

Análogamente, existen medidas que permiten estimar la calidad de una conexión entre dos nodos. Mediante estas medidas se puede determinar si se debe añadir o eliminar una arista entre dos nodos [42]. Así, permiten añadir más aristas informativas al grafo o determinar qué aristas son más frágiles. Algunas de las más destacadas son:

- *Coeficiente de Jaccard:* El valor de la conexión entre los nodos u, v viene dado por el coeficiente de Jaccard aplicado a sus conjuntos de adyacencia:

$$l_{Jaccard}(u, v) = \frac{|Adj(u) \cap Adj(v)|}{|Adj(u) \cup Adj(v)|},$$

  es decir, el número de vecinos comunes ponderado por la cantidad de vecinos posibles.

- *Índice de Adamic-Adar:* El valor de esta conexión viene dado por la suma del número de vecinos comunes, penalizando a aquellos vecinos que tienen muchas conexiones:

$$l_{AdamicAdar}(u, v) = \sum_{w \in Adj(u) \cap Adj(v)} \frac{1}{\log |Adj(w)|}.$$

  Nótese que w debe de tener al menos dos vecinos, por lo que no se tienen divisiones entre 0.

- *Centralidad por vecinos comunes:* El valor de calidad asociado a un par de nodos $u, v \in V$ viene dado por una combinación lineal entre el número de vecinos comunes y el inverso de la distancia entre ellos:

$$l_{common}(u, v) = \alpha \cdot |Adj(u) \cap Adj(v)| + (1 - \alpha) \cdot \frac{|V|}{d_g(u, v)},$$

  donde α denota el peso que queremos darle a los vecinos comunes frente a la distancia y la distancia esta ponderada por el número de nodos en el grafo [3].

3.2. Algoritmos Genéticos

Los algoritmos genéticos son una de las categorías de metaheurísticas más populares. Este tipo de algoritmos evolutivos ha sido aplicado con éxito en diversas áreas de la inteligencia artificial, desde la optimización, la clasificación e incluso para el aprendizaje de parámetros en redes neuronales. Estos toman como fuente de inspiración los procesos biológicos descritos por la teoría Darwinista de la evolución. En la naturaleza, cada especie es el resultado de innumerables mutaciones, donde aquellas que permiten una mayor supervivencia se transmiten a las nuevas generaciones, mientras que las mutaciones que tienen un efecto negativo son descartadas por medio de la selección natural. El resultado de este proceso evolutivo es una amplia diversidad de individuos especializados a las condiciones de su entorno [59].

Los algoritmos genéticos intentan simular este proceso de forma simplificada. Para ello, se toma una población P de *cromosomas*, donde cada cromosoma codifica una solución del problema que se quiere resolver. El objetivo de estos algoritmos es replicar el proceso de selección natural que se produce en la naturaleza mediante los operadores de selección, cruce (o recombinación), mutación y reemplazo para obtener nuevas poblaciones de cromosomas mejor adaptados. Para determinar cuales son los mejores cromosomas, se tiene una *función objetivo*, la cual se busca maximizar o minimizar, y aquellos cromosomas más aptos serán los que consigan un mejor valor de esta función. En la Figura 3.1 se presenta un diagrama que representa el funcionamiento de estos algoritmos. A continuación, se detallan cada uno de los procesos involucrados en la mejora de los cromosomas.

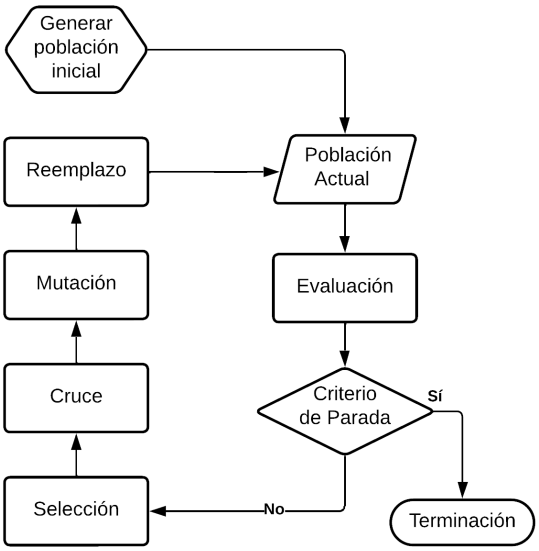


Figura 3.1: Diagrama de un algoritmo genético. Fuente Propia.

- *Selección.* Mediante este mecanismo, se seleccionan dos elementos para posteriormente obtener una nueva solución a partir de ellos mediante el mecanismo de cruce. El objetivo que los cromosomas con un mejor valor de la función objetivo tengan más probabilidades de ser seleccionados, mientras que a la vez, los cromosomas menos aptos también tengan la posibilidad de reproducirse para mantener una diversificación. Existen diversas estrategias de selección, algunas de las más utilizadas son:
  - *Selección mediante un orden lineal.* La selección se realiza ordenando las soluciones según el valor de la función objetivo y tomando las k mejores soluciones.
  - *Selección por ruleta.* A cada cromosoma se le asocia una probabilidad según la calidad de su solución, de forma que las mejores soluciones tengan una mayor probabilidad de ser elegidas y, posteriormente, se escoge un número de cromosomas de forma aleatoria siguiendo estas probabilidades.
  - *Selección por torneo.* Los cromosomas son agrupados aleatoriamente en k subconjuntos llamados "torneos" y de cada torneo se escoge un número m de ganadores, que corresponderán con las soluciones que tengan un mayor valor de la función objetivo.
- *Cruce.* Este mecanismo busca simular la reproducción de dos cromosomas seleccionados. Dados dos cromosomas, se obtiene una descendencia a partir de ellos recombinado las soluciones. El objetivo consiste en generar nuevas soluciones a partir de ellos que compartan algunas de las características de sus progenitores. También existen diferentes estrategias de cruce, de las cuales algunas serán mejores para un tipo de problemas que otras. Algunos ejemplos son:
  - *Cruce por un punto.* Se determina un índice j de corte y se intercambian las secciones de los cromosomas a partir de dicho índice. De esta forma se

obtienen dos nuevos cromosomas donde cada cual comparte las j primeras componentes con uno de los progenitores y el resto con el otro progenitor.

- *Cruce Aritmético.* El nuevo cromosoma se obtiene mediante una suma ponderada de los progenitores, por ejemplo la media aritmética. Suele ser utilizado cuando el cromosoma toma valores numéricos reales.
- *Cruce Uniforme.* Cada una de las componentes del nuevo cromosoma es seleccionada de forma aleatoria entre dicha componente de sus progenitores. Esta estrategia permite una mayor diversidad y es aplicable a todo tipo de cromosomas, pero puede llegar a producir un gran numero de soluciones idénticas si la probabilidad de mutación es baja.

- *Mutación.* Simulando las mutaciones que ocurren en la naturaleza, este mecanismo es aplicado tras el cruce y permite, de forma aleatoria, introducir alguna modificación en un subconjunto pequeño de las partículas generadas. Esto permite ampliar la diversificación y alcanzar zonas que puedan no haber sido exploradas por las generaciones anteriores.
- *Reemplazo.* Finalmente, se seleccionan los cromosomas que van a formar parte de la nueva generación. Estos pueden ser seleccionados de entre los generados en la fase de cruce y mutación, aunque también pueden tomarse alguno de la generación anterior. En este segundo caso, se habla de elitismo, el cual consiste en tomar uno o varios de las mejores cromosomas de la población previa, garantizando que no haya un pérdida de calidad.

La población inicial puede tomarse de forma aleatoria, siguiendo una distribución uniforme sobre el espacio de búsqueda. En cada iteración del algoritmo se llevan a cabo estos pasos con el objetivo de ir mejorando las soluciones obtenidas, hasta que se cumpla un criterio de parada, por ejemplo un máximo de iteraciones. Finalmente, se devuelve la mejor solución encontrada hasta la fecha y ese será el resultado del algoritmo.

3.3. El algoritmo GPGAC

Una vez introducidos los conceptos necesarios de teoría de grafos y algoritmos genéticos, se presenta el algoritmo GPGAC *(Graph Prunning based Genetic Algorithm for Clustering)*. En primer lugar, se presenta la idea general del algoritmo y como este ha sido desarrollado para, finalmente, detallar el funcionamiento paso por paso del algoritmo.

3.3.1. Idea general

La inspiración de este algoritmo nace de las limitaciones observadas al estudiar otros algoritmos de clustering basados en grafos, como DBSGRAPH [30] o AMOEBA [22]. En DBSGRAPH, se generaba un grafo de proximidad inicial y, a partir de él, se obtenía una clasificación difusa donde la pertenencia viene determinada por las medidas de centralidad por cercanía de los nodos, aunque el grafo permanecía inalterado. Por otro lado, AMOEBA parte de un grafo obtenido mediante una triangulación de Delaunay e iterativamente elimina aristas del grafo, desconectando el grafo para obtener nuevas clasificaciones conexas más finas. Sin embargo, la decisión sobre qué aris-

tas eliminar dependía únicamente del peso de ellas, no capturando la información completa del grafo.

Una aproximación que resolviese ambas situaciones podría consistir en la modificación del grafo inicial a partir de estas medidas de centralidad. De esta forma, los puntos que tuviesen un valor de centralidad bajo podrían ser identificados como ruido o como puntos frontera. Los puntos frontera deberían ser aquellos puntos que separan un cluster de otro, por lo tanto, estos deben estar unidos al núcleo del cluster, pero a partir de ellos no debería incorporarse ningún nuevo nodo. Una forma de llevar esto a cabo es mediante el uso de grafos no dirigidos, de esta forma, cuando un punto es clasificado como frontera, podemos eliminar los arcos salientes de esto. De esta forma, dicho punto frontera pertenecerá a un cluster únicamente si existe un punto núcleo del que partiese un arco hasta él.

La forma de inicializar un grafo dirigido a partir de un conjunto de datos más eficaz es mediante los k vecinos más cercanos G_k. Sea $u \in X$, denotamos por $\text{Vec}_k(u) = \{v_1, v_2, \ldots, v_k\} \subset X$ a los k vecinos más cercanos de u, es decir, $d(u, v_i) \leq d(u, v_{i+1}) \forall 1 \leq i \leq k-1$ y $d(u, v_k) \leq d(u, x) \forall x \in X \setminus \text{Vec}_k(u)$, entonces el grafo G_k vendrá dado por $G_k = (X, E_k)$ donde $E_k = \{(u, v) | u \in X, v \in \text{Vec}_k(u)\}$, es decir, el conjunto de arcos está formado por los arcos de origen en cada $u \in X$ y destino cada uno de sus k vecinos más cercanos. Esta relación no es simétrica, dando lugar a la posibilidad de definir un grafo dirigido. Además, al no depender de una distancia absoluta, es más adaptable al conjunto de datos que DBSGRAPH. Además, empleando *KD-Trees*, hallar los k vecinos más cercanos de un nodo tiene una complejidad $O(k \cdot N \log N)$, donde $N = |X|$, lo que es más eficiente que calcular la matriz de distancias, con orden $O(N^2)$.

En la Figura 3.2 puede observarse un ejemplo de un grafo obtenido mediante esta inicialización. Se observa que todos los nodos tienen un grado de salida igual a k. En cuanto al grado de entrada, los nodos más centrales, como los nodos 3 y 10, tienen un grado de entrada mayor. Tras calcular una medida de centralidad, se decide que los nodos 1, 7 y 16 deben ser considerados como frontera. Entonces, se eliminan los arcos salientes de cada uno de estos nodos. En el caso del nodo 1, como no está tan alejado del resto del cluster, hay arcos que inciden en él. Por lo tanto, a pesar de ser considerado como frontera, sigue permaneciendo dentro del cluster. En cuanto al nodo 7, este actúa como puente entre los dos clusters, por lo que tendrá una menor medida de centralidad y por tanto será considerado como frontera. Al eliminar sus arcos salientes, como ningún nodo del cluster izquierdo incide en él, los clusters terminan separados. Finalmente, en el caso del nodo 16, como $g^-(16) = 0$, al ser considerado como frontera queda completamente desconectado, pasando a ser considerado ruido.

Esta idea permite tener una base sobre la cual modificar el grafo para mejorar la clasificación inicial dada por las componentes débilmente conexas. El proceso consistiría en, dado un umbral μ y una medida de centralidad, calcular qué nodos tienen un valor de centralidad menor que μ y considerarlos como frontera, eliminando los arcos salientes. Este proceso puede ser muy sensible a la determinación de μ, no siendo práctico que este sea determinado como un parámetro fijo. A su vez, al eliminar arcos del grafo, las medidas de centralidad de los nodos que permanezcan dentro pueden cambiar. Esto da lugar a poder repetir este proceso una vez más, con un nuevo umbral μ'. Se plantea entonces la pregunta de determinar cuál es número de

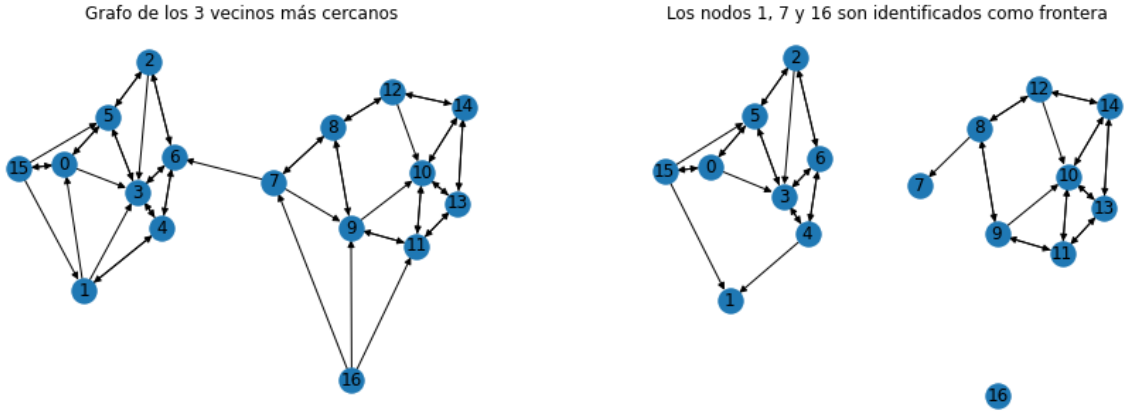


Figura 3.2: Ejemplo de grafo de los tres vecinos más cercanos y la clasificación de los puntos 1, 7 y 16 como frontera.

iteraciones y los valores umbrales que encuentren la mejor clasificación final.

El espacio de posibles combinaciones de umbrales, teniendo en cuenta además que tendremos una cantidad variable de ellos, es inabarcable. Abordar este problema evaluando todas las posibilidades es una tarea irrealizable. Por eso, se ha decidido tratar esta búsqueda a través de una metaheurística. La metaheurística por la que se optó fueron los algoritmos genéticos de longitud variable. La razón de esta elección se debe a su adaptabilidad y la posibilidad de usar cromosomas de distintas longitudes. En este caso, cada cromosoma codifica una secuencia de pasos con sus respectivos umbrales. Además, debido a que será el algoritmo quien decida qué pasos seguir y qué umbrales emplear, es posible añadir funciones diferentes, como por ejemplo otras medidas de centralidad; reducir la longitud máxima α de las aristas presentes en el grafo, eliminando las que unan nodos demasiado alejados y permitiendo descartar elementos ruidosos; o modificando el parámetro k_0 de forma dinámica, haciendo que el algoritmo sea independiente de este parámetro, entre otras cosas.

En conclusión, la idea general de GPGAC se puede resumir en los siguientes pasos. Se parte de un grafo G_0 asociado a los k_0 vecinos más cercanos. Un algoritmo genético trata de encontrar el mejor cromosoma, donde cada cromosoma codifica una secuencia de funciones sobre el grafo eliminando arcos hasta obtener el grafo G_i. El valor de la función objetivo de cada cromosoma viene dado por el índice de Calinsky-Harabasz calculado sobre la clasificación obtenida mediante las componentes débilmente conexas del grafo G_i tras la aplicación de estas funciones. Mediante los mecanismos de selección, cruce, mutación y reemplazo, se crean nuevas generaciones de cromosomas mejor adaptados. Finalmente, la clasificación asociada al mejor cromosoma encontrada es la clasificación final que devuelve GPGAC.

3.3.2. Implementación de GPGAC

Una vez explicada la idea general del algoritmo, procedemos a explicar los detalles formales de este. El pseudocódigo completo de este algoritmo se presenta en el Algoritmo 1. Las descripciones de cada uno de los pasos se encuentran a continuación.

- **Codificación de las secuencias:** Este algoritmo cuenta con una población P de n_0 cromosomas, donde cada cromosoma codifica una secuencia de pasos a

seguir dada como una lista de pares. Cada uno de estos pares viene dado por una función f y un umbral μ. La clasificación codificada por cada uno de estos cromosomas se obtiene aplicando la secuencia de funciones iterativamente a partir del grafo inicial G_0. Tras aplicar todas las funciones, se obtiene el grafo G_i asociado al cromosoma c_i. A partir de las componentes débilmente conexas de G_i, se obtiene la clasificación, donde las componentes conexas con menos de 5 elementos son consideradas ruido.

- **Parámetros e inicialización:** GPGAC cuenta con un parámetro propio k_0 relativo al grafo inicial, además de los parámetros n_0, max_it, p, m y e asociados al algoritmo genético. A la hora de inicializar el algoritmo, se procede calculando el grafo dirigido G_0 asociado a los k_0 vecinos más cercanos. La población inicial de cromosomas P será generada creando n_0 cromosomas iniciales. Para cada nuevo cromosoma c_i, se elige una longitud escogiendo un entero aleatorio l entre 1 y 10, dando mayor probabilidad a los valores pequeños, ya que estos resultarán en un proceso más eficiente en tiempo. Posteriormente, se generan l genes para el cromosoma, donde cada gen es un par (f_i^j, μ_i^j) con $1 \leq j \leq l$. Estos pares se forman con f_i^j un "movimiento" escogido al azar y μ_i^j un umbral pequeño para este movimiento. Además, el cromosoma guarda su propio valor de k, el cual toma como $k \leftarrow k_0$ cuando es inicializado. En cuanto al resto de parámetros, max_it denota el número de generaciones que se calcularan con el algoritmo, p es la probabilidad de mutación, m el número de elementos seleccionados por torneo y e el número de elementos preservados mediante elitismo. Estos parámetros tendrán un impacto bajo en la calidad de los resultados del algoritmo, en especial k_0, el cual actúa únicamente como una referencia inicial, pero podrá ser modificado por los cromosomas, como veremos más adelante. De esta forma, el algoritmo será capaz de adaptarse sin necesidad de afinar los parámetros manualmente de forma precisa.

- **Evaluación de la función objetivo:** Para evaluar cada cromosoma c_i, en primer lugar se extrae el grafo G_i. Para calcularlo, tomamos $G_i^0 \leftarrow G_0$ y posteriormente, aplicamos los movimientos codificados por c_i de la siguiente manera: $G_i^j = f_i^j\left(G_i^{j-1}, \mu_i^j\right) \forall 1 \leq j \leq l$. Finalmente, $G_i = G_i^l$. Sobre este grafo final G_i, se calculan las componentes débilmente conexas, obteniendo la clasificación asociada a c_i. El valor de la función objetivo de c_i vendrá dado el resultado de una medida de calidad calculada sobre esta clasificación. En este caso, la medida de calidad por la que se ha optado es el *índice de Calinski-Harabasz*, ya que es la medida más robusta frente a *outliers* y ruido [27]. Por tanto, el objetivo final del algoritmo es generar nuevos cromosomas con el objetivo de maximizar esta función objetivo.

  Los denominados "movimientos" son funciones que, al ser aplicadas al grafo con un umbral, devuelven este grafo tras haberlo modificado. Los movimientos que se han considerado pueden modificar los parámetros asociados al cromosoma, considerar puntos como frontera según una medida de centralidad o eliminar arcos según las medidas de predicción entre conexiones de nodos. Un punto que pasa a ser considerado frontera elimina todos sus arcos salientes. También se impide que nuevos arcos puedan partir desde él. Así, este punto pertenecerá a un cluster solo si existe un arco incidente en él que parte de un núcleo. Los movimientos considerados por el algoritmo son los siguientes:

- *Cambiar k.* Este movimiento recibe un entero μ, positivo o negativo, como umbral. El cromosoma modifica su valor de k como $k \leftarrow k + \mu$, añadiendo o eliminando los arcos correspondientes. Cuando μ es positivo, se añaden los arcos de $\{(v, w) \mid v \in X \setminus F_i, w \in \text{Vec}_{k+\mu}(v) \setminus \text{Vec}_k(v)\}$, que van desde v hasta su i vecino más cercanos, para cada $k < i \leq k + \mu$ de cada punto v que no es frontera, donde F_i denota el conjunto de puntos considerados como frontera por el cromosoma c_i hasta el momento. De esta forma, se fortalecen las uniones de los puntos que salen de los núcleos. En caso de que u sea negativo, se eliminan los arcos de $\{(v, w) \mid v \in X \setminus F_i, w \in \text{Vec}_k(v) \setminus \text{Vec}_{k+\mu+1}(v)$, asociados a los i-ésimos vecinos más cercanos para $k + u < i \leq k$ de todos los puntos. Mediante este movimiento, el algoritmo puede modificar el parámetro k_0 de forma automática, reduciendo en gran medida la dependencia frente a este parámetro, o aumentar la información del grafo en un momento concreto.

- *Reducir α.* Debido al carácter discreto de la inicialización mediante los vecinos más cercanos, no se delimita las distancia máxima de estas. Por esta razón, el grafo captura la misma información que DBSGRAPH. Para tratar esta situación, se añade este movimiento que permite descartar los arcos de mayor peso. Para ello, dado un umbral $\mu \in (0, 1)$, se toma w_{max} igual al peso del arco más largo en el grafo. Denotamos $l = \mu \cdot w_{max}$, donde l va a ser el límite superior de las distancias permitidas en el nuevo grafo. De esta forma, se eliminan todas las aristas (v_1, v_2) tales que $w(v_1, v_2) \geq l$. Por tanto, mediante este movimiento, se eliminan todas las aristas que tengan un peso superior a una fracción μ del mayor arco de peso en el grafo.

- *Frontera mediante centralidad por cercanía.* Mediante este movimiento, se declaran como frontera aquellos puntos que presenten un valor de la medida de *closeness centrality* por debajo de un cierto umbral μ, después de normalizar. Para ello, se calcula la *closeness centrality* de cada nodo $v \in X$ del grafo, denotamos este valor como r_v. Sean $a = \max_{v \in X}(r_v), b = \min_{v \in X}(r_v)$, entonces, comprobamos para qué nodos se cumple que $\frac{r_v - b}{a - b} < \mu$. Los nodos que verifiquen esto, son considerados como puntos frontera, añadiéndolos a F_i, eliminando todos sus arcos salientes e impidiendo que se vuelvan a añadir arcos cuyo origen sea este nodo.

- *Frontera mediante la longitud media de las aristas.* De forma análoga, se considerarán como puntos frontera aquellos nodos para los cuales la longitud media de los arcos que inciden en él sea mayor que un umbral μ. Denotamos por $d_u = \frac{1}{g(u)} \sum_{v \in Adj(u)} w(u, v)$, donde no distinguimos los arcos según la dirección. Aquellos nodos v para los que $\frac{d_v - b}{a - b} > \mu$ son denotados como frontera, donde $a = \max_{v \in X}(d_v), b = \min_{v \in X}(d_v)$. Este movimiento, inspirado en AMOEBA permite detectar puntos como frontera siguiendo un criterio diferente al anterior. Estos deben complementarse para dar al algoritmo un abanico más grande de posibilidades.

- *Eliminar aristas mediante el coeficiente de Jaccard.* En este caso, el movimiento permite descartar aristas directamente, sin necesidad de determinar un punto como frontera. Para ello, se calcula el índice de Jaccard $j_{u,v}$ para cada par de nodos unidos en el grafo. De forma análoga, tomamos

$a = \max_{(u,v)\in E}(j_{u,v}), b = \min_{(u,v)\in E}(j_{u,v})$ y eliminamos todas las aristas (u, v) para las cuales $\frac{j_{u,v}-b}{a-b} < \mu$. Este movimiento permite una forma diferente de reducir la información en el grafo, además separa los clusters que estén unidos por elementos con pocos vecinos comunes. De la misma manera, el objetivo es que estos movimientos se complementen y el algoritmo los seleccione en el orden que dé lugar a una mejor clasificación.

- *Eliminar aristas mediante la centralidad por vecinos comunes.* Análogamente al caso anterior, eliminamos las aristas que, tras normalizar, tengan un valor para la centralidad por vecinos comunes menor que μ. Debido a que esta medida toma un parámetro α adicional que regula el peso del número de vecinos comunes frente a la centralidad, este parámetro también es introducido en la tupla asociada a cada gen para que el algoritmo determine su mejor valor. De esta forma, una tupla para este movimiento vendrá dada por $(f, (\mu, \alpha))$. Este movimiento tiene más en cuenta la distancia entre los nodos, frente al carácter discreto de Jaccard que solo observa el número de conexiones entre ambos. De esta forma, estos movimientos pueden complementarse al ser empleados por el algoritmo.

- **Selección:** En cada paso de recombinación, se seleccionan dos cromosomas como progenitores mediante una selección por torneo. Para esta, se toma un subconjunto de m de cromosomas de la población P, de forma aleatoria sin repetición. El elemento con mejor valor de entre estos m es seleccionado y se le denota como c'_1. Posteriormente, se vuelve a seleccionar un subconjunto de m cromosomas de $P \setminus \{c'_1\}$, y denotamos como c'_2 al mejor de este subconjunto. Así, c'_1 y c'_2 serán los elementos seleccionados como progenitores para la fase de cruce. Este mecanismo permite asegurar un grado de variabilidad, mientras que se seleccionan los cromosomas más aptos. Para entender el efecto del parámetro m, cuando $m = n_0$, siempre se tomarían los dos mejores cromosomas para la selección, pues todos los cromosomas están dentro del torneo. En cambio, cuando $m = 2$, puede seleccionarse incluso el penúltimo peor cromosoma, si este entra en competición con el peor de todos. Por tanto, un valor mayor de m favorece la intensificación, debido a que existen mayores probabilidades de tomar un buen elemento entre los m seleccionados, mientras que valores más pequeños favorecen la exploración. Para este algoritmo, se ha seleccionado $m = 3$.

- **Recombinación o cruce:** Los algoritmos genéticos se basan en recombinar los dos cromosomas elegidos mediante la selección, con el objetivo de producir una descendencia más adaptada y diversa. En este caso, al emplear cromosomas de longitud variable, el mecanismo de recombinación debe ser capaz de tratar con estos casos. Para ello, se usa la recombinación por cruce en un punto. Dados los dos cromosomas progenitores $a = (a_1, a_2, \ldots, a_r)$, $b = (b_1, b_2, \ldots, b_s)$, tomamos $k = \min(r, s)$ y elegimos un entero aleatorio $1 \leq t \leq k$. Así, la descendencia vendrá dada por $d_1 = (a_1, \ldots, a_t, b_{t+1}, \ldots b_s)$ y $d_2 = (b_1, \ldots, b_t, a_{t+1}, \ldots, a_r)$. En el caso en el que $s = r = 1$, se toma como descendientes $d_1 = (a_1, b_1)$ y $d_2 = (b_1, a_1)$. De esta forma, podemos tratar cromosomas de diferentes tamaños, manteniendo siempre una parte sustancial de los pasos y el orden codificados por los progenitores, pero favoreciendo una descendencia más diversa.

- **Mutación:** La mutación es el mecanismo de exploración por excelencia de los algoritmos genéticos, imprescindible para realizar una búsqueda más completa

del espacio de soluciones. Para explorar todos las secuencias de este algoritmo, la mutación debe encargarse también de modificar el tamaño de los cromosomas, por lo que funciona de la siguiente manera: cada vez que un nuevo descendiente es obtenido por recombinación, con una probabilidad p, este descendiente recibe una mutación. Esta consistirá en la modificación de uno de los genes de forma aleatoria. Cuando un cromosoma es mutado en este algoritmo, se elige una de entre cuatro posibles mutaciones: decrecer, crecer, cambiar un movimiento o cambiar un umbral. Cuando el cromosoma decrece, se elimina uno de los genes aleatoriamente, mientras que cuando crece, se añade un gen nuevo en una posición aleatoria de la secuencia. Estas mutaciones permiten modificar el tamaño de los cromosomas y explorar sobre la longitud de estos. En los otros casos, la mutación puede o cambiar el movimiento que se encuentra en una posición al azar por otro distinto, tomando un nuevo umbral; o únicamente cambiar el umbral, sumando un valor obtenido de una distribución normal. La probabilidad de escoger cada tipo de mutación es ligeramente diferente, los cromosomas más cortos tienen una menor probabilidad de encoger, mientras que todos los cromosomas tienen una mayor tendencia a cambiar solo el umbral o cambiar el movimiento. Esta decisión se debe a que estas mutaciones son más ligeras, de esta forma nos aseguramos que los saltos que damos no son demasiado grandes para no perjudicar la intensificación.

- **Reemplazo:** Cada vez se genera una nueva población, los e individuos con mejor valor de la función objetivo son preservados para la próxima generación. Este mecanismo, denominado elitismo, permite asegurar que no se empeore la solución en las próximas generaciones. Una vez se preservan los e mejores elementos, se generan el resto de descendientes empleando los mecanismos de selección, recombinación y mutación hasta alcanzar un tamaño igual a la población inicial. En este caso, $e = 2$.

Así, el algoritmo queda completamente definido. Sin embargo, cabe destacar una serie de detalles cuyo objetivo es mejorar la eficiencia. En primer lugar, algunas de las medidas de centralidad no son replicables o producen resultados diferentes al emplearlas en grafos dirigidos. Por esta razón, todas las medidas se calculan sobre el grafo no dirigido asociado, aunque modifican el grafo dirigido directamente. En el caso de la centralidad por cercanía, esta medida requiere hallar los caminos más cortos dentro del grafo, lo que es muy poco eficiente. Por esta razón, se ha decidido emplear la distancia real entre cada par de elementos de la misma componente conexa, en lugar de restringirse al camino más corto sobre el grafo. Este cambio ha permitido reducir el coste en tiempo en gran medida y obtener resultados más acorde con lo esperado.

Por esta razón, para evitar tener que realizar una copia del grafo no dirigido asociado, en la práctica el algoritmo trabaja sobre un grafo no dirigido. Para conseguir replicar estas definiciones de frontera en el grafo no dirigido, se debe ser muy cuidadoso a la hora de eliminar las aristas. Para ello, cuando un nodo v es considerado frontera, las aristas descartadas serán aquellas (v, w) donde la conexión de w hasta v sea más débil. Consideramos que la conexión de w a v es más débil cuando v es el j-ésimo vecino más cercano de w y $j > k$, o si w es también punto frontera. De esta forma, se simula el grafo dirigido, sin la necesidad de acudir a copias cada vez que se calcula una medida de centralidad.

Mediante este esquema algorítmico, GPGAC cuenta con una población de cromosomas, donde cada uno codifica una secuencia de modificaciones al grafo. Aplicando esas modificaciones al grafo inicial, se obtiene un nuevo grafo del cual se halla una clasificación. Aplicando los mecanismos de selección, recombinación y mutación, podemos generar max_it generaciones, donde cada generación trata de adaptarse mejor al conjunto de datos en cuestión. Una vez se han completado todas las generaciones, el algoritmo devuelve la clasificación asociada al mejor cromosoma como clasificación final. Mediante este algoritmo, se ha conseguido combinar con éxito las nociones de grafos, densidad y metaehurísticas para generar un algoritmo capaz de obtener resultados competitivos. Además, este algoritmo es está definido de forma que presente una dependencia con respecto de sus parámetros muy débil, ya que estos influyen más en la eficiencia que en el resultado final obtenido.

Algoritmo 1 GPGAC(k_0)

Calcular G_0 como el grafo asociado a los k_0 vecinos más cercanos
Inicializar la población $P \leftarrow \emptyset$
for $i = 0$ to n_0 **do** ▷ Introducimos n_0 cromosomas en P
  $c_i \leftarrow inicializar\_cromosoma()$
  $P \leftarrow P \cup \{c_i\}$
end for
for $it = 0$ to max_it **do** ▷ Evaluación de los cromosomas
  for $c_i \in P$ **do** ▷ Modificación del grafo G_i
    $G_i \leftarrow G_0$
    for $t = 0$ to $c_i.long$ **do**
      $(f, \mu) \leftarrow c_i[t]$
      $G_i \leftarrow f(G_i, \mu)$ ▷ Aplicar la función f con umbral μ a G_i
    end for
    $CC \leftarrow weak\_connected\_components(G_i)$
    $c_i.labels \leftarrow \emptyset,\ C_{-1} \leftarrow \emptyset$
    for $cc_p \in CC$ **do** ▷ Obtención de la clasificación a partir de G_i
      $k \leftarrow 0$
      if $|cc_p| \leq 4$ **then** ▷ Si tiene menos de 5 elementos es considerada ruido
        $C_{-1} \leftarrow C_{-1} \cup cc_p$
      else
        $C_k \leftarrow cc_p,\ c_i.labels \leftarrow c_i.labels \cup \{C_k\}$
        $k \leftarrow k + 1$
      end if
      $c_i.labels \leftarrow c_i.labels \cup C_{-1}$
      $c_i.value \leftarrow f_{obj}(c_i.labels)$ ▷ f_{obj} es el índice de Calinki-Harabasz
    end for
  end for
  $P' \leftarrow \emptyset$
  for $j = 0$ to e **do** ▷ Aplicamos elitismo
    $P' \leftarrow Pelit \cup best(P \setminus P')$
  end for
  while $|P'| < n_0$ **do**
    $T_1 \leftarrow random\_sample(P, m)$ ▷ Selección por Torneo
    $c^1 \leftarrow best(T_1)$
    $T_2 \leftarrow random\_sample(P \setminus \{c^1\}, m)$
    $c^2 \leftarrow best(T_2)$
    $c'_1, c'_2 \leftarrow cruce(c^1, c^2)$ ▷ Recombinación
    if $random(0, 1) < p$ **then** ▷ Mutación a cada nuevo cromosoma
      $c'_1.mutar()$
    end if
    if $random(0, 1) < p$ **then**
      $c'_2.mutar()$
    end if
    $P' \leftarrow P' \cup \{c'_1, c'_2\}$
  end while
  $P \leftarrow P'$ ▷ Reemplazo
end for
return c_{best} el cromosoma con el mejor valor encontrado

Capítulo 4

Resultados

Con el objetivo de determinar la validez del nuevo algoritmo GPGAC, este va a ser utilizado para clasificar conjuntos de datos con diferentes características, como el grado de separación entre clusters, el número de dimensiones o la cantidad de ruido. A su vez, se realizará una comparación de este con algunos algoritmos representativos del estado del arte para comprobar que se hayan alcanzado los objetivos de este trabajo. Se elegirán algoritmos de diferentes categorías para la comprobación, siendo los elegidos: BIRCH, *k-means++*, HDBSCAN y DBSGRAPH. Para los tres primeros, se ha utilizado la implementación de scikit-learn [52], mientras que para DBSGRAPH se ha utilizado la implementación original [30].

Los conjuntos de datos que van a ser clasificados para esta labor han sido generados mediante el algoritmo implementado en la librería *genCluster* [53] de R, por Weiliang Qiu y Harry Joe. Este lleva a cabo una mejora del algoritmo creado por Milligan [47] para la generación de datasets para clustering, empleado a menudo para la comparación de diferentes algoritmos, como puede observarse en [27]. Esta mejora consiste en la implementación de un parámetro, que denominaremos σ, para indicar el grado de separación entre cada cluster C_i y su cluster vecino más cercano C_j. Además, el algoritmo asegura que los clusters no sean fácilmente detectables mediante únicamente dos dimensiones, aplicando una rotación a la matriz de covarianza. La ventaja de emplear datasets generados por este método para comparar algoritmos reside en que, modificando los parámetros asociados a la separación, al ruido o a los tamaños de los clusters, podemos dar con conjuntos de datos que pongan a prueba nuestro algoritmo, permitiendo determinar sus puntos fuertes y sus flaquezas [53].

Mediante este método, se han generado conjuntos de datos diferentes de dimensiones, con el número de dimensiones $D \in \{2,\ 5,\ 10\}$. Además, los datasets con 10 dimensiones incluyen una dimensión ruidosa, es decir, una dimensión que no contribuye a la separación de los clusters. Los conjuntos de dos dimensiones nos permitirán obtener gráficas para visualizar el resultado. Para cada una de estas dimensiones, se generan 4 datasets con los valores de separación y ruido dados por los pares $(\sigma, r) \in \{(0.02, 0.5\,\%), (0.05, 1\,\%), (0.1, 2\,\%), (0.2, 2\,\%)\}$. El nombre de cada dataset viene dado por el número de dimensiones, seguido de una D mayúscula y el valor de σ, de forma que "2D005" hace referencia al dataset de 2 dimensiones y grado de separación $\sigma = 0.05$. Los clusters con un mayor grado de separación serán más fáciles de distinguir, por lo que se les añade una mayor penalización en forma de ruido y así comprobar la eficacia del algoritmo en presencia de este. Para cada clasificación, se

calcula el coeficiente de Calinsky-Harabasz, de Silhouette y de Davies-Bouldin para poder comparar la calidad según diferentes métricas, observando para qué conjuntos se comporta mejor el algoritmo. Además, se ha generado un conjunto de datos con clusters no convexos, denominado "2DNoConvexo", mediante la librería scikit-learn [52], para comprobar cuales de estos algoritmos tienen la capacidad de detectar clusters no convexos. Los datasets empleados así como el algoritmo GPGAC pueden encontrarse en el siguiente enlace de GitHub.

4.1. Resultados de GPGAC sobre datasets con diferentes características

En primer lugar, se va a analizar los resultados que obtiene GPGAC sobre cada uno de los datasets. El objetivo de esta sección es comprobar las diferencias que presenta este cuando cambiamos algunos parámetros, para determinar su robustez y sus puntos fuertes.

Los resultados del algoritmo van a ser comparados según tres medidas de calidad. Como el algoritmo trata de encontrar la clasificación que optimice una de ellas, el primer punto donde vamos a iterar el algoritmo es sobre qué medida de calidad toma GPGAC como función objetivo. Para ello, cada dataset va a ser clasificado por el algoritmo, variando la función objetivo sobre estas métricas. Los resultados se encuentran representados en la Tabla 4.1. En cada una de las filas de ella, se presentan los valores de clasificación según las tres medidas de calidad, donde la métrica resaltada en negrita es aquella que ha sido empleada como función objetivo. A su vez, resaltamos el valor óptimo obtenido para cada medida de calidad en cada uno de los dataset, donde el índice de Davies-Bouldin debe ser minimizado y las otras dos métricas deben ser maximizadas.

Podemos observar que en la amplia mayoría de casos el valor óptimo de cada medida es alcanzado cuando se usa dicha medida como función objetivo. Se concluye que GPGAC es capaz de optimizar su función objetivo de forma eficaz. Por tanto, en el caso de querer optimizar una medida de calidad diferente, podríamos utilizar este algoritmo para hallar una clasificación buena según esa medida. Las pocas excepciones pueden darse a casos en los que el algoritmo acaba atrapado en un óptimo local, por ejemplo en 2D005, donde el óptimo para índice de Calinski-Harabasz es encontrado usando el coeficiente de Silhouette como función objetivo. También se observa que, a menudo, una mejor clasificación para una de las métricas no siempre implica una mejor clasificación para las demás. Esto se ejemplifica claramente usando el índice de Davies-Bouldin como función objetivo. De esta forma, siempre se obtiene el óptimo para este índice, aunque se obtienen valores muy bajos para el índice de Calinski-Harabasz. A la vista de estos datos, se ha decidido que la función objetivo final sea el índice de Calinski-Harabasz ya que, en general, es la más consistente y la que menos empeora a las demás.

Una de las razones principales por las que los valores difieren al usar diferentes funciones objetivo es el comportamiento frente al ruido de cada una de las medidas de calidad. En la práctica, puede tratarse el ruido de dos formas. La primera forma consiste en calcular el valor para la medida únicamente usando los elementos clasificados, ignorando aquellos identificados como ruido, de forma que no se recibe ninguna penalización por descartar elementos. El segundo método consiste en incluir el ruido

Tabla 4.1: Resultados de GPGAC sobre todos los datasets, donde se ha empleado la medida de calidad en negrita como función objetivo. Aquí, σ es el grado de separación entre clusters, r es el porcentaje de ruido y D es el número de dimensiones.

		$\sigma = 0.02$ $r = 0.5\,\%$	$\sigma = 0.05$ $r = 1\,\%$	$\sigma = 0.1$ $r = 2\,\%$	$\sigma = 0.2$ $r = 2\,\%$
$D = 2$	**Calinski-Harabasz**	**612.019**	555.913	**1224.673**	**970.729**
	Silhouette	0.389	0.523	**0.579**	**0.658**
	Davies-Bouldin	1.039	2.210	0.998	0.458
	Calinski-Harabasz	0.022	**627.642**	1183.785	**970.729**
	Silhouette	**0.583**	**0.542**	0.564	**0.658**
	Davies-Bouldin	35.300	0.553	0.480	0.458
	Calinski-Harabasz	8.674	3.814	7.638	663.402
	Silhouette	0.553	0.303	0.534	0.595
	Davies-Bouldin	**0.296**	**0.475**	**0.326**	**0.417**
$D = 5$	**Calinski-Harabasz**	**166.123**	**186.047**	**345.512**	**409.305**
	Silhouette	0.263	0.252	0.329	**0.485**
	Davies-Bouldin	1.922	1.776	1.617	0.793
	Calinski-Harabasz	1.741	5.520	321.651	275.628
	Silhouette	**0.304**	**0.440**	**0.379**	0.484
	Davies-Bouldin	2.573	**0.410**	1.918	1.448
	Calinski-Harabasz	2.752	5.520	3.925	4.288
	Silhouette	0.265	**0.440**	0.334	0.403
	Davies-Bouldin	**0.577**	**0.410**	**0.485**	**0.467**
$D = 10$	**Calinski-Harabasz**	**65.290**	**73.651**	**85.429**	**112.258**
	Silhouette	0.200	0.169	0.158	**0.366**
	Davies-Bouldin	2.446	2.343	2.696	1.131
	Calinski-Harabasz	6.677	5.533	8.425	77.696
	Silhouette	**0.500**	**0.451**	**0.540**	0.352
	Davies-Bouldin	**0.372**	**0.413**	**0.336**	1.490
	Calinski-Harabasz	6.677	5.533	8.425	2.025
	Silhouette	**0.500**	**0.451**	**0.540**	0.200
	Davies-Bouldin	**0.372**	**0.413**	**0.336**	**0.685**

como otro cluster al calcular la medida. En la Figura 4.1 se representan los resultados que se obtiene al tratar el ruido de estas dos formas para diferentes clasificaciones sobre el mismo dataset, donde el número de elementos identificados como ruido es diferente. Para cada clasificación, se muestra el valor del índice de Calinki-Harabasz utilizando ambos tratamientos del ruido. En la primera gráfica, la clasificación se ha obtenido mediante GPGAC, aplicando el segundo tratamiento del ruido al calcular la función objetivo. Al haber muy pocos elementos ruidosos, se alcanza el máximo valor para la medida que descarta el ruido sin ser penalizada. A medida que se descartan más elementos ruidosos, el valor obtenido mediante el primer método crece, ya que los clusters quedan más separados y compactos. Sin embargo, la segunda medida decrece, pues al haber más ruido, este tratamiento recibe una mayor penalización. En la gráfica de la derecha se puede ver un caso extremo. Esta clasificación fue obtenida por el algoritmo GPGAC donde la función objetivo es el índice de Calinski-Harabasz descartando los elementos ruidosos sin penalización. Debido a que GPGAC optimiza esta función, la clasificación que obtiene busca tener el mayor número de elementos

considerados como ruido, pues estos dan un mayor valor de la medida tomada como función objetivo. Esta situación es la menos deseable, por esa razón, en el algoritmo GPGAC se ha implementado de forma que usa el segundo tratamiento del ruido en la función objetivo.

A continuación se presentan gráficamente los resultados obtenidos sobre los datasets de dos dimensiones, según la medida de calidad que ha sido empleada como función objetivo:

- En la Figura 4.2 podemos ver la clasificación obtenida cuando la función objetivo es el índice de Calinski-Harabasz. En este caso, todas las clasificaciones son consistentes con lo esperado. El algoritmo identifica los clusters correctamente, aunque el grado de separación sea muy pequeño.
- En la Figura 4.3, la función objetivo corresponde al coeficiente de Silhouette. En los tres últimos datasets, encuentra una clasificación correcta, pero en el primero acaba en un mínimo local donde únicamente descarta dos nodos. Esto se debe a que cualquier cambio pequeño conllevaría descartar más ruido, obteniendo una penalización que impide la mejora.
- En la Figura 4.4 se emplea el índice de Davies-Bouldin como función objetivo. En este caso, el fenómeno de quedar atrapados en mínimos locales en tres de los cuatro datasets. Además, en estos tres casos existe un único punto que es descartado como ruido. Esta medida de calidad no es robusta frente al ruido, por lo que se ve muy penalizada por la aparición de este, no siendo aplicable para la obtención de los clusters deseados.

A la vista de estas gráficas, se concluye que el índice de Calinski-Harabasz es la función objetivo que da lugar a los mejores resultados, pues es la más resiliente al ruido. Estos resultados coinciden con los expuestos en trabajos previos [27], donde se concluye que esta métrica es la más robusta a outliers. Las otras dos métricas pueden servir de referencia para comparar clasificaciones, pero no es recomendable utilizarlas durante la obtención de la clasificación mediante el algoritmo, pues quedan atrapadas en mínimos locales. A partir de aquí, la función objetivo de GPGAC se considera siempre el índice de Calinski-Harabasz sin descartar el ruido.

En cuanto al parámetro de k_0, se ha utilizado el algoritmo tomando distintos valores de este para comprobar su dependencia sobre él. En la Tabla 4.2 se muestran los resultados obtenidos para los valores $k_0 \in \{5, 7, 10, 12, 15\}$ del parámetro. Como se puede observar, incluso para valores distantes, la solución tiene una calidad similar en la mayoría de los casos. Incluso, cuando la separación es suficientemente alta, se alcanza siempre la misma clasificación final. Las situaciones en las que los resultados de uno de los parámetros queda por debajo de los demás suelen deberse a que, en dicha ejecución, el algoritmo acabó atrapado en un mínimo local. Sin embargo, existen dos sutiles diferencias cuando se varía este parámetro. En primer lugar, cuanto mayor es k_0, mayor es el tiempo de ejecución del algoritmo, pues el número de aristas en el grafo de partida es proporcional a este parámetro. Como el rendimiento de las medidas de centralidad es mayor cuanto mayor es el número de aristas, cuando se toma un mayor valor de k_0, el algoritmo tarda algo más de tiempo. Por otro lado, se observa que, a medida que k_0 es más pequeño, el algoritmo es más propenso a quedar atrapado en óptimos locales, sobre todo cuando la dimensión crece. La razón reside en que, a mayor k_0, el grafo modeliza una mayor cantidad de información

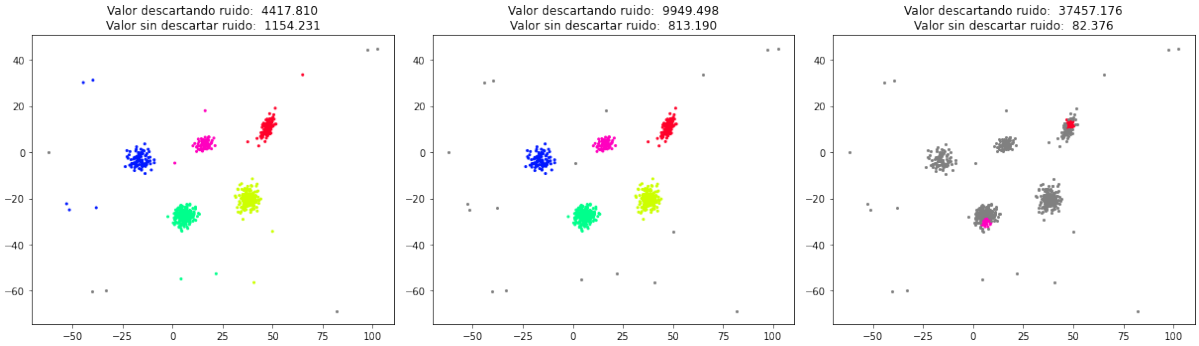


Figura 4.1: Cambios en la medida de calidad según el tratamiento del ruido. En la primera gráfica se representa la solución obtenida por GPGAC cuando la función objetivo considera el ruido como una clase diferente. En la segunda gráfica, se ha aplicado el movimiento "reducir α" para descartar los valores atípicos para mejorar la solución. En este caso, la medida de calidad calculada descartando el ruido mejora considerablemente, aunque, si se mantiene el ruido al calcularla, empeora. Por esta razón, el algoritmo prefiere la primera clasificación. La tercera gráfica se ha obtenido mediante GPGAC, donde la función descarta todo el ruido. En este caso, tiende a quedarse con los clusters más pequeños posibles, dando resultados no deseables.

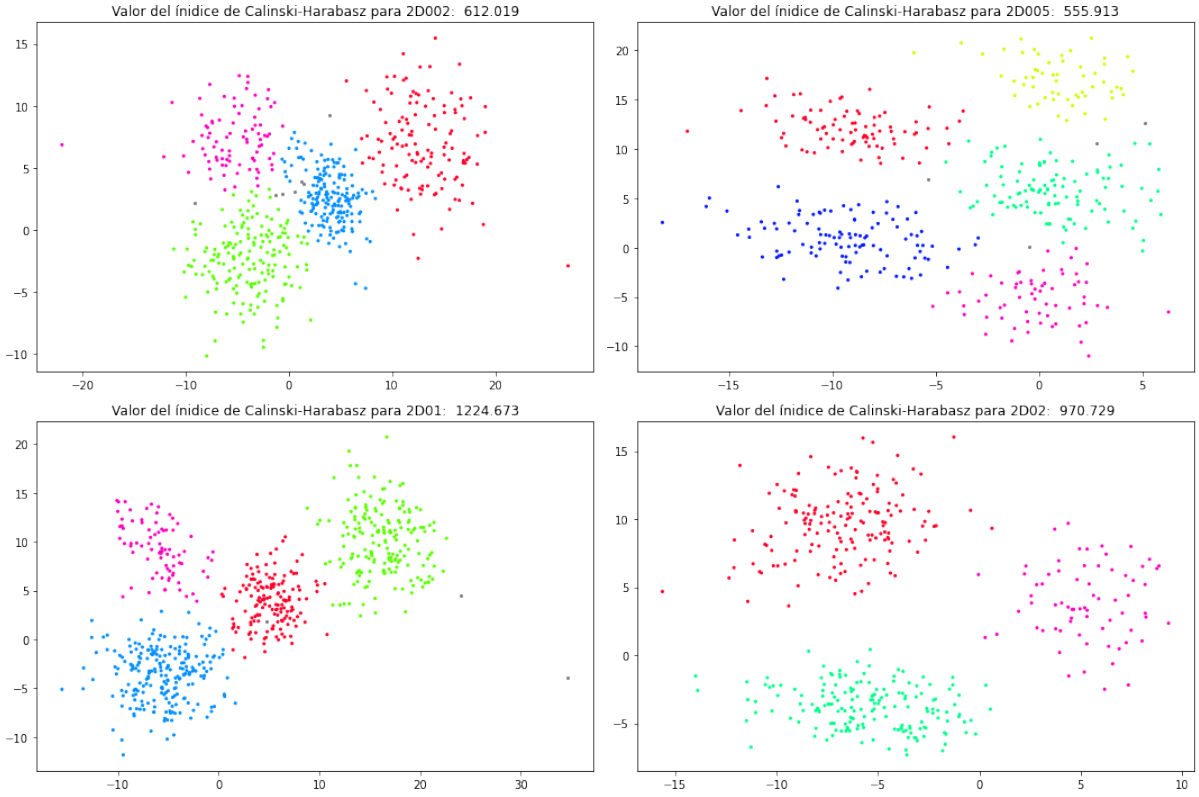


Figura 4.2: Resultados de GPGAC sobre los datasets bidimensionales usando el índice de Calinski-Harabasz como función objetivo. Esta es considerada como la clasificación final.

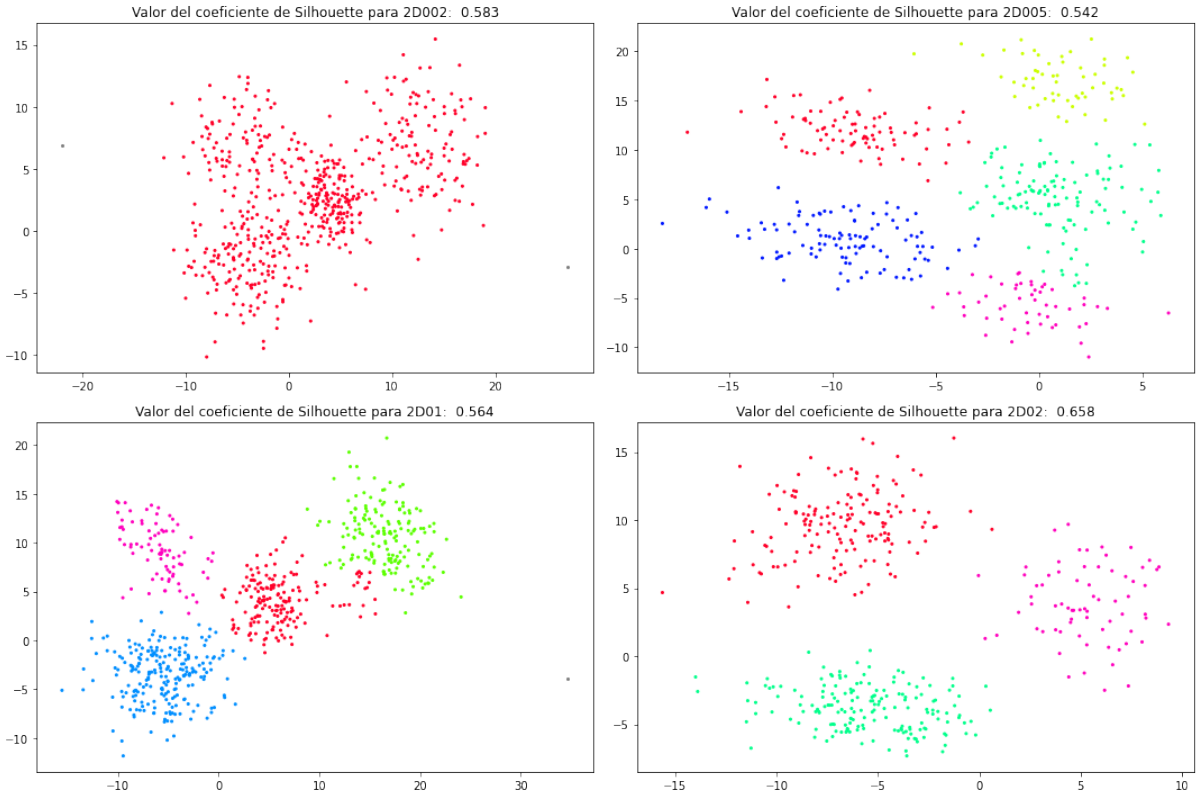


Figura 4.3: Resultados de GPGAC sobre los datasets bidimensionales usando el coeficiente de Silhouette como función objetivo.

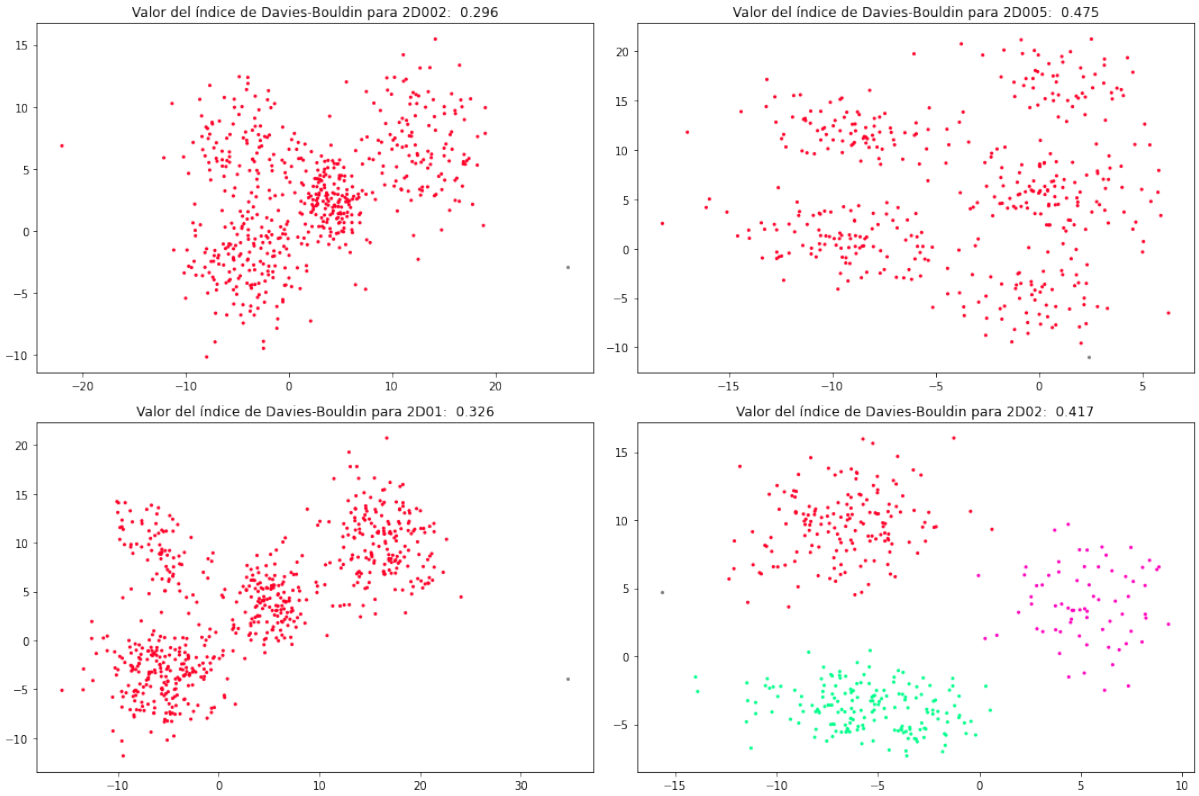


Figura 4.4: Resultados de GPGAC sobre los datasets bidimensionales usando el índice de Davies-Bouldin como función objetivo. El dataset con un mayor grado de separación es clasificado correctamente, mientras que los otros 3 quedan atrapados en mínimos locales.

y, por tanto, los movimientos pueden modificar el grafo de manera más informada, llegando a mejores soluciones. Por esta razón, se puede afirmar que el algoritmo es muy poco sensible a este parámetro. Esto se debe a que el propio algoritmo genético puede manipular el número de vecinos cercanos que cuenta cada cromosoma, por lo que k_0 actúa únicamente como parámetro inicial. Aunque manipular el parámetro k_0 en función del dataset puede dar lugar a un mayor rendimiento.

4.2. Comparación de GPGAC frente al estado del arte

A continuación, vamos a comparar el algoritmo con algunos de los algoritmos más representativos de las categorías definidas en el estado del arte. De entre los algoritmos jerárquicos, escogemos BIRCH debido a su eficiencia y a sus buenos resultados. En cuanto a los particionales, se escoge k-*means*++, debido a que es el estándar y el algoritmo más popular. HDBSCAN es el escogido entre los algoritmos basados en densidad, pues es la mejora de DBSCAN más exitosa y tiene una dependencia muy ligera de los parámetros. Finalmente, de entre los algoritmos basados en grafos escogemos a DBSGRAPH, el cual ha servido como inspiración para este trabajo.

Los parámetros seleccionados para cada algoritmo han sido aquellos que maximizan el índice de Calinski-Harabasz, considerando el ruido, al igual que en GPGAC. Sin embargo, durante la comparación, para no penalizar a aquellos que descartan elementos ruidosos, los resultados que se muestran para cada una de las medidas de calidad han sido calculados descartando el ruido. Así, la comparación es mas justa, pues algoritmos como HDBSCAN o GPGAC, que detectan el ruido, no se ven tan perjudicados frente a otros como k-*means*, el cual no sufre penalización pues no distingue el ruido. Estos datos están presentes en la Tabla 4.3. El valor óptimo de cada medida para cada dataset se marca en negrita, mientras que en azul se marca el segundo mejor resultado. En el Anexo A se muestran las gráficas asociadas a las mejores clasificaciones obtenidas por cada uno de los algoritmos para los datasets bidimensionales.

A la vista de los resultados, HDBSCAN queda generalmente por encima en términos generales, sobre todo en bajas dimensiones. Esto se explica debido a que HDBSCAN tiene falicidad para detectar elementos ruidosos y, al no estar penalizado, la medida de calidad tiende a mejorar en presencia de ruido. Cuando la dimensión es baja, GPGAC queda como el segundo mejor algoritmo, destacando incluso con un grado de separación muy pequeño. A medida que sube la dimensión, k-*means*++ y BIRCH se mantienen los más estables, ateniéndonos al índice de Calinski-Harabasz. En dimensiones mayores que dos, GPGAC se ve más perjudicado si el grado de separación disminuye. Sin embargo, cuando $D = 10$ y se tiene una dimensión ruidosa, GPGAC se mantiene en el tercer puesto de forma consistente según Calinsky-Harabasz, que es la métrica que busca optimizar, quedando muy cercano a k-*means*++ y BIRCH en la mayoría de los datasets.

En cuanto a DBSGRAPH, este algoritmo solo consigue destacar cuando el grado de separación en muy alto. Esto se debe a que, debido a cuenta con un *threshold* global para la distancia, α, no es capaz de diferenciar dos clusters donde la distancia entre los dos elementos de cada cluster es menor que α. En cambio, GPGAC puede diferenciar estos clusters correctamente, ya que estos elementos más cercanos estarán en zonas de menor densidad, quedando identificados como frontera y permitiendo

distinguir los clusters. Otro caso donde GPGAC mejora a DBSGRAPH aparece cuando se tienen dos clusters muy diferenciados unidos por un camino de baja densidad de puntos. Este caso se encuentra representado en la Figura 4.5. En este ejemplo, cualquier valor de α mayor que $\alpha = 0.02$ será incapaz de diferenciar los dos clusters circulares, pues todos los puntos del segmento intermedio estarán en la misma componente conexa. Para este valor, la mayor parte de los puntos de los clusters son descartados con ruido, dando lugar peores clasificaciones si reducimos este α. Por esta razón, DBSGRAPH es incapaz de distinguir bien este caso. Aplicando GPGAC a este conjunto, sí es posible diferenciar los tres clusters presentes, incluso a pesar de que el grado de separación es muy pequeño. Debido a la importancia de este parámetro α, DBSGRAPH solo es capaz de encontrar una fracción de todas las posibles clasificaciones, siendo necesario iterar el resultado para distintos valores de este y quedando prácticamente siempre en mínimos locales. Esto se debe a que únicamente emplea la información del grafo a la hora de determinar el grado de pertenencia de los puntos, pero no durante la clasificación. En cambio, GPGAC implementa técnicas que lo hacen más independiente del parámetro k_0, incluso permitiendo a cada cromosoma alterarlo, a la vez que puede modificar el grafo según estas nociones de centralidad en grafos. Gracias a ello, GPGAC resulta ser un algoritmo muy adaptable al conjunto de datos y flexible al parámetro k_0.

Por otro lado, otra ventaja de GPGAC es que, gracias a su estructura de grafo, puede encontrar clasificaciones que incluyan clusters no convexos, al igual que HDBSCAN y DBSGRAPH, mientras que k-*means*++ o BIRCH son incapaces. Sin embargo, como la métrica de Calinsky-Harabasz tiene en cuenta la distancia al centroide, en general va a preferir clusters más compactos y, por tanto, convexos. Si quisiésemos encontrar clusters de forma arbitraria mediante este algoritmo, sería conveniente usar una función objetivo diferente. En la Figura 4.6 se muestra una clasificación sobre el dataset "2DNoConvexo", donde se ve la capacidad de estos algoritmos para clasificar conjuntos no convexos. En el caso de GPGAC, este encuentra los conjuntos no convexos usando el indice de Rand ajustado, medida que necesita las etiquetas del dataset y por tanto es supervisado. Usando Calinski-Harabasz como función objetivo, GPGAC ha encontrado una solución que maximiza el índice, pero no se ajusta al conjunto, pues esta métrica prefiere clusters compactos. Por tanto, GPGAC puede encontrar clusters siempre y cuando tome una función objetivo válida para ella.

En cuanto a los demás algoritmos, tanto DBSGRAPH como HDBSCAN reconocen la forma no convexa del algoritmo. Sin embargo, si el grado de separación entre los clusters fuese más pequeño, puede que DBSGRAPH no hubiese sido capaz de clasificar correctamente. Por otro lado, BIRCH y k-*means*++ no distinguen este tipo de clasificaciones, siendo esta una ventaja para GPGAC.

Tras analizar los resultados, se concluye que GPGAC es capaz de competir con los algoritmos del estado del arte, sobre todo para bajas dimensionalidades. Además, cuenta con la ventaja de poder optimizar una métrica en particular, de forma que si se desea encontrar la mejor clasificación según una medida de calidad concreta, este algoritmo se adaptará a los resultados de esta, encontrando mejores clasificaciones. Esto le permite también ser capaz de encontrar clusters no convexos, dada la medida adecuada. Otro factor positivo a tener en cuenta es que, aunque cuenta con parámetros, la calidad de la solución encontrada a penas se ve afectada por ellos, no siendo necesario un afinamiento preciso. Este algoritmo se ha visto robusto frente al ruido, ya que consigue resultados superiores incluso a HDBSCAN cuando

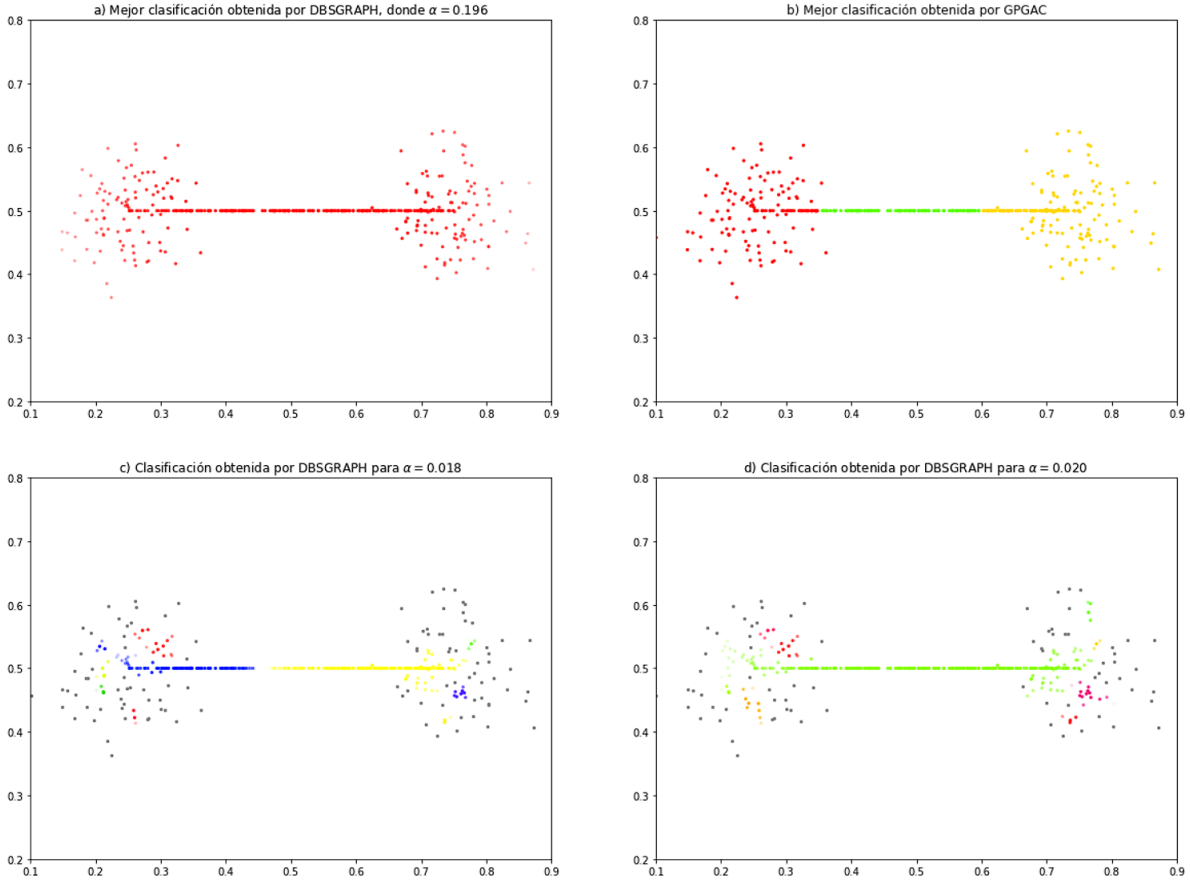


Figura 4.5: Diferentes diagramas obtenidos tras emplear DBSGRAPH y GPGAC sobre el mismo conjunto de datos. En a) se presenta la mejor clasificación obtenida tras iterar los valores de α. En b) se representa la solución obtenida por GPGAC, el algoritmo desarrollado en este trabajo. Finalmente, c) y d) representan la clasificación obtenida para $\alpha = 0.018$ y $\alpha = 0.02$ respectivamente. Se observa que cualquier valor de $\alpha \geq 0.02$ no será capaz de discernir los dos clusters, mientras que aún muchos puntos siguen siendo considerados como ruido.

se incluye una dimensión ruidosa, ateniéndonos al índice de Calinski-Harabasz. Sin embargo, no es del todo eficaz a la hora de determinar qué elementos son ruidosos. Esto se debe a que la penalización empleada en la función objetivo para el ruido prefiere encontrar clasificaciones con menos ruido o donde los elementos ruidosos estén más agrupados. Por esta razón, desarrollando una medida de calidad que verifique simultáneamente una independencia del ruido a la par que que asegura de tener un porcentaje de elementos descartados razonable, el algoritmo GPGAC utilizando esta métrica sería capaz de detectar mejor el ruido y, por tanto, de mejorar los resultados al igual que HDBSCAN cuando se ve beneficiado por descartar más elementos en los datasets con $D = 2$. Sin embargo, GPGAC puede tener problemas de escalabilidad cuando los datasets tienen un número muy grande de datos, pues tendrá por tanto más aristas y un mayor coste computacional.

Tabla 4.2: Resultados de GPGAC sobre todos los datasets definidos, cambiando el valor de k_0 inicial para observar la dependencia del algoritmo sobre este parámetro.

			$\sigma = 0.02$ $r = 0.5\,\%$	$\sigma = 0.05$ $r = 1\,\%$	$\sigma = 0.1$ $r = 2\,\%$	$\sigma = 0.2$ $r = 2\,\%$
$D = 2$	$k = 5$	Calinski-Harabasz	579.430	437.698	672.120	**970.729**
		Silhouette	0.488	0.343	0.314	**0.658**
		Davies-Bouldin	**0.678**	0.760	1.315	**0.458**
	$k = 7$	Calinski-Harabasz	586.597	521.605	1216.711	**970.729**
		Silhouette	**0.489**	0.408	**0.582**	**0.658**
		Davies-Bouldin	0.681	0.915	1.179	**0.458**
	$k = 10$	Calinski-Harabasz	586.597	521.598	1199.508	**970.729**
		Silhouette	**0.489**	0.402	0.495	**0.658**
		Davies-Bouldin	0.681	**0.558**	1.136	**0.458**
	$k = 12$	Calinski-Harabasz	**589.262**	566.218	1216.711	**970.729**
		Silhouette	0.488	**0.479**	**0.582**	**0.658**
		Davies-Bouldin	0.682	1.045	1.179	**0.458**
	$k = 15$	Calinski-Harabasz	559.045	**567.747**	**1224.673**	**970.729**
		Silhouette	0.472	0.428	0.579	**0.658**
		Davies-Bouldin	0.727	0.579	**0.998**	**0.458**
$D = 5$	$k = 5$	Calinski-Harabasz	101.716	86.043	268.504	**409.305**
		Silhouette	0.129	0.116	0.302	**0.485**
		Davies-Bouldin	2.350	2.483	2.152	**0.793**
	$k = 7$	Calinski-Harabasz	108.042	**204.460**	192.450	**409.305**
		Silhouette	0.150	0.254	0.099	**0.485**
		Davies-Bouldin	2.027	**1.418**	2.371	**0.793**
	$k = 10$	Calinski-Harabasz	176.501	134.335	357.335	**409.305**
		Silhouette	0.245	0.236	0.324	**0.485**
		Davies-Bouldin	**1.305**	4.052	**1.509**	**0.793**
	$k = 12$	Calinski-Harabasz	116.130	103.544	296.791	**409.305**
		Silhouette	0.213	0.193	0.287	**0.485**
		Davies-Bouldin	2.402	4.523	1.601	**0.793**
	$k = 15$	Calinski-Harabasz	**177.732**	170.706	**342.826**	277.025
		Silhouette	**0.283**	**0.257**	**0.326**	0.481
		Davies-Bouldin	1.820	3.163	1.593	1.394
$D = 10$	$k = 5$	Calinski-Harabasz	60.712	58.078	56.133	96.090
		Silhouette	0.176	0.157	0.066	0.340
		Davies-Bouldin	2.608	3.221	3.065	1.173
	$k = 7$	Calinski-Harabasz	69.216	64.928	75.719	**112.258**
		Silhouette	0.201	0.183	0.138	**0.366**
		Davies-Bouldin	2.353	2.658	2.709	**1.131**
	$k = 10$	Calinski-Harabasz	65.760	**72.411**	82.841	**112.258**
		Silhouette	0.192	**0.207**	0.162	**0.366**
		Davies-Bouldin	2.454	**2.518**	2.864	**1.131**
	$k = 12$	Calinski-Harabasz	61.088	46.414	**84.279**	**112.258**
		Silhouette	0.175	0.113	**0.164**	**0.366**
		Davies-Bouldin	2.608	3.656	2.815	**1.131**
	$k = 15$	Calinski-Harabasz	**70.931**	56.096	66.748	**112.258**
		Silhouette	**0.205**	0.105	0.067	**0.366**
		Davies-Bouldin	**2.198**	2.848	**2.636**	**1.131**

Tabla 4.3: Comparación con los algoritmos del estado del arte. Los valores de las medidas de calidad se han calculado ignorando el ruido. El valor óptimo de cada medida para cada datasets está marcado en negrita, mientras que en azul se marca el segundo mejor valor.

			$\sigma = 0.02$ $r = 0.5\,\%$	$\sigma = 0.05$ $r = 1\,\%$	$\sigma = 0.1$ $r = 2\,\%$	$\sigma = 0.2$ $r = 2\,\%$
$D = 2$	GPGAC	Calinski-Harabasz	**819.793**	711.336	1688.571	970.729
		Silhouette	0.495	0.573	0.587	0.658
		Davies-Bouldin	0.672	0.524	0.529	0.458
	DBSGRAPH	Calinski-Harabasz	78.599	399.830	1359.573	997.297
		Silhouette	-0.026	0.509	0.513	0.662
		Davies-Bouldin	0.803	0.691	0.593	0.454
	k-means++	Calinski-Harabasz	740.663	707.222	1622.904	971.764
		Silhouette	0.473	0.568	0.586	0.658
		Davies-Bouldin	0.833	0.532	0.533	0.462
	BIRCH	Calinski-Harabasz	800.267	674.444	1415.276	970.729
		Silhouette	0.489	0.557	0.548	0.658
		Davies-Bouldin	0.667	0.539	0.602	0.458
	HDBSCAN	Calinski-Harabasz	737.217	**807.920**	**2059.215**	**1010.743**
		Silhouette	**0.574**	**0.631**	**0.648**	**0.664**
		Davies-Bouldin	**0.586**	**0.465**	**0.457**	**0.454**
$D = 5$	GPGAC	Calinski-Harabasz	195.128	205.067	286.540	409.305
		Silhouette	0.321	0.229	0.176	0.485
		Davies-Bouldin	1.146	1.346	1.047	0.793
	DBSGRAPH	Calinski-Harabasz	61.874	81.740	140.438	247.675
		Silhouette	0.187	0.091	0.130	0.447
		Davies-Bouldin	1.055	1.513	1.283	0.979
	k-means++	Calinski-Harabasz	**251.134**	**322.962**	**490.265**	410.392
		Silhouette	0.363	0.384	0.433	0.486
		Davies-Bouldin	0.976	0.974	0.833	0.792
	BIRCH	Calinski-Harabasz	236.950	295.213	446.883	406.611
		Silhouette	0.349	0.366	0.414	0.483
		Davies-Bouldin	0.993	1.013	0.848	0.794
	HDBSCAN	Calinski-Harabasz	122.760	270.295	431.524	**451.971**
		Silhouette	**0.427**	**0.489**	**0.526**	**0.545**
		Davies-Bouldin	**0.948**	**0.695**	**0.661**	**0.673**
$D = 10$	GPGAC	Calinski-Harabasz	59.428	81.632	113.408	112.464
		Silhouette	0.171	**0.226**	0.227	0.366
		Davies-Bouldin	2.656	1.285	1.463	1.134
	DBSGRAPH	Calinski-Harabasz	47.403	24.252	47.013	107.380
		Silhouette	0.254	0.225	0.174	0.402
		Davies-Bouldin	1.500	**1.210**	1.288	0.957
	k-means++	Calinski-Harabasz	**98.071**	**108.654**	**139.112**	**112.527**
		Silhouette	0.186	0.224	0.195	0.366
		Davies-Bouldin	1.818	1.734	1.833	1.136
	BIRCH	Calinski-Harabasz	89.985	104.443	131.100	**112.527**
		Silhouette	0.172	0.218	0.185	0.366
		Davies-Bouldin	1.914	1.751	1.905	1.136
	HDBSCAN	Calinski-Harabasz	78.863	7.060	59.930	80.320
		Silhouette	**0.423**	0.159	**0.359**	**0.419**
		Davies-Bouldin	**0.897**	1.240	1.007	**0.874**

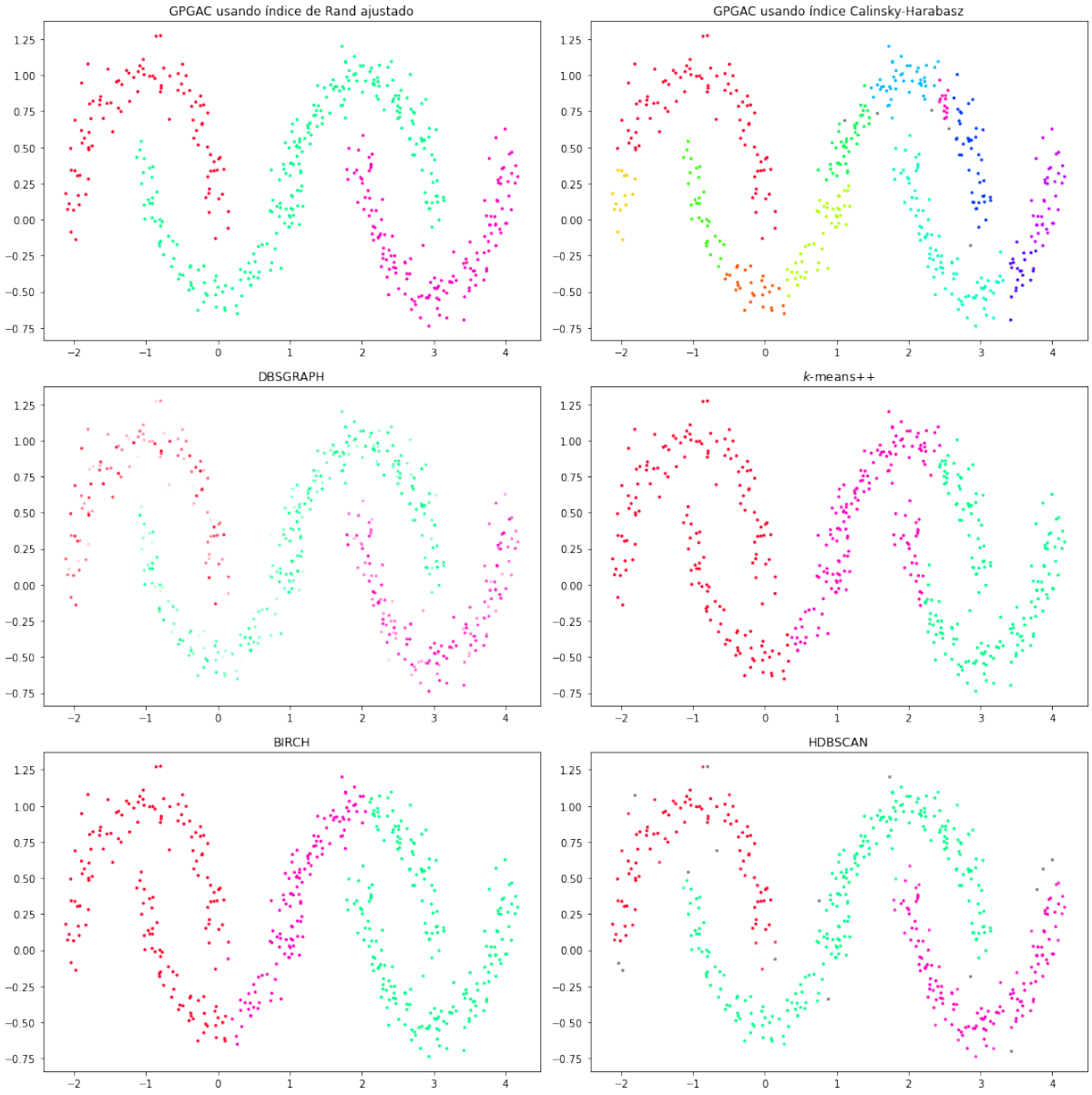


Figura 4.6: Clasificación mediante los algoritmos de 2DNoConvexo. Para GPGAC se usan dos métricas, el índice de Rand ajustado y el índice de Calinski-Harabasz, donde solo la primera se adapta bien a los datos.

Capítulo 5

Conclusiones

En este trabajo se ha desarrollado el algoritmo GPGAC *(Graph Prunning based Genetic Algorithm for Clustering)*, una propuesta innovadora que combina las técnicas de los algoritmos genéticos con las nociones de densidad de la teoría de grafos. En él, se parte de un grafo inicial generado a partir de los k_0 vecinos más cercanos, para posteriormente ir modificándolo mediante diferentes mecanismos, eliminando aristas hasta obtener una mejor clasificación. Aquí, el algoritmo genético toma el papel de seleccionar la mejor secuencia de movimientos a aplicar, donde cada cromosoma tiene una longitud variable y codifica una cadena de modificaciones que toman en cuenta las medidas de centralidad en el grafo.

El algoritmo GPGAC surgió a partir de la inspiración en DBSGRAPH, con el objetivo de aprovechar en mayor medida la información del grafo a la hora de obtener una mejor clasificación. A la vista de los resultados, se comprueba que GPGAC supera en gran medida las capacidades de DBSGRAPH y es capaz de medirse con algunos de los algoritmos más populares como BIRCH, k-*means*++ o HDBSCAN al clasificar conjuntos de datos con distintas dimensiones y diferentes grados de separación. Además, GPGAC soluciona uno de los problemas más frecuentes al emplear algoritmos basados en densidad como DBSCAN, que es la fuerte dependencia frente los parámetros. Esto se consigue permitiendo al algoritmo genético manipular el parámetro k_0 durante la ejecución.

Además, GPGAC puede emplear diferentes métricas de calidad como función objetivo, pudiendo ser empleado para optimizar la métrica que mejor identifique la calidad de la clasificación sobre el conjunto de datos según el criterio del usuario. Esto le da una mayor flexibilidad, pues seleccionando la métrica que más se adecue a nuestras necesidades, podemos encontrar la clasificación buscada, por ejemplo clasificando correctamente conjuntos de datos con clusters no convexos o con un grado de separación muy pequeño.

Se puede concluir que, gracias a la investigación llevada a cabo durante la realización del trabajo, se han alcanzado los objetivos principales marcados al principio de este. Cabe destacar que, aunque este nuevo algoritmo surge de la inspiración sobre las diferentes técnicas empleadas en el estado del arte, este aporta un nuevo enfoque innovador. Por esta razón, este trabajo también contribuye al desarrollo de mejores algoritmos de clustering, dando lugar a nuevas posibles líneas de investigación.

Futuras líneas de investigación

Las ideas que han sido aportadas en este trabajo dan lugar a un amplio abanico de posibilidades de cara a aprovechar los conceptos aquí implementados para fabricar nuevos algoritmos a partir de ellos.

Como resumen general, este algoritmo emplea algoritmos genéticos para buscar una secuencia de funciones que, al ser aplicadas sobre un grafo, generan una clasificación adecuada, tratando de maximizar una función objetivo. La elección de los algoritmos genéticos como la metaheurística que guia la búsqueda se debe a su adaptabilidad para trabajar con soluciones de diferente tamaño, así como a su facilidad a la hora de explorar espacios que mezclan variables discretas y continuos, como son el tipo de movimientos y los umbrales de estos, respectivamente. Sin embargo, futuros trabajos pueden tomar este algoritmo como esquema inicial y emplear una metaheurística diferente para encontrar la mejor secuencia, por ejemplo usando IDPSO.

Del mismo modo, innovaciones en la función objetivo pueden dar lugar a mejores clasificaciones. Como ya se ha visto antes, algunas de las medidas de calidad internas del clustering tienen determinadas carencias a la hora de tratar con valores atípicos o ruidosos. Por esta razón, el desarrollo de una nueva medida de calidad para tratar estas situaciones puede complementar en gran medida el funcionamiento de este algoritmo. Será necesario encontrar una medida que sea independiente del ruido, mientras que a la vez asegure que este no se dispara. Otras posibilidades podrían consistir en el desarrollo de medidas que favorezcan clusters continuos sin depender de la distancia al centroide, facilitando que el algoritmo encuentre clusters no convexos. Finalmente, un análisis multiobjetivo del algoritmo puede dar lugar a clasificaciones que se ajusten correctamente a todas las variables deseadas.

De forma análoga, se podría estudiar el efecto que puede tener en el algoritmo la aplicación de distintas medidas de similaridad, observando si es posible mejorarlo con medidas que incorporen un peso específico a cada variable, mejorando la clasificación cuando se tienen variables ruidosas. En este aspecto, sería interesante emplear una medida de similaridad que lo haga apto para trabajar sobre espacios donde las variables son cualitativas o discretas.

Por otro lado, este algoritmo se basa en descartar nodos mediante unos movimientos que tratan de estimar la densidad dentro del grafo. Una mayor investigación en torno a las medidas de centralidad dentro del grafo puede dar lugar a generar "movimientos" que permitan una mayor adaptabilidad del algoritmo. Una forma de aprovechar mejor los movimientos consistiría en que estas medidas fuesen tomadas de forma independiente para cada cluster, permitiendo obtener una clasificación donde cada una de las clases pueda tener una densidad diferente.

En cuanto a la inicialización del algoritmo, este parte de los k_0 vecinos más cercanos, pero el efecto de diferentes inicializaciones puede ser interesante de cara a mejorar la eficiencia. Por ejemplo, un esquema inspirado en la triangulación de Delaunay, inspirado en el algoritmo AMOEBA, puede resultar útil para reducir el número de aristas del grafo, aumentando su rendimiento.

Finalmente, GPGAC puede no ser muy eficiente para conjuntos de datos con un gran número de nodos, pues su rendimiento depende del número de aristas en el grafo. Por ello, se abren posibles líneas de investigación que tomen inspiración de algoritmos

como CURE, BIRCH o CLARANS, donde se toma un conjunto de representantes para clasificarlos, clasificando finalmente el resto del conjunto de datos a partir de los representantes. Una forma de llevarlo a cabo podría consistir en determinar la mejor secuencia de acciones para un grafo formado únicamente por los representantes para posteriormente aplicar esa secuencia a un grafo que cuente con todos los elementos del conjunto de datos, aumentando la escalabilidad del algoritmo.

Bibliografía

[1] S. Agarwal and S. Kumar. Survey on clustering problems using metaheuristic algorithms. *2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)*, pages 1–5, 2024.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Record*, 27:94–105, 1998.

[3] I. Ahmad, M. Akhtar, S. Noor, and A. Shahnaz. Missing link prediction using common neighbor and centrality based parameterized algorithm. *Scientific Reports*, 10:364, 2020.

[4] S. Alam, G. Dobbie, and P. Riddle. An evolutionary particle swarm optimization algorithm for data clustering. *2008 IEEE Swarm Intelligence Symposium*, 2008.

[5] M. Ankerst, M. Breunig, P. Kröger, and J. Sander. OPTICS: Ordering points to identify the clustering structure. *Sigmod Record*, 28:49–60, 1999.

[6] D. Arthur and S. Vassilvitskii. K-Means++: The advantages of careful seeding. *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms*, 8:1027–1035, 2007.

[7] S. Bandyopadhyay and U. Maulik. Nonparametric genetic clustering: comparison of validity indices. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31:120–125, 2001.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[9] P. Berkhin, J. Beche, and D. Randall Randall. Interactive path analysis of web site traffic. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 414–419. Association for Computing Machinery, 2001.

[10] J. C. Bezdek, R. Ehrlich, and W. Full. FCM: The fuzzy c-means clustering algorithm. *Computers Geosciences*, 10:191–203, 1984.

[11] P. Bonacich. Technique for analyzing overlapping memberships. *Sociological Methodology*, 4:176–185, 1972.

[12] S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. *Proceedings of the 2008 SIAM International Conference on Data Mining (SDM)*, pages 243–254, 2008.

[13] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30:136–145, 2008.

[14] T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods*, 3:1–27, 1974.

[15] R. Campello, D. Moulavi, and J. Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Advances in Knowledge Discovery and Data Mining*, pages 160–172. Springer Berlin Heidelberg, 2013.

[16] D. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 224 – 227, 1979.

[17] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20:364–366, 1977.

[18] B. Delaunay. Sur la sphère vide. *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na*, 1934(6):793–800, 1934.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39:1–22, 2018.

[20] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3:32–57, 1973.

[21] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[22] V. Estivill-Castro and I. Lee. AMOEBA: Hierarchical clustering based on spatial proximity using delaunaty diagram. 2000.

[23] A. Ezugwu, A. Ikotun, O. Oyelade, L. Abualigah, J. Agushaka, C. I. Eke, and A. Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110, 2022.

[24] L. Feng, M. Qiu, Y. Wang, Q. Xiang, Y. Yang, and K. Liu. A fast divisive clustering algorithm using an improved discrete particle swarm optimizer. *Pattern Recognition Letters*, 31:1216–1225, 2010.

[25] E. W. Forgy. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[26] I. Gialampoukidis, S. Vrochidis, and I. Kompatsiaris. A hybrid framework for news clustering based on the DBSCAN-martingale and LDA. 9729:170–184, 2016.

[27] L. Guerra, V. Robles, C. Bielza, and P. Larrañaga. A comparison of clustering quality indices using outliers and noise. *Intelligent Data Analysis*, 16:703–715, 2012.

[28] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. *SIGMOD Record*, 27:73–84, 1998.

[29] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25:345–366, 2000.

[30] S.o Gutiérrez. Paralelización y optimización de DBSGRAPH. No Publicado, 2019.

[31] S. Harifi, M. Khalilian, and J. Mohammadzadeh. Swarm based automatic clustering using nature inspired emperor penguins colony algorithm. *Evolving Systems*, 14, 2023.

[32] S. Harifi, M. Khalilian, J. Mohammadzadeh, and S. Ebrahimnejad. Emperor penguins colony: a new metaheuristic algorithm for optimization. *Evolutionary Intelligence*, 12, 2019.

[33] E. Hruschka and N. Ebecken. A genetic algorithm for cluster analysis. *Intelligent Data Analysis*, 7:15–25, 2003.

[34] P. Jana and A. Naik. An efficient minimum spanning tree based clustering algorithm. *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*, pages 1–5, 2009.

[35] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[36] G. Karypis, E. Han, and V. Kumar. Chameleon: A hierarchical clustering algorithm using dynamic modeling. *Computer*, 32:68 – 75, 1999.

[37] L. Kaufman and P. Rousseeuw. Finding groups in data: An introduction to cluster analysis. *Wiley, New York. ISBN 0-471-87876-6.*, 1990.

[38] L. Kaufmann and P. Rousseeuw. Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 1987.

[39] L. Kaufmann and P. Rousseeuw. Finding groups in data: An introduction to cluster analysis. *Wiley, New York*, 1990.

[40] J. Kennedy and R. Eberhart. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4:1942–1948 vol.4, 1995.

[41] P. Kuila and P. Jana. A novel differential evolution based clustering algorithm for wireless sensor networks. *Applied Soft Computing*, 25:414–425, 2014.

[42] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, page 556–559, 2003.

[43] Q. Liu. An improved discrete particle swarm optimization algorithm. *Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012*, pages 883–890, 2013.

[44] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, University of California, 1, 281-297, 1967.

[45] C. Malzer and M Baum. A hybrid approach to hierarchical density-based cluster selection. In *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, 2020.

[46] Leland McInnes, John Healy, and Steve Astels. HDBSCAB: Hierarchical density based clustering. *The Journal of Open Source Software*, 2:205, 2017.

[47] G. Milligan. An algorithm for generating artificial test clusters. *Psychometrika*, 50:123–127, 1985.

[48] F. Murtagh. A Survey of Recent Advances in Hierarchical Clustering Algorithms. *The Computer Journal*, 26:354–359, November 1983.

[49] I. Naim, S. Datta, J. Rebhahn, J. Cavenaugh, T. Mosmann, and G. Sharma. SWIFT — scalable clustering for automated identification of rare cell populations in large, high-dimensional flow cytometry datasets, part 1: Algorithm design. *Cytometry Part A*, 85, 2014.

[50] J. Palacio-Niño and F. Berzal. Evaluation metrics for unsupervised learning algorithms, 2019.

[51] S. Pandit and S. Gupta. A comparative study on distance measuring approaches for clustering. *International Journal of Research in Computer Science*, 2:29, 2011.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] W. Qiu and H. Joe. Generation of random clusters with specified degree of separation. *Journal of Classification*, 23:315–334, 2006.

[54] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[55] S. Salhi. Handbook of metaheuristics (2nd edition). *Journal of the Operational Research Society*, 65, 2014.

[56] J. Saramäki, M. Kivelä, J. Onnela, K. Kaski, and J. Kertész. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75, 2007.

[57] R. Sharan and R. Shamir. CLICK: A clustering algorithm with applications to gene expression analysis. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*, 8:307–16, 2000.

[58] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16:30–34, 1973.

[59] A. Sohail. Genetic algorithms in the fields of artificial intelligence and data sciences. *Annals of Data Science*, 10, 2021.

[60] J. Swarndeep and S. Pandya. An overview of partitioning algorithms in clustering techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 5:1943–1946, 2016.

[61] Raymond T. and J. Han. Efficient and effective clustering methods for spatial data mining. In *Very Large Data Bases Conference (VLDB '94)*, 1994.

[62] E. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing Management*, 22:465–476, 1986.

[63] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, pages 97 – 102, 1908.

[64] W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, page 186–195, 1997.

[65] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[66] J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *2012 IEEE 12th International Conference on Data Mining*, pages 705–714, 2012.

[67] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. *SIGMOD Record*, 25:103–114, 1996.

[68] W. Zhang, X. Wang, D. Zhao, and X. Tang. Graph degree linkage: Agglomerative clustering on a directed graph. In *European Conference on Computer Vision*, 2012.

[69] C. Zhong, D. Miao, R. Wang, and X. Zhou. DIVFRP: An automatic divisive hierarchical clustering method based on the furthest reference points. *Pattern Recognition Letters*, 29:2067–2077, 2008.

Apéndice A

Comparación de GPGAC con el estado del arte en dos dimensiones

En este anexo, se muestran las gráficas asociadas a la clasificación de los datasets bidimensionales para cada uno de los algoritmos usados durante la comparación. Estos son GPGAC en la Figura A.1, DBSGRAPH en la Figura A.2, *k-means++* en la Figura A.3, BIRCH en la Figura A.4 y HDBSCAN en la Figura A.5. Para una comparación más equilibrada, los valores de las métricas que se muestran en las imágenes de esta sección son calculados ignorando el ruido.

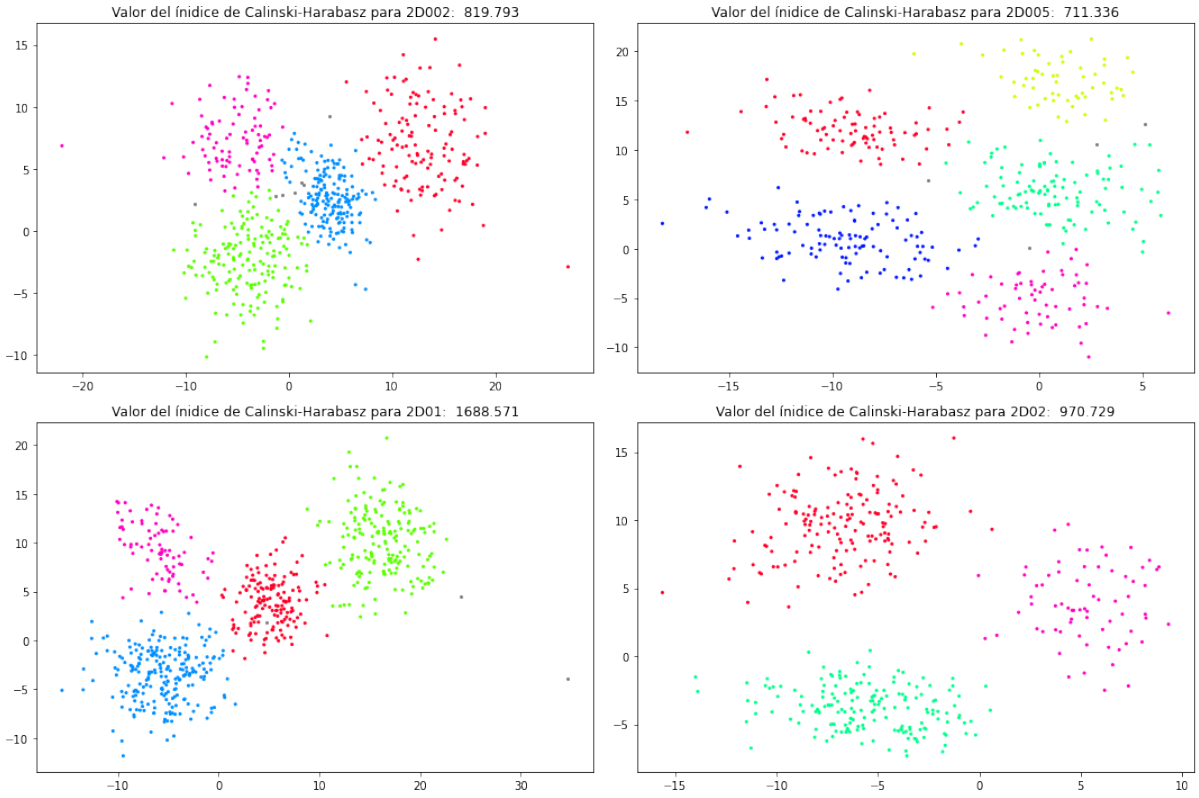


Figura A.1: Resultados de GPGAC sobre los datasets bidimensionales. Para el cálculo de las métricas, se ha ignorado el ruido.

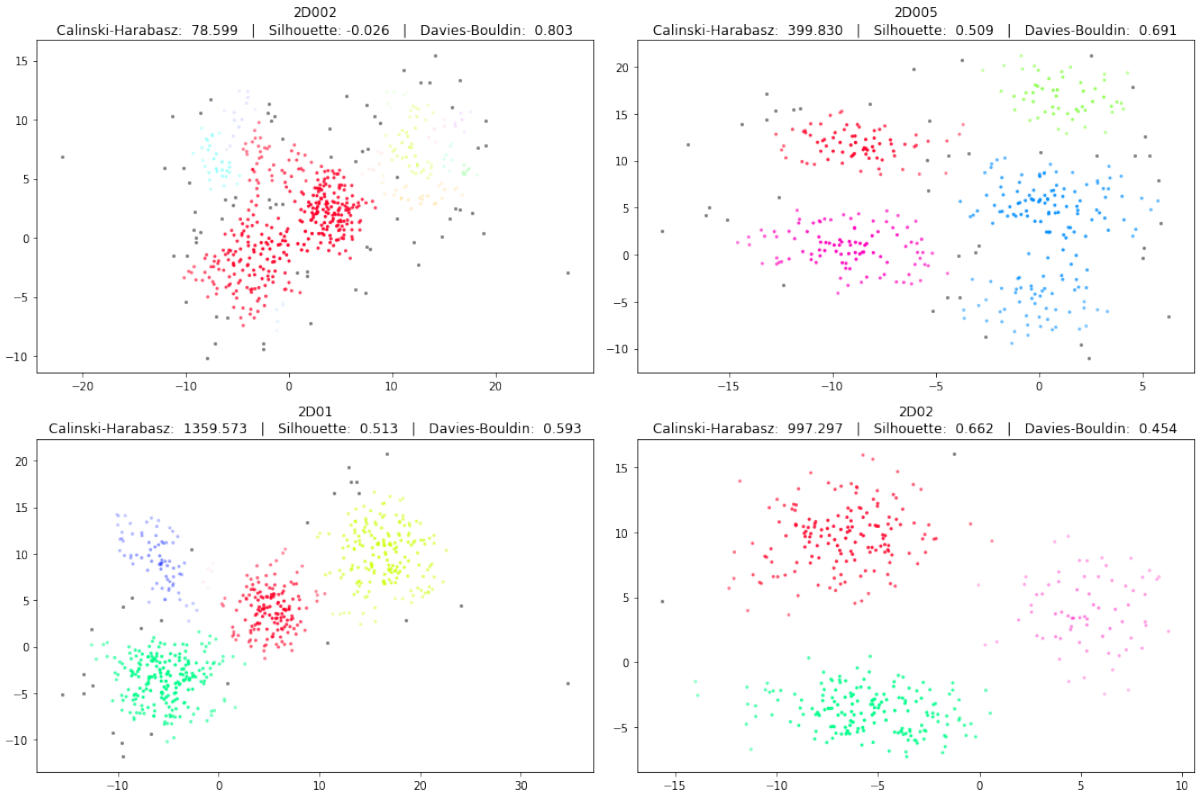


Figura A.2: Resultados de DBSGRAPH sobre los datasets bidimensionales. Para el cálculo de las métricas, se ha ignorado el ruido. El nivel de transparencia corresponde con el valor de pertenencia al cluster.

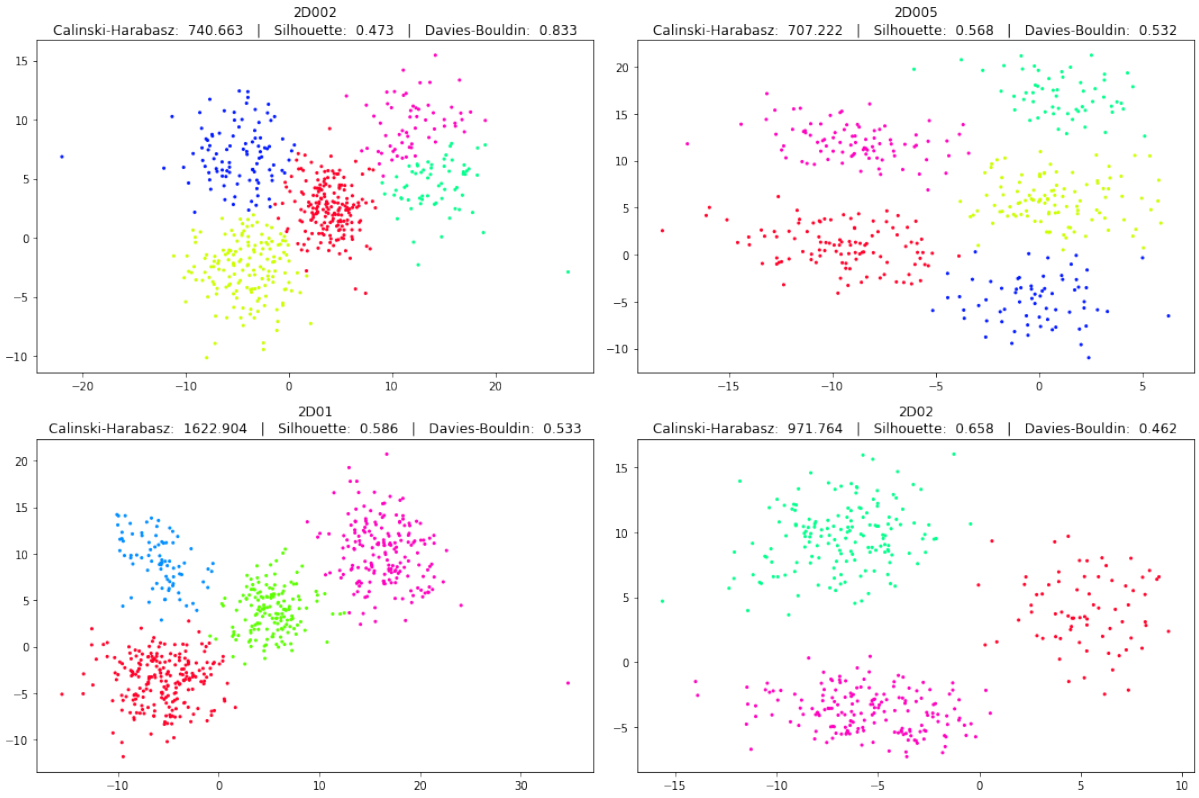


Figura A.3: Resultados de k-*means* sobre los datasets bidimensionales. Para el cálculo de las métricas, se ha ignorado el ruido.

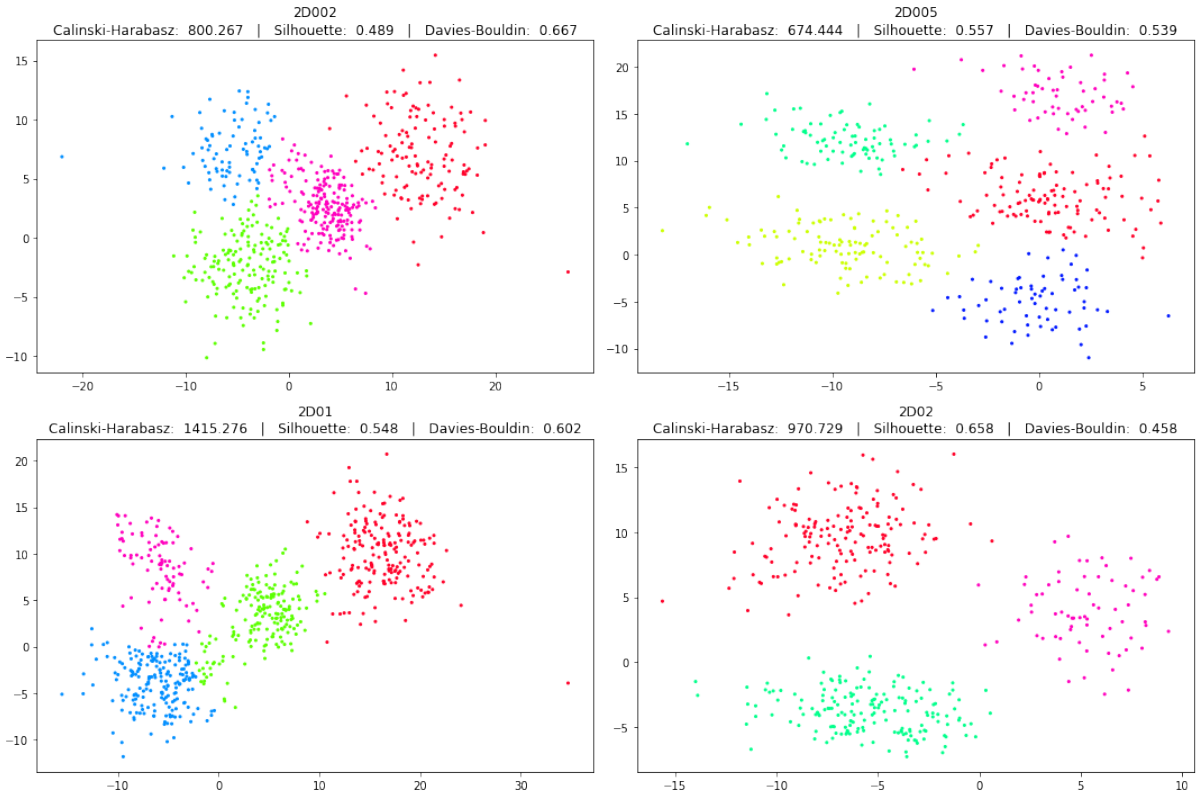


Figura A.4: Resultados de BIRCH sobre los datasets bidimensionales. Para el cálculo de las métricas, se ha ignorado el ruido.

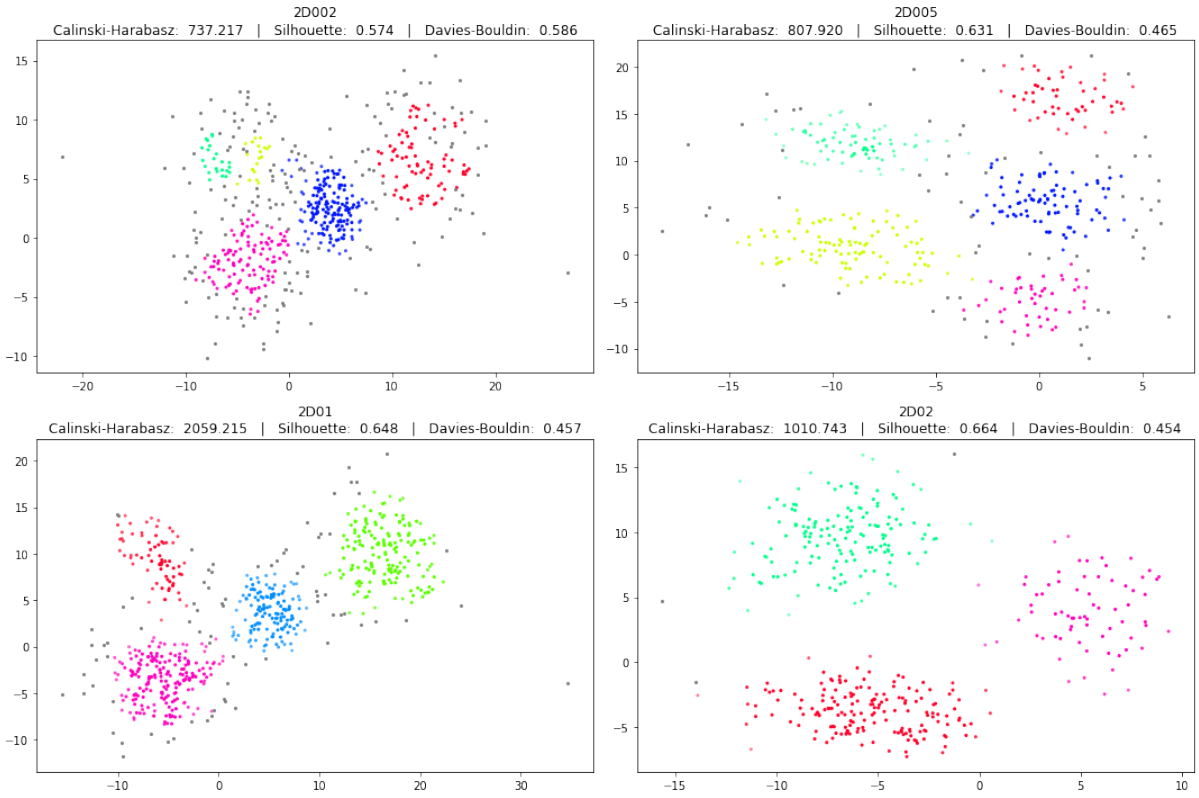


Figura A.5: Resultados de HDBSCAN sobre los datasets bidimensionales. Para el cálculo de las métricas, se ha ignorado el ruido. El nivel de transparencia corresponde con el valor de pertenencia al cluster.