Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana / Trabajo Fin de Grado
Fisikako Gradua / Grado en Física

Algoritmos de Clusterización e Identificación de Clústeres de Galaxias

Egilea/Autor:
Mikel Garcia de Andoin Bolaño
Zuzendaria/Director:
Gotzon Madariaga Menéndez

Leioa, 2020ko ekainaren 11a / Leioa, 11 de junio de 2020

Abstract

In a world overwhelmed with new data that is generated constantly we need efficient ways to obtain valuable information from it. In this aspect, clustering algorithms prove to be a useful tool to face this problem. This work reviews briefly some of the most important clustering algorithms that can be used on databases based on points. In this aspect, this work faces the problem of galaxy clustering, in which the available databases have of the order of 10^6 data points. This work discusses the best algorithm to face this problem. Implementing the DBSCAN algorithm, and using the SDSS DR16 galaxy database, I identify successfully at least 33% of the clusters previously identified in the SPIDERS cluster database. The implementation of DBSCAN made for this work uses variable clustering parameters, which could be used in other problems that can't be solved with constant parameters.

Contenidos

1. Introducción y fundamentos

Entender la forma del universo y su estructura a gran escala es uno de los objetivos de la cosmología. En otras ramas de la física se pueden preparar experimentos para medir un suceso de distintas, y de esta manera modelizar distintos sistemas. Sin embargo, la cosmología tiene un problema en este sentido; existe un único universo observable que solo se puede observar de una manera, desde la tierra en el presente. Por suerte, según nos alejamos más a la hora de observar objetos astronómicos, nos acercamos más al origen del universo.

Los datos extraídos a partir de observar el universo sirven para calcular una gran cantidad de datos. Viendo la escala del universo uno puede pensar que estas mediciones harán referencia a propiedades de gran escala, como pudieran ser propiedades de los agujeros negros, ondas gravitacionales o explosiones de supernovas. Pero por paradójico que parezca, se puede extraer información sobre las fluctuaciones cuánticas del universo temprano, o incluso puede servir para medir la masa de los neutrinos, con una precisión impensable en ningún otro laboratorio en la fecha de su publicación [1].

La magnitud del universo es ciertamente grande. Se estima que el universo observable contiene del orden de 10^{12} galaxias, repartidas a lo largo de un volumen del orden de 10^{13} Mpc^3 [2]. Este dato se obtiene al resolver las ecuaciones de Friedmann usando los parámetros del modelo del universo que mejor se ajusta a los datos observados, el modelo "Lambda-Cold Dark Matter" (ΛCDM). Esto hace que sea imposible un análisis manual de los datos observacionales que se puedan obtener. Esto es un ejemplo perfecto de problema relacionado con el *big data*. El tamaño de los datos disponibles, y la necesidad de analizarlos en conjunto, hace indispensable encontrar un método eficiente de extraer información.

En concreto en este trabajo se estudiará los clústeres de galaxias. Un clúster es una agrupación de datos suficientemente relacionados según una definición dada. Aplicados a las galaxias, los clústeres son agrupaciones de galaxias suficientemente compactas como para que las fuerzas entre ellas impidan que ninguna salga fuera del clúster. El estudio de los clústeres permite extraer información útil acerca de la estructura del universo y acerca de la formación de nuevas galaxias, entre otros. Dejando de lado este tipo de análisis posterior, este trabajo se centrará solamente en la identificación de clústeres de galaxias a partir de su posición en el universo. El objetivo será, a partir de una base de datos de gran tamaño, generar un mapa de todos los posibles clústeres, dados unos parámetros de clusterización variables obtenidos a partir de un cálculo previo.

En este trabajo se definen primero algunos conceptos recurrentes a lo largo del trabajo (Sec.1.1), y se dan unas nociones básicas de qué es un clúster (Sec.1.2). A continuación, se presentan algunos de los algoritmos de clusterización de más amplio uso (Sec.2). Pensando en el desarrollo de un algoritmo para obtener clústeres de galaxias, se eligen tres de los algoritmos presentados para implementarlos en el lenguaje de programación Matlab (Sec.3). Tras realizar una prueba de rendimiento de estos algoritmos, se elige el que mejor se ajusta a las características buscadas. Posteriormente, se presenta el problema de los clústeres de galaxias. Usando la base de datos del Sloan Digital Sky Survey (SDSS), se obtiene un catálogo de posibles clústeres (Sec.4). Para comprobar la validez de los resultados, se comparan los resultados obtenidos con varios clústeres previamente identificados. Al final de este trabajo se comentan los resultados obtenidos y las posibles mejoras que se pudieran aplicar al algoritmo desarrollado en un futuro (Cap.5).

1.1 Algunas definiciones

En este trabajo se utiliza notación y terminología de diversos ámbitos. Se muestran a continuación algunas definiciones necesarias para poder seguir el trabajo:

Distancia y métrica:

A la hora de trabajar con clústeres, la información está codificada en puntos repartidos en un cierto espacio. Una de las maneras que se tiene de encontrar una relación entre ellos es a partir de la distancia que los separa.

La distancia entre dos puntos en un espacio se define a través de su métrica. La métrica es una función que da para dos puntos del espacio $\{x, y\}$ su distancia $d(x, y)$, y que tiene que cumplir las siguientes propiedades:

1. $d(x, y) \geq 0$ (Axioma de separación)
2. $d(x, y) = 0 \Leftrightarrow x = y$ (Axioma de coincidencia)
3. $d(x, y) = d(y, x)$ (Simetría)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (Desigualdad triangular)

El ejemplo más común de métrica es la del espacio euclídeo, en la que la distancia entre dos puntos en un espacio de dimensión n viene dada por

$$d_{\text{eucl}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{1}$$

En este trabajo se usará también la métrica Manhattan, que tiene en cuenta la suma de las distancias en cada eje, y está dada por

$$d_{\text{manh}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i| . \tag{2}$$

Grafos:

Un grafo $G(V, E)$ es un conjunto de nodos (o vértices) V y aristas E que relacionan un par de vértices. Se dice que dos nodos son adyacentes o vecinos si están unidos a través de una arista. Un camino entre dos nodos x, y es una secuencia de vértices no repetidos adyacentes entre sí, empezando en x y terminando en y. Un ciclo es un camino no vacío que empieza y termina en el mismo nodo. Un grafo conexo es aquel en el que existe siempre un camino entre dos nodos cualesquiera. A partir de sus características, se pueden definir distintos tipos de grafos. En este trabajo se trabajan principalmente con dos: el grafo completo y el árbol.

Un grafo completo es un grafo en el que todos los nodos son vecinos entre sí. Esto implica que en un grafo completo de n nodos, el conjunto de las aristas contiene $n(n-1)/2$ elementos.

Un árbol es un grafo conexo acíclico (que no contiene ningún ciclo). Esto implica que el camino entre dos nodos es único y que el número de aristas tiene que ser uno menos que el número de nodos (Fig.1).

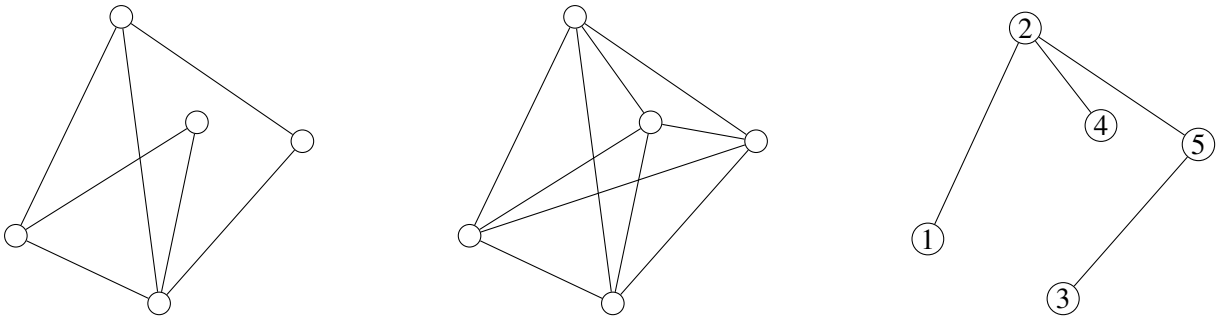

Fig. 1: Diagramas de un grafo (izquierda), un grafo completo (centro) y un árbol (derecha). Tomando como raíz del árbol el nodo 2, se tiene un árbol de dos niveles. El primer nivel lo forman los hijos de la raíz (nodo 2), que son los nodos 1, 4 y 5. El segundo nivel está formado solo por el nodo 3. Las hojas de este árbol serán los nodos 1, 4 y 3. Se verifica que, teniendo 5 nodos, un grafo completo tiene $5(5-1)/2 = 10$ aristas, y un árbol $5 - 1 = 4$ aristas.

En el ámbito de la computación, un árbol es una estructura de datos con la misma topología que la definida en matemáticas. El árbol se empieza a definir etiquetando uno de los nodos como la raíz. El árbol se organiza por niveles, que se definen como el número de aristas del camino que une un nodo a la raíz. Un hijo es un nodo de un nivel $n + 1$ conectado por una arista a un nodo padre del nivel n. Una hoja es un nodo sin hijos. Una de las propiedades más útiles para este trabajo es que la elección de la raíz no afecta a sus propiedades, ya que la topología de la estructura se mantiene invariante [3, pp. 13-17].

En este trabajo se han usado estructuras de árbol para representar los puntos de las bases de datos. Cada tipo de árbol usado se definirá a partir de la forma en la que se construye y la información contenida en cada arista o nodo.

Complejidad computacional:

En este trabajo se tratará en profundidad la complejidad computacional de los distintos algoritmos. La eficiencia de un algoritmo se mide acuerdo a dos variables en función del tamaño del problema: el número de pasos (tiempo) para llegar a la solución y la cantidad de memoria (espacio) utilizada. Un paso se puede definir como un conjunto de operaciones que se realizan de forma iterativa. La complejidad temporal de un algoritmo se calcula teniendo en cuenta el número de veces que se repite el paso más pequeño. De esta manera, para dar con la complejidad temporal de un algoritmo suele bastar con calcular el número máximo de veces que se repite una misma operación a lo largo de la ejecución.

No todos los algoritmos requieren el mismo número de pasos para resolver distintos problemas de igual tamaño. Por ello, a la hora de referirse a la complejidad computacional de un algoritmo se suelen definir tres tipos de notación. La notación O sirve para indicar una cota superior del coste del algoritmo. En algunos algoritmos en los que el coste depende de los parámetros de entrada, se suele identificar el coste para el peor y el mejor de los casos, con Θ y Ω respectivamente. El coste computacional se da como coste asintótico cuando el tamaño del problema es arbitrariamente grande. Siendo el coste computacional la suma de funciones elementales en función del tamaño del problema, solo se dará la que da un crecimiento más rápido. Si el coste del problema está multiplicado por un factor constante se suele obviar, salvo en los casos en los que esa constante contenga información relevante[1].

[1] Por ejemplo, en un problema en el que se necesiten $n + 10$ pasos, su complejidad será $O(n)$. En otro que necesite $3n^2 + n + 1$ pasos, la complejidad será $O(n^2)$.

1.2 Concepto de clúster

Un clúster es una agrupación de datos suficientemente relacionados. Esta vaga idea de lo que es un clúster viene del problema de que no existe una definición universal. Cada tipo de problema necesitará una definición adecuada para obtener unos resultados coherentes.

Una de las características que definen a los clústeres es su forma. Algunos algoritmos centran su búsqueda en clústeres elipsoidales. Estos clústeres tienen la peculiaridad que su centro de masas estará siempre en su interior. Esta definición impide que existan clústeres cóncavos o de media luna, y tampoco permite que un clúster rodee a otro en su interior. Por ello, una definición más amplia permite que los clústeres tomen cualquier forma. Un ejemplo gráfico de este problema puede ser la identificación de los brazos de una galaxia. Si se usara la definición de clúster elipsoidal, solo se obtendría un gran clúster que cubriría toda la galaxia. Al extender la definición a clústeres de forma arbitraria se podrían identificar correctamente sus brazos.

Otra característica importante es la forma en la que se define la relación entre los puntos. Aquí entra en juego la elección de la métrica que se use para definir las distancias entre los puntos. El tipo de métrica que se use dependerá del problema. En problemas de procesado de lenguaje natural, la distancia entre palabras puede venir dada por su posición en una frase, por las relaciones gramaticales, por el número de letras en común, etc [4]. Otros problemas pueden necesitar definir una métrica en varias dimensiones, como en el procesado de imagen, donde la métrica está definida (entre otras cosas) por el color y la distancia de los píxeles [5]. En este trabajo, la métrica viene dada por la métrica del espacio-tiempo del universo, la cual se desarrollará mas adelante.

Ya definida la forma y la métrica, queda por diferenciar un clúster de otro. Una manera de visualizar la forma en la que se pueden agrupar los puntos es a través de un grafo completo ponderado. Cada nodo representará un punto y las aristas con su peso representarán la distancia entre los puntos. Sobre este grafo completo, se deberán eliminar las aristas hasta que se tengan subgrafos conexos que representarán cada uno un clúster. Dejando de lado las definiciones concretas de cada algoritmo, existen dos formas comunes para definir los clústeres:

- Clúster tipo Friends-of-Friends (FoF): Es un tipo de clúster en el que todas las aristas tienen una longitud menor que una distancia máxima $w_{ij} < b$.
- Clúster tipo density-reachable: En este tipo de clústeres se usa el mismo criterio que en los de tipo FoF para eliminar las aristas sobrantes. Sin embargo, aquí se diferencian dos clases de puntos. Los puntos de core son puntos con un número mínimo de vecinos. Los puntos de frontera son puntos que no son puntos de core, pero que están conectados al menos a uno de ellos. Los clústeres lo podrán formar solo los puntos de core, o ambos tipos de puntos si se define una forma de decidir sobre puntos de frontera conectados a puntos de core de dos clústeres distintos [6].

Además de esto, también se puede tener en cuenta el tamaño o número de puntos de cada clúster para discriminar entre clústeres de verdad y los formados por el ruido. Sobre esto, en este trabajo se ha decidido usar un número mínimo de puntos necesarios para identificar una agrupación de puntos como clúster ($nMin$). De esta manera se busca evitar identificar una gran cantidad de clústeres en los que solo se tendrían 2 o 3 puntos conectados entre sí.

2. Algoritmos de clusterización

Frente a un catálogo de puntos desordenado y sin relación, los algoritmos de clusterización buscan relaciones de semejanza entre los distintos datos. A continuación se presentan algunos de los algoritmos más usados, divididos por el tipo de estrategia que usan. Es necesario advertir que existe una grandísima cantidad de algoritmos distintos para este problema. Debido a la gran cantidad de literatura disponible, esta selección contiene solamente los que he considerado más relevantes o los que mejor se pueden adaptar al problema de los clústeres de galaxias.

2.1 Algoritmos de cuantificación vectorial

Los algoritmos de cuantificación vectorial se basan en la búsqueda de elementos representativos en una base de datos para identificar los puntos a través de los vectores de un libro de códigos, que cuantifican las características de cada punto. De esta manera, se consigue reducir la cantidad de información necesaria para almacenar la información relevante extraíble de esta base de datos. Aunque los algoritmos particionales (Sec.2.2) entran también en esta categoría, se muestran a continuación algunos algoritmos que usan este principio de cuantificación vectorial [7, Sec. 1.2]:

2.1.1 Neural Gas Vector Quantisation (NG)

La idea de este algoritmo es dejar evolucionar un sistema de puntos como si fuera un gas. Para ello se genera un conjunto aleatorio de k puntos representativos del sistema, con k suficientemente menor que el tamaño del problema. Para cada iteración del algoritmo, el sistema evolucionará según un modelo simple, en el que todos los puntos representativos interaccionan con un elemento de la base de datos original según una función

$$\mathbf{v}(t+1,k,\mathbf{x}) = \mathbf{v}(t,k) + \varepsilon(t)e^{-k/\lambda(t)}\left(\mathbf{x} - \mathbf{v}(t,k)\right), \tag{3}$$

donde $\mathbf{v}(t,k)$ es la posición del k-ésimo punto representativo más cercano a $\mathbf{x}$ en la iteración t, $\varepsilon(t)$ es un parámetro de tamaño del paso y $\lambda(t)$ es un parámetro de vecindad. Estos dos últimos parámetros son decrecientes con el número de iteraciones, por lo que se asegura que las oscilaciones de los puntos representativos que pudiera haber alrededor del punto de equilibrio se reducen hacia el final del algoritmo, con lo que se asegura la convergencia. A continuación se desarrolla este algoritmo (Alg.1).

Algoritmo 1 Neural Gas Vector Quantisation

Require: Tamaño del paso $\varepsilon(t)$, parámetro de vecindad $\lambda(t)$

1: Generar k puntos representativos en el espacio de la base de datos inicial
2: t=1
3: **repeat**
4: Elegir un punto aleatoriamente de la base de datos inicial, $\mathbf{x}$
5: Ordenar los puntos representativos según la distancia a $\mathbf{x}$
6: Actualizar la posición de los puntos representativos según la ecuación (Eq.3)
7: t=t+1
8: **until** La actualización de la posición de los puntos representativos es despreciable

Este algoritmo es útil no solo para detectar un número dado de clústeres, sino que además puede usarse para reducir el tamaño de la base de datos inicial. Sobre este conjunto representativo de puntos se puede aplicar otro algoritmo para encontrar clústeres, con la ventaja que que ahora su aplicación será más rápida debido a la reducción del tamaño del problema.

2.1.2 Growing Neural Gas (GNG)

Un problema que presenta el algoritmo anterior es que está limitado a escoger un número de puntos representativos fijo, por lo que dependiendo del número de puntos escogidos se puede llegar a perder la información que queremos obtener acerca de los clústeres. Para evitar esto, GNG presenta un algoritmo de expansión del número de puntos representativos usando una estructura de grafo.

Partiendo de un par de puntos representativos, el algoritmo se deja evolucionar de la misma manera que en NG. La diferencia principal viene a la hora de aplicar el paso 6 de (Alg.1). Siendo $\mathbf{w}_{s1}$ el punto representativo más cercano al punto de la base de datos elegido en el paso 4 ($\mathbf{x}$), en GNG tan solo se actualiza la posición de $\mathbf{w}_{s1}$ y sus vecinos. Si el segundo punto más próximo a $\mathbf{x}$ ($\mathbf{w}_{s2}$) no es un vecino de $\mathbf{w}_{s1}$ se creará una arista entre ellos.

A las aristas que conectan los puntos representativos se les asocia un número entero, la edad de la arista. La edad aumenta de tal manera de que en cada iteración la edad de todas las aristas salientes de $\mathbf{w}_{s1}$ aumenta en una unidad. Cuando una arista llega a cierta edad se elimina, eliminando también todos los nodos que queden inconexos. Además de esto, en cada iteración se actualiza el parámetro de error de cuantización, añadiendo a $\mathbf{w}_{s1}$ un error igual a la distancia $|\mathbf{w}_{s1} - \mathbf{x}|^2$. Cuando se lanza el algoritmo, el error de cuantización de los puntos representativos iniciales es 0.

Cada cierto número de iteraciones se introduce un nuevo punto representativo. Para ello, se busca el punto representativo con el mayor error de cuantización ($\mathbf{w}_{e1}$). El nuevo punto ($\mathbf{w}_n$) se coloca entre $\mathbf{w}_{e1}$ y su vecino con mayor error ($\mathbf{w}_{e2}$), actualizando el grafo. El error de $\mathbf{w}_n$ inicializa con el valor actual del error de $\mathbf{w}_{e1}$, y los errores de $\mathbf{w}_{e1}$ y $\mathbf{w}_{e2}$ se multiplican por un factor $\alpha < 1$. El resultado de este paso es un grafo con una suma de errores de cuantización de todos sus nodos menor que en la iteración anterior. Una condición de salida de este algoritmo puede ser que esta suma de errores sea menor que cierto umbral.

2.1.3 Topology Representing Network (TRN) y Dynamic TRN

Partiendo de la idea de NG, se puede dejar evolucionar el gas de puntos representativos iterativamente. Pero además, se puede obtener información extra generando un grafo a partir de estos puntos. Las aristas de este grafo representarán el grado de conexión entre los nodos, lo que implica una alta densidad de puntos de la base de datos entre los puntos representativos que conectan.

El algoritmo TRN (Alg.2) se puede construir modificando ligeramente NG. Después de actualizar las posiciones de los puntos representativos, y usando una estrategia similar a la usada en GNG, se actualiza la edad de las aristas que parten del nodo más cercano al punto de la base de datos seleccionado en NG. La edad de la arista que conecta los dos nodos más cercanos ($\mathbf{w}_{s1}$ y $\mathbf{w}_{s2}$) al punto $\mathbf{x}$ se establece al mínimo posible, 1. Cuando la edad de una arista supera un límite dado, esta arista se elimina del grafo, eliminando también los nodos que

queden inconexos. Esta estrategia de formación del grafo se conoce como regla de competición de Hebb.

Algoritmo 2 Topology Representing Network

Require: Tamaño del paso $\varepsilon(t)$, parámetro de vecindad $\lambda(t)$, edad máxima de una arista T

1: Generar 2 puntos representativos en el espacio de la base de datos inicial
2: t=1
3: **repeat**
4: Aplicar los pasos 4-6 de NG (Alg.1).
5: Aumentar en 1 la edad de todas las aristas que parten de $\mathbf{w}_{s1}$
6: Asignar 1 a la edad de la aristas que conectan $\mathbf{w}_{s1}$ a sus vecinos
7: Eliminar las aristas con una edad > T
8: t=t+1
9: **until** Se dan las condiciones de convergencia

De la misma manera que se plantea GNG para dar obtener el mejor número de clústeres, se tiene una versión dinámica de TRN. En esta versión, para aplicar los pasos 5-8 del algoritmo de TRN (Alg.2) es necesario que el punto $\mathbf{w}_{s1}$ esté suficientemente cerca de $\mathbf{w}_t$. Si no se da esta condición, se genera un nuevo punto representativo sobre $\mathbf{w}_t$, conectándolo a $\mathbf{w}_{s1}$.

2.2 Algoritmos particionales

El objetivo de los algoritmos particionales es encontrar la mejor división del espacio sobre la que se encuentran los puntos de la base de datos en un número dado de clústeres. Para ello, se busca maximizar la distancia entre clústeres, a la vez que se intenta minimizar la distancia entre los nodos dentro del mismo. A continuación se presentan algunos de los algoritmos basados en esta idea:

2.2.1 k-means

K-means es probablemente el algoritmo de clusterización más conocido y usado [8]. La motivación de este algoritmo es minimizar la distancia total de los nodos al centro de su respectivo clúster. La función a minimizar se llama Intra Cluster Spread (ICS) y está definida como

$$ICS(\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_k) = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in \mathbf{C}_i} d(\mathbf{x}_j, \mathbf{C}_i), \tag{4}$$

donde la función $d(\mathbf{x}_j, \mathbf{C}_i)$ es la distancia entre el nodo $\mathbf{x}_j$ y el centro del clúster i ($\mathbf{C}_i$) según la métrica en la que se esté trabajando.

Para construir el algoritmo es necesario fijar previamente un número de clústeres. En la primera iteración se generarán los centros de los clústeres de forma aleatoria, mientras que para el resto de iteraciones se harán calculando el punto medio entre los puntos del clúster. El algoritmo, descrito en (Alg.3), tiene dos grandes inconvenientes: no es capaz de diferenciar un clúster "bueno" de los puntos que representan ruido, ni tampoco de ajustar el número de clústeres durante su ejecución. Sin embargo, la complejidad del algoritmo es $O(n)$, por lo que si el número esperado de clústeres es pequeño en comparación con el número de puntos, es razonable iterar sobre el número de clústeres hasta encontrar el valor óptimo.

Algoritmo 3 k-means

Require: Número de clústeres k

1: Seleccionar k puntos aleatorios del espacio como centros de los clústeres
2: **while** Los centros de los clústeres se han movido respecto a la iteración anterior **do**
3: Asignar cada punto a su centro más próximo
4: Calcular los nuevos centros de los clústeres haciendo la media de los puntos que pertenecen a cada uno de ellos

2.2.2 k-medioids

Frente a la elección de un punto del espacio como centro del clúster, los algoritmos basados en k-medioids asignan el centro del clúster al elemento más centrado del clúster.

Con el algoritmo Partitioning Around Medoids (PAM) [9] se eligen k elementos como centros de los clústeres, y se compara con cada uno del resto de elementos. Se evalúa una función de coste de sustitución del centro del clúster $\mathbf{C}_i$ por un nuevo centro $\mathbf{C}_h$

$$TC_{ih} = \sum_{j=1}^{n} C_{ijh}, \tag{5}$$

donde C_{ijh} es el coste de sustitución para cada punto j y $\mathbf{C}_h$ es el punto de la base de datos más próximo a la posición media de los puntos del clúster. Este coste se calcula como

$$C_{ijh} = \begin{cases} d(\mathbf{x}_j, \mathbf{C}_{j2}) - d(\mathbf{x}_j, \mathbf{C}_i) & \text{, si } \mathbf{x}_j \text{ pertenece al clúster } \mathbf{C}_i \text{ y } d(\mathbf{x}_j, \mathbf{C}_{j2}) \geq d(\mathbf{x}_j, \mathbf{C}_h) \\ d(\mathbf{x}_j, \mathbf{C}_h) - d(\mathbf{x}_j, \mathbf{C}_i) & \text{, si } \mathbf{x}_j \text{ pertenece al clúster } \mathbf{C}_i \text{ y } d(\mathbf{x}_j, \mathbf{C}_{j2}) \leq d(\mathbf{x}_j, \mathbf{C}_h) \\ d(\mathbf{x}_j, \mathbf{C}_h) - d(\mathbf{x}_j, \mathbf{C}_{j2}) & \text{, si } \mathbf{x}_j \text{ pertenece al clúster } \mathbf{C}_{j2} \text{ y } d(\mathbf{x}_j, \mathbf{C}_{j2}) \geq d(\mathbf{x}_j, \mathbf{C}_h) \\ 0 & \text{, si } \mathbf{x}_j \text{ pertenece al clúster } \mathbf{C}_{j2} \text{ y } d(\mathbf{x}_j, \mathbf{C}_{j2}) \leq d(\mathbf{x}_j, \mathbf{C}_h) \end{cases}, \tag{6}$$

con $\mathbf{C}_{j2}$ el segundo centro de clúster más cercano a $\mathbf{x}_j$. El algoritmo PAM se desarrolla en (Alg.4).

Algoritmo 4 PAM

Require: Número de clústeres k

1: Escoger k puntos aleatorios como centros de los clústeres
2: **while** True **do**
3: **for** i=1:k, h=1,n **do**
4: Calcular TC_{ih}
5: **if** min(TC_{ih})<0 **then**
6: Colocar el centro C_i en el punto C_h
7: **else**
8: Salir del ciclo

Una estrategia basada en PAM aplicada a sistemas con una gran cantidad de puntos es Clustering LARge Applications (CLARA) [9]. Para elegir correctamente los centros de los clústeres sin hacer operaciones sobre toda la base de datos, en CLARA se coge una muestra aleatoria de cierto tamaño. Aplicando PAM a esa muestra, se pueden seleccionar unos candidatos. Se compara entonces la idoneidad de los centros obtenidos a partir de esta muestra con los centros obtenidos en iteraciones previas, guardando el mejor resultado obtenido. Este algoritmo

se puede iterar un número dado de veces, teniendo en cuenta que nunca se va a encontrar un resultado tan bueno como en PAM. La ventaja de CLARA frente a PAM, es que mientras PAM opera sobre todos los puntos con un coste de $O(k(n-k)^2)$, CLARA opera sobre un subconjunto más pequeño, logrando una mayor rapidez en la búsqueda de unos pocos clústeres en una gran base de datos $O(k^3 + kn)$.

El algoritmo Clustering Large Applications based on RANdomized Search (CLARANS) [10] agrupa la estrategia de búsqueda de los centros de PAM con la búsqueda de candidatos a través de una selección aleatoria de CLARA. Para ello, se visita un número prefijado de vecinos candidatos para ser el centro, intentando minimizar el ICS (Eq.4). Cuando se termina, se compara el centro del clúster obtenido con los obtenidos anteriormente, guardando el mejor en cada iteración (Alg.5). Para obtener todos los centros de los clústeres, basta con aplicar este algoritmo k veces sobre puntos que no sean ya centros, con k el número de clústeres que se buscan. Si el objetivo es asignar todos los puntos a su respectivo clúster, basta con calcular la distancia mínima a cada centro de cada uno de los puntos.

Algoritmo 5 Iteración del algoritmo CLARANS

Require: Número máximo de visitas a vecinos maxVecinos, número máximo de iteraciones maxIter

```
Inicializar el coste mínimo de sustitución, minCost=∞.
for i=1:maxIter do
    Escoger un punto aleatoriamente, x_c
    for j=1:maxVecinos do
        Escoger otro punto aleatoriamente, x_s
        Evaluar TC para x_c y x_s
        if TC<minCost then
            x_c=x_s
    if TC<minCost then
        minCost=TC
        Guardar x_c como el mejor centro
Devolver el mejor centro
```

2.2.3 Fuzzy means

En los dos tipos de algoritmos mencionados anteriormente, la pertenencia o no de un punto a un clúster es categórica, pertenece o no pertenece. Esta forma de asignación puede llevar a que un punto en la frontera de un clúster pueda cambiar demasiadas veces de clúster. Esta forma de asignación de los puntos a un clúster se denomina hard-clustering. Una forma de evitar este coste es usar una asignación gradual normalizada, soft-clustering o fuzzy-clustering.

La pertenencia a un clúster se puede visualizar fácilmente a través de una matriz de incidencia $n \times k$, donde n es el número de puntos y k el de clústeres. La estrategia del fuzzy-clustering consiste en calcular una función de cercanía del punto a cada uno de los centros. Así, se puede asignar un grado de pertenencia normalizado $D(\mathbf{x}_j, C_i)$ del nodo x_j al clúster C_i, que tendrá el mismo tamaño que la matriz de incidencia, definido como

$$D(\mathbf{x}_j, C_i)^{-1} = \sum_{p=1}^{k} \left(\frac{d(\mathbf{x}_j, \mathbf{C}_i)}{d(\mathbf{x}_j, \mathbf{C}_p)} \right)^{1/(m-1)}, \tag{7}$$

con m el parámetro de esponjosidad. Este parámetro controla el nivel de pertenencia los puntos al resto de clústeres. Cuanto más grande sea el valor de m, más repartido estará el nivel de pertenencia de un punto sobre el resto de clústeres. De esta manera, la función que debe minimizar el algoritmo se puede escribir como

$$J = \sum_{j=1}^{n} \sum_{i=1}^{k} d(\mathbf{x}_j, \mathbf{C}_i) \cdot D(\mathbf{x}_j, C_i). \tag{8}$$

Si la distancia entre puntos del espacio viene dada por una métrica euclídea, el algoritmo FCM (Fuzzy c-means) [11] propone asignar el centro del clúster a un punto calculado como

$$\mathbf{C}_i = \frac{\sum_{j=1}^{n} D(\mathbf{x}_j, C_i)^m \cdot \mathbf{x}_j}{\sum_{j=1}^{n} D(\mathbf{x}_j, C_i)^m}. \tag{9}$$

Con esta definición, se pasaría a minimizar la expresión (Eq.8). Se presenta en (Alg.6) el algoritmo que implementa esta idea.

Algoritmo 6 Fuzzy c-means

Require: Parámetro de esponjosidad m, límite iteraciones maxIter, límite de convergencia ε

1: Escoger k puntos aleatorios como centros de los clústeres
2: **for** i=1:maxIter **do**
3: Calcular los centros de los clústeres usando la ecuación (Eq.9)
4: Calcular J_t con la ecuación (Eq.8) para esta iteración
5: **if** $J_{t-1} - J_t < \varepsilon$ **then**
6: Salir del ciclo
7: Devolver los centros de la última iteración

Otra manera de definir la función a minimizar, es aplicando una mezcla gaussiana[1] a la ecuación anterior

$$F_i = \frac{\sum_{j=1}^{n} D(x_j, C_i)^m \cdot (x_j - C_i) \otimes (x_j - C_i)^T}{\sum_{j=1}^{n} D(x_j, C_i)^m}, \tag{10}$$

con $\otimes$ siendo el producto tensorial.

Partiendo de esta definición, el algoritmo GG (Gath–Geva) [13], trata de minimizar la suma de los hipervolumen difuso (V_i) de todos los clústeres

$$FHV = \sum_{i=1}^{k} V_i, \tag{11}$$

donde el hipervolumen difuso de un clúster está dado por la raíz del determinante de F_i

$$V_i = \sqrt{|F_i|}. \tag{12}$$

El problema de los algoritmos particionales es que si el número de clústeres está por debajo del óptimo, los puntos aislados se acaban asignando a un clúster. Si por el contrario, se han buscado demasiados clústeres, se pueden no detectar clústeres poco densos. Además, este tipo de algoritmos, por la definición de idoneidad del clúster dada por (Eq.8), solo son capaces de identificar correctamente clústeres que tienden a ser esféricos. Para solucionar estos problemas hay que usar un tipo de estrategia distinta.

[1] Una mezcla es una combinación normalizada de distribuciones $f_i(x)$, $\sum_{i=1}^{k} p_i f_i(x)$, $\sum_{i=1}^{k} p_i = 1$. Como su propio nombre indica, en una mezcla gaussiana todas las distribuciones $f_i(x)$ son funciones gaussianas.

2.3 Algoritmos basados en la densidad

Una forma intuitiva para encontrar clústeres es buscar regiones con mayor densidad de puntos, separadas entre sí por regiones con menor densidad. En este simple principio se basan los algoritmos basados en la densidad. El objetivo de estos algoritmos es encontrar máximos locales en la densidad y acotarlos adecuadamente, de tal manera que se obtengan clústeres con las propiedades impuestas. Entre estos algoritmos destacan 2:

2.3.1 Density Based Spatial Clustering of Applications with Noise (DBSCAN)

Para este algoritmo se usa la definición de clúster que diferencia puntos de core y de frontera, dada en la sección 1.2. DBSCAN se basa en la expansión de un clúster a partir de un candidato, iterando hasta haber visitado todos los puntos al menos una vez [6]. DBSCAN usa la definición de clúster tipo density-reachable, por lo que para lanzarlo se tiene que definir una distancia máxima de interacción entre puntos (b) y un mínimo de puntos con los que tiene que interactuar otro para poder considerarse un punto core ($minVecinos$).

Para identificar los clústeres, DBSCAN asigna a cada punto 3 posibles etiquetas: no etiquetado, referencia del clúster al que pertenece y ruido. Los puntos etiquetados como ruido serán puntos que no podrán ser puntos de core en ningún caso, ya que no tienen el número mínimo de puntos cerca para ello. Una vez etiquetado como ruido, este deja de estar dentro de la lista de puntos desde donde se puede expandir un clúster. La aplicación de este algoritmo de expansión (Alg.7) es iterada hasta que todos los puntos de la base de datos tienen una etiqueta.

Algoritmo 7 Expansión de un clúster para DBSCAN

Require: b, minVecinos, punto de inicio startPoint, referencia para el nuevo clúster clusterName, base de datos con etiquetas

```
 1: candidatos=puntos con d(startPoint,candidato)<b
 2: if número de candidatos<minVecinos then
 3:     etiqueta de startPoint=ruido                    ▷ startPoint no es un punto de core
 4: else
 5:     etiqueta de startPoint=clusterName
 6:     while número de candidatos > 0 do
 7:         puntoActual=un punto de candidatos
 8:         candidatosActuales=puntos con d(puntoActual,candidato actual)<b
 9:         if número de candidatosActuales>minVecinos then
10:             etiqueta de puntoActual=clusterName     ▷ puntoActual es un punto de core
11:             añadir candidatosActuales a candidatos
12:         else
13:             etiqueta de puntoActual=ruido           ▷ puntoActual es un punto de frontera
14:         eliminar puntoActual de candidatos
```

El coste de este algoritmo de expansión depende en gran medida de la operación de búsqueda de puntos cercanos, hecha en las líneas 1 y 8. En una base de datos desordenada esta búsqueda tiene un coste de $O(n)$, lo que haría que el coste de DBSCAN aumentara al menos hasta $O(n^2)$. Para hacer eficiente este algoritmo, es necesario organizar la base de datos de tal manera que esta búsqueda sea más rápida. Usando una estructura de R*-tree desarrollada en detalle en [14], se puede optimizar esta operación hasta un coste de $O(\log n)$. De esta forma, DBSCAN tiene

un coste total de $O(qn \log n)$, donde q es el número de veces que se visita cada punto de media. Para una estructura de árbol y unos parámetros adecuados, q debería ser mucho menor que n, por lo que la dependencia del coste está principalmente en el tamaño del problema. Para asegurar que se cumpla esta condición, la estructura del árbol tiene que distribuir los puntos en sus nodos de una manera lo más uniforme posible.

En este algoritmo se toman como parte del clúster solamente los puntos del core. Si se quisieran tomar también los puntos de la frontera se tendría que hacer una pasada extra sobre todos los puntos de cada clúster, buscando todos los puntos a una distancia $< b$ que se han etiquetado como ruido. Sin embargo, esto podría provocar choques entre clústeres en los casos en los que un punto esté en "disputa" por pertenecer a dos clústeres distintos, por lo que habría que definir una estrategia para resolver estos casos. Una solución podría ser asignar el punto en disputa al clúster con el punto de core más próximo. En este trabajo se ha decidido dejar estos puntos fuera del clúster.

2.3.2 DENsity-based CLUstEring (DENCLUE)

Otra aproximación a los algoritmos basados en la densidad es buscar una función de densidad. Así, simplemente imponiendo una densidad mínima (g_{min}), el espacio que cumpla la condición $g(x) > g_{min}$ será un clúster. La función de densidad viene dada por la suma de las funciones de influencia de todos los puntos

$$g(x) = \sum_{i=1}^{n} g_i(x). \tag{13}$$

Esta función de densidad dependerá de la naturaleza de los datos que se estén usando y la métrica del espacio. Para un punto en la posición x_i, se proponen dos tipos de funciones de influencia básicas con sus respectivos gradientes sobre un eje d: una cuadrada

$$g_i(x) = \theta(b - d(x_i, x)),\ \nabla g_i^{(d)}(x) = -\frac{\partial d(x_i, x)}{\partial x^{(d)}} \cdot \delta(b - d(x_i, x)), \tag{14}$$

con $\theta(x)$ la función escalón de Heaviside[1] y $\delta(x)$ la delta de Dirac; y una gaussiana

$$g_i(x) = \exp\left(\frac{-d(x_i, x)^2}{2b^2}\right),\ \nabla g_i^{(d)}(x) = -\frac{\partial d(x_i, x)}{\partial x^{(d)}} \cdot \frac{d(x_i, x)}{b^2} \cdot \exp\left(\frac{-d(x_i, x)^2}{2b^2}\right). \tag{15}$$

Una función de influencia cuadrada da como resultado una clusterización idéntica a la dada por DBSCAN; mientras que la gaussiana puede ser más efectiva para separar mejor clústeres próximos. Para el caso de los clústeres de galaxias, puede dar más información definir la función de influencia a partir del campo gravitatorio. Para mantener los valores de la función de influencia entre 0 y 1, se puede desplazar la función del campo, de tal manera que la función de influencia sea

$$g_i(x) = \frac{1}{(1 + d(x_i, x))^2},\ \nabla g_i^{(d)}(x) = \frac{\partial d(x_i, x)}{\partial x^{(d)}} \cdot \frac{-2}{(1 + d(x_i, x))^3}. \tag{16}$$

Esta idea de generar una función densidad puede parecer sencilla, pero a la hora de computar esta función directamente aparece un problema, y es que para computar esta cantidad habría que tener en cuenta todos los puntos del espacio, llegando a un algoritmo de coste inasumible.

[1] Definida como $\theta(x \geq 0) = 1$, $\theta(x < 0) = 0$

Para evitar este coste, el algoritmo DENCLUE [15] propone hacer este cálculo en dos pasos. El primero, trata de limitar el espacio de búsqueda a las regiones del espacio donde haya puntos. Para ello, se usa la estrategia de los algoritmos basados en rejilla (Sec.2.4) para determinar qué celdas están ocupadas, y cuales son las celdas con las que puede interactuar. La elección del tamaño de esta rejilla depende de la base de datos, pero los autores del algoritmo proponen que con celdas de lado $2b$ se obtiene una buena optimización. Partiendo de una rejilla con mucha densidad, el segundo paso trata de encontrar el máximo local de la función densidad. Con este máximo, y mediante una estrategia de construcción de la función densidad iterativa, se busca encontrar todos los puntos dentro de la región con densidad suficiente. El algoritmo se detalla a continuación (Alg.8), pero su idea básica es parecida a la construcción por expansión usada en DBSCAN.

Algoritmo 8 Identificación de un clúster en DENCLUE

Require: Una base de datos separada en rejillas con información de las rejillas contiguas, parámetro de distancia mínima b, función de influencia

1: Buscar una celda con una densidad alta
2: Calcular el punto medio de los puntos en esa celda, c_1
3: Buscar los puntos a una distancia 4b de c_1, objetosCercanos ▷ El factor de 4 está ajustado por los autores
4: $x_c = c_1$
5: **repeat** ▷ Procedimiento de "escalar la montaña" para obtener el máximo local
6: $x'_c = x_c$
7: $x_c = x_c + \nabla g(x_c)/\,|\nabla g(x_c)|$ ▷ Para evaluar la función densidad usar solo objetosCercanos
8: Buscar objetosCercanos para el nuevo punto x_c
9: Guardar los puntos x_i tales que $d(x_c, x_i) < b/2$
10: **until** $g(x_c) < g(x'_c)$
11: Asignar todos los puntos guardados al clúster con el máximo x_c

Para identificar todos los clústeres, habría que identificar todos los puntos que pueden pertenecer a un clúster. Como a la hora de "escalar" hasta el máximo de la función densidad se van guardando los puntos del camino, este algoritmo reduce la cantidad de veces que se tiene que iterar hasta recorrer todos los puntos. Además, la identificación de los posibles clústeres a través de la ocupación de la rejilla permite detectar fácilmente los puntos de ruido que estén suficientemente dispersos.

2.4 Algoritmos basados en rejilla

Estos algoritmos se basan en idea de divide y vencerás. Al dividir el espacio en una serie de celdas pequeñas, el espacio de búsqueda se reduce drásticamente. De esta manera, los únicos puntos del espacio con información útil son más rápidamente accesibles desde cualquier otro punto del espacio. Reducir el tiempo de búsqueda ayuda en gran medida a reducir la complejidad de los algoritmos de clusterización.

2.4.1 STatistical INformation Grid-based algorithm (STING)

El algoritmo STING [16] hace uso de una estructura en rejilla jerarquizada para minimizar al máximo el tiempo de cada búsqueda de puntos próximos. Para ello, se parte de una raíz que

ocupe todo el espacio de los datos. Se divide la celda raíz en un número dado de celdas de igual tamaño. Sus autores proponen dividir una celda de 2 dimensiones en 4 regiones de igual tamaño, por lo que en 3 dimensiones parece razonable dividirla en 8. Todas estas subdivisiones se organizan en un árbol, con un tamaño de celda del nivel más bajo del árbol tal al dividir esa celda, su arista sea más pequeña que la distancia característica del problema (b).

Para poder obtener información rápidamente, cada celda contiene información acerca de los puntos que contiene. Para el caso de puntos en el espacio, la información relevante se limita al número de puntos por celda. Para otro tipo de análisis se pueden dar datos como la media, la desviación estándar, los valores máximo y mínimo y el tipo de distribución que siguen los datos. Generar esta estructura de datos tiene un coste de $O(nl)$, con l el número de capas del árbol, pero la ventaja que se obtiene aparece al reducir el tiempo de búsqueda de un punto a $O(l)$.

Con la base de datos organizada, se propone una pequeña variación del algoritmo STING para resolver un problema de clusterización de puntos en un espacio (Alg.9).

Algoritmo 9 Modificación de STING recursiva para detectar clústeres de puntos en un espacio

Require: Base de datos jerarquizada en l_{max} capas

1: **function** STING(l)
2: Estimar la probabilidad (p_{il}) de que la celda i de la capa l contenga o sea parte de un clúster comparando su número de puntos (n_{il}) con el del resto, por ejemplo $p_{il} = n_{il} / \sum_j n_{jl}$
3: Identificar las celdas que son susceptibles de tener un clúster por comparación al resto de la capa, por ejemplo las que cumplan $p_{il} \geq P$
4: **if** $l \neq l_{max}$ **then**
5: **for** todas las celdas escogidas en el paso 3 **do**
6: STING(l-1)
7: **else**
8: Identificar la celda como clúster si cumple un criterio de clusterización dado
9: **return**
10: Buscar celdas contiguas que sean un clúster y agruparlas si en conjunto forman también un clúster

Con esta modificación del algoritmo STING se consigue implementar una estrategia de expansión de un clúster a partir de una semilla, similar a la estrategia usada en DBSCAN (Sec.2.3.1). Para conseguir clústeres completos, sin bordes de sierra creados por el uso de la rejilla, hay que aplicar alguna pasada de otro algoritmo. Sin embargo, este algoritmo permite obtener unos candidatos de clústeres buenos en un tiempo reducido, por lo que podría usarse como punto de partida para otros algoritmos.

2.5 Algoritmos de grafos: Friends-of-Friends

Estos algoritmos se basan principalmente en organizar los datos en distintas estructuras de grafos. Frente a almacenar los datos de forma desordenada, estas estructuras permiten hacer búsquedas más rápidas. Debido a esta ventaja, otros algoritmos suelen incorporan esta forma de organizar los datos. Dentro de todos los algoritmos basados únicamente en grafos destacan los siguientes:

2.5.1 Minimal Spannning Tree (MST)

Un MST se construye a partir de un grafo completo, en el que cada arista tiene un peso asociado $w_{ij} \geq 0$. En el caso de los clústeres, este peso es la distancia entre los dos nodos que conecta una arista. El MST se construye de tal manera que todos los nodos estén conectados y que la suma de los pesos de las aristas sea mínima. Siendo el peso de todas las ramas un número real positivo, el MST tendrá una estructura de árbol como la definida en la sección 1.1.

Para construirlo, se puede usar un algoritmo simple propuesto en [17]. Este algoritmo tiene un coste $O(n^2)$, por lo que será el cuello de botella del algoritmo de clusterización.

Algoritmo 10 Algoritmo para obtener un MST

Require: Un grafo completo de n nodos

```
for i=1:n-1 do
    repeat
        Elegir una arista no visitada, tal que el peso w_{i,i-1} sea mínimo
        Añadir esta arista a la lista de aristas visitadas
    until La arista escogida no forma un ciclo con el MST
    Añadir la arista al MST
```

Una propiedad del MST es que si se elimina la arista con mayor peso, los dos subgrafos restantes formarán dos MST. De esta manera, es fácil generar un algoritmo que cree clústeres tan solo iterando este paso hasta eliminar todas las aristas con un peso mayor a una distancia máxima. Para que uno de los subgrafos finales se pueda identificar como un clúster se pueden usar diferentes estrategias. La más simple, definir un número mínimo de elementos que tiene que tener un clúster.

Algoritmo 11 Algoritmo de clusterización basado en MST

Require: Un grafo completo de n nodos, una distancia máxima de conexión entre nodos b

1: Generar un MST a partir del grafo de entrada
2: Eliminar las aristas con $w_{ij} >$ b
3: Comprobar la validez los subgrafos obtenidos como clústeres

Este algoritmo falla en identificar clústeres con formas "extrañas". Se pueden detectar clústeres lineales, en los que cada elemento (o gran parte de ellos) solamente tiene 2 nodos a una distancia $< b$. Otro problema del algoritmo es que puede tender a agrupar clústeres distintos. Puede ocurrir que donde se tienen 2 clústeres claramente diferenciados pero que conectados por un clúster lineal, se detecten ambos como un solo gran clúster.

2.5.2 Hybrid Minimal Spanning Tree: Gath-Geva (Hybrid MST-GG)

Una forma de solucionar el problema de la agrupación incorrecta de clústeres con MST, es usando un mejor criterio para comprobar la validez de los clústeres obtenidos. Para ello, se utilizan estrategias usadas en algoritmos de fuzzy-clustering (Sec.2.2.3) aplicadas a los resultados del algoritmo de MST [13].

La idea detrás de esta mejora está en usar el criterio para minimizar el fuzzy hipervolume de los clústeres obtenidos por MST. Para partir todavía más los árboles obtenidos, se propone

añadir un parámetro de validez extra, a través de la medida de Fukuyama-Sugeno. Se divide un clúster en dos subconjuntos de puntos k, la medida entonces viene dada por

$$FS(S) = \sum_{k=1}^{2} \sum_{j=1}^{N_k} (x_j^k - v_k)^2 - (v_k - v)^2, \tag{17}$$

con N_k el número de puntos x_j^k de la partición k, v_k la media de la posición de los puntos en la partición k y v la media de todos los puntos del clúster del que se ha partido. Si este valor es pequeño, indica que el clúster tiene una densidad constante, y que por lo tanto es un buen candidato. Si es grande, indica una dispersión desigual de los puntos dentro del clúster, por lo que se podría encontrar una partición tal que FS para cada una de las nuevas particiones sea menor que la anterior.

Aunque esta estrategia puede servir para mejorar la calidad de los clústeres obtenidos por MST, este algoritmo tiene el mismo problema que los algoritmos particionales. Los nuevos clústeres obtenidos solo podrán tener formas que tiendan a elipsoides. Ajustando adecuadamente los parámetros de la parte del algoritmo GG, sería posible diferenciar dos clústeres unidos por un hilo de tamaño superior al tamaño de estos.

2.5.3 Dual Tree Friends-of-Friends (DTFoF)

El algoritmo DTFoF propuesto en [18] recorre en paralelo una estructura de KD-tree, reduciendo en cada pasada el número de pasos necesarios para recorrerlo. En líneas generales, este algoritmo genera clústeres fusionando clústeres conectados por parejas. Para que este algoritmo sea eficiente, se siguen una serie de estrategias.

Algoritmo 12 Búsqueda de puntos cercanos en un KD-tree

Require: KD-tree, un punto de inicio, distancia mínima b

Obtener los límites de la celda de inicio, celdaInicio
Guardar una lista con las celdas que pueden estar a una distancia < b, candidatos=[1]
Guardar una lista con los puntos en las celdas cercanas, resultado=[]
while candidatos no está vacío **do**
    for nodo=candidatos **do**
        if limDistanciaMin(celdaInicio,celda del candidato) < b **then**
            if Los hijos de candidato son hojas **then**
                resultado=[resultado, hijos de candidato]
            else
                candidatosNew=[candidatosNew, hijos de candidato]
    candidatos=CandidatosNew, candidatosNew=[]
Comprobar la distancia entre los puntos en resultado, quitar los puntos no conectados

function LIMDISTANCIAMIN(celdaA,celdaB)
    resultado=0
    for dim=1:dimTotal **do** ▷ dimTotal es la dimensión del espacio
        if Los bordes de una celda caen dentro de los bordes de la otra **then**
            continue
        Calcular la distancia Manhattan mínima entre los límites de las celdas en cada dirección, resultado=min(resultado,|maxA-maxB|,|maxA-minB|,|minA-maxB|,|minA-minB|)

Algoritmo 13 Creación de un KD-tree

Require: Base de datos con puntos k dimensionales, número mínimo de puntos n_{max}, distancia mínima b

Inicializar el tamaño de la celda raíz, celdas(1)=[$x_1^{min}, x_1^{max}, x_2^{min}, x_2^{max}, ..., x_k^{min}, x_k^{max}$]
Inicializar el KD-tree, KDtree(1)=[1,2,3] ▷ [padre,hijo 1,hijo 2]
Inicializar una lista de asignación de los puntos, auxKDtree=[]
En cada iteración, crear una lista de nodos pendientes para partir, pendientes=[1]
Guardar el número de celdas creadas hasta el momento, nCeldas=1
Guardar la dimensión sobre la que se va a hacer la siguiente partición, partición=1
while pendientes no está vacío **do**
    for nodo={pendientes} **do**
        if número de puntos en nodo > n_{max} AND diagonal celda > $\sqrt{d \cdot b^2}$ **then**
            Partir las celdas según la dimensión dada por partición
            Asignar estos valores calculados a celdas(nCeldas+1) y celdas(nCeldas+2)
            KDtree(nodo)(2:3)=[nCeldas+1, nCeldas+2] ▷ Actualizar KD-tree
            KDtree(nCeldas+1)(1)=nodo, KDtree(nCeldas+2)(1)=nodo
            pendientes=[pendientes,[nCeldas+1, nCeldas+2]] ▷ Actualizar pendientes
            nCeldas=nCeldas+2
        else
            for puntos={puntos en nodo} **do** ▷ Al ser hojas, los puntos no tienen hijos
                Asignar puntos correspondientes al nodo, auxKDtree(punto)(1)=nodo
    partición=(partición módulo k) +1
KDtree=[KDtree, auxKDtree]▷ Se guardan los puntos y los nodos en una misma estructura

El KD-tree se define como un árbol con $M < n$ nodos. Un nodo tiene asociados todos los puntos de sus sucesivos hijos, pero cada punto de la base de datos está asociado solo a una hoja. La conexión entre los puntos se dará a través de una lista de parentesco. Esta lista se define de tal manera que el elemento en la posición i es hijo del elemento $H(i)$. Si $H(i) = i$, quiere decir que ese punto es una raíz.

Para generar el árbol, se usa una estrategia de partición sucesiva del espacio en distintas direcciones, según cual sea la arista más larga de la celda. Este proceso de partición se repite hasta que se ha llegado a una profundidad adecuada, dado por un límite en el tamaño de la celda o por el número de elementos en ella. Para hacer más rápido el cálculo de los límites de distancia entre los puntos, se elige una implementación del KD-tree que contiene información sobre los límites de cada celda. Además, dado que la base de datos tendría un tamaño demasiado grande, se optimiza el espacio guardando solamente cada punto una vez en su padre, mejorando en este sentido la implementación propuesta por los autores (Alg.13).

Una vez se tiene la estructura de datos creada, la operación de búsqueda de puntos cercanos a otro se puede acelerar usando los límites de las celdas. Usando un razonamiento similar al usando en el artículo donde se propuso por primera vez esta estructura de datos [19], se puede estimar un límite superior para la distancia mínima entre dos puntos fácilmente usando la distancia Manhattan. Así, la operación de buscar puntos cercanos a otros se puede hacer visitando solamente los nodos que tengan sus celdas a una distancia inferior al parámetro de distancia mínima b. Así, se propone el siguiente algoritmo para buscar los puntos cercanos a otro (Alg.12).

Definidas estas funciones y la estructura de datos, basta con generar los clústeres usando

un algoritmo de expansión. La ventaja principal de esta implementación del KD-tree, es que todos los puntos dentro de una celda del nivel más bajo están conectados entre sí. Esto se puede hacer buscando todas las uniones entre puntos y entre cada par de celdas cercanas. Pero para comprobar si los puntos de dos celdas están conectados, basta con encontrar una pareja de puntos conectados para que todos lo estén. Si se repite la búsqueda hasta no tener más puntos nuevos sobre los que buscar, se tendrán todos los puntos del clúster.

Cuando se tiene un árbol con varios niveles, el tiempo de búsqueda aumenta con la profundidad. Para mantener este tiempo de búsqueda bajo, se propone un mecanismo de extensión, cuyo resultado es que todos los hijos de una raíz se conviertan en sus hojas. Para ello se tiene el siguiente algoritmo (Alg.14), que además, aplica esta operación a todos los puntos visitados en el proceso.

Algoritmo 14 Extensión de una raíz (Algorithm 2 [18])

Require: Una lista H de parentesco, un punto de inicio puntoInicio

```
raíz=puntoInicio
puntosVisitados=puntoInicio
while H(raíz)≠raíz do
    Añadir raíz a puntosVisitados
    raíz=H(raíz)
for i={puntosVisitados} do
    H(i)=raíz
return raíz
```

Para encontrar todos los pares de puntos conectados a una distancia $< b$ entre 2 nodos se usa una función de enumeración de parejas ENUM, desarrollado en (Alg.15). Esta función hace uso de la distancia Manhattan para determinar si evalúa las distancias entre cada par de puntos de 2 celdas, dejando de calcular si las celdas del árbol están demasiado lejos.

Algoritmo 15 Enumeración recursiva de uniones de puntos (basado en Algorithm 1 [18])

Require: Estructura de árbol binario (Tree), nodoA, nodoB, distancia máxima de enlace b

```
if Los hijos de nodoA y nodoB son puntos then
    return parejas de puntos a una distancia <b
else
    if los hijos de nodoA son puntos then
        Intercambiar nodoA↔nodoB
    if limDistanciaMin(hijoA de nodoA,nodoB)<b then
        parejasA=ENUM(hijoA de nodoA,nodoB)
    if limDistanciaMin(hijoB de nodoA,nodoB)<b then
        parejasB=ENUM(hijoA de nodoA,nodoB)
    return parejasA ∪ parejasB
```

Tras haber obtenido todo el conjunto de parejas de puntos conectadas, basta con aplicar el algoritmo (Alg.14) para identificar cada clúster con el índice de su raíz. Dado que las raíces de los clústeres serán puntos arbitrarios, para obtener la información de la lista de parentesco habrá que encontrar todos los puntos con un mismo valor. Para determinar si un clúster es válido o no, basta con contar el número de puntos del clúster.

3. Implementación de algunos algoritmos

Parte de este trabajo consiste en implementar y adaptar algunos de los algoritmos de clusterización presentados al problema de la búsqueda de clústeres de galaxias. Los algoritmos que se vayan a implementar tienen que cumplir las siguientes condiciones:

- Son algoritmos de fuerza bruta; es decir, son algoritmos que se lanzan sin tener conocimiento previo acerca de la base de datos.
- El número de clústeres encontrados no tiene que ser un parámetro de entrada. Para lanzar la búsqueda solo tienen que darse las condiciones para identificar un conjunto de puntos como clúster.
- Los clústeres identificados tienen que poder tener una forma arbitraria.
- Debido al tamaño del problema, los algoritmos tienen que usar una cantidad razonable de espacio de memoria.
- En la medida de lo posible, los algoritmos tienen que ser robustos, en el sentido de que tienen que dar el mismo resultado en sucesivas llamadas.

Dadas estas limitaciones, todos los algoritmos basados en k-means quedan inmediatamente descartados por necesitar un número dado de clústeres. Algunos autores [10] [11] [20] [21] afirman que estos algoritmos pueden lanzarse con un número distinto de clústeres que buscar, y así encontrar el número de particiones que mejor se ajusta a las condiciones dadas. Sin embargo, esta estrategia es poco eficiente si no se conoce un rango aproximado del número de clústeres esperados, o si el orden de magnitud de este número es demasiado elevado.

Uno de los problemas de todos los algoritmos particionales es la forma que tienen los clústeres encontrados. Debido a la forma en la que se calcula la idoneidad de las particiones, todos los clústeres tienen forma elipsoidal. Esto hace que no se puedan detectar clústeres con formas convexas o clústeres que se extienden como las ramas de un árbol. Dado que los clústeres de galaxias no tienen a priori una forma definida, los algoritmos tienen que poder adaptarse a clústeres de formar arbitrarias. Por esta razón, se han desechado todos los algoritmos particionales.

Los programas que se han hecho en este trabajo se han escrito en lenguaje MATLAB, en su versión MATLAB 9.7 (R2019b). Se ha elegido este lenguaje por su facilidad de programación, portabilidad, posibilidad de dibujar gráficas sin recurrir a programas externos y por sus funcionalidades de programación en paralelo, en concreto, la posibilidad de paralelizar los ciclos *for* fácilmente.

Para probar los distintos programas escritos en esta sección, se han lanzado varios *benchmarks* analizando su rendimiento frente a distintos problemas de tamaño variable. Para ello se han usado 5 conjuntos de puntos sintéticos de la base de datos SEQUOIA2000: *streams, islands, 11, 22* y *33* [22]. También se ha usado la base de datos *worms*, creada para [23] y que se ha accedido desde el repositorio de *benchmarks* creado por sus mismos autores [24]. Estas bases de datos están dadas por puntos en un espacio de dimensión 2. Los ficheros que contienen las distintas bases de datos dan las coordenadas de cada punto por líneas, ordenados de forma que no ofrezca ninguna ventaja para el cálculo de los clústeres. Además de esto, se ha creado una base de datos con el objetivo de ilustrar el funcionamiento de los algoritmos (Fig.A.1).

En este capítulo se comentan las implementaciones de los algoritmos escogidos:

3.1 MST

Para implementar este algoritmo se ha decidido usar una representación del grafo según su matriz de incidencia. Dado que el tamaño de la matriz aumenta cuadráticamente con el tamaño del problema, el uso de la memoria de una implementación normal puede ser demasiado pesada. Para usar el mínimo espacio en memoria, se ha elegido una representación de la matriz de incidencia triangular, en el que cada arista aparece en la matriz de incidencia solo una vez. Si además de esto se usa una representación de matriz dispersa, se consigue reducir el uso de memoria aproximadamente a la mitad.

La representación del MST se ha hecho usando una representación idéntica a la usada en [18]. Esta representación del árbol consiste en un vector a en el que cada uno de sus elementos $a(i)$ corresponde al padre del nodo i. Si se tiene que $a(i) = i$, indicará que el nodo i es una raíz. Esta representación es útil en el sentido de que a partir de la misma, un clúster puede representarse fácilmente modificando el árbol para transformarlo en uno con una raíz y el resto de elementos hojas. Se desarrolla esta transformación en la figura (Fig.2). De esta manera, al recorrer el vector del grafo, basta con buscar los índices de todos los elementos iguales. Se puede ir más allá e identificar directamente todos los elementos de un mismo clúster usando un identificador, teniendo así una forma general de representación de un clúster válida para todas las implementaciones que se vayan a hacer en este trabajo.

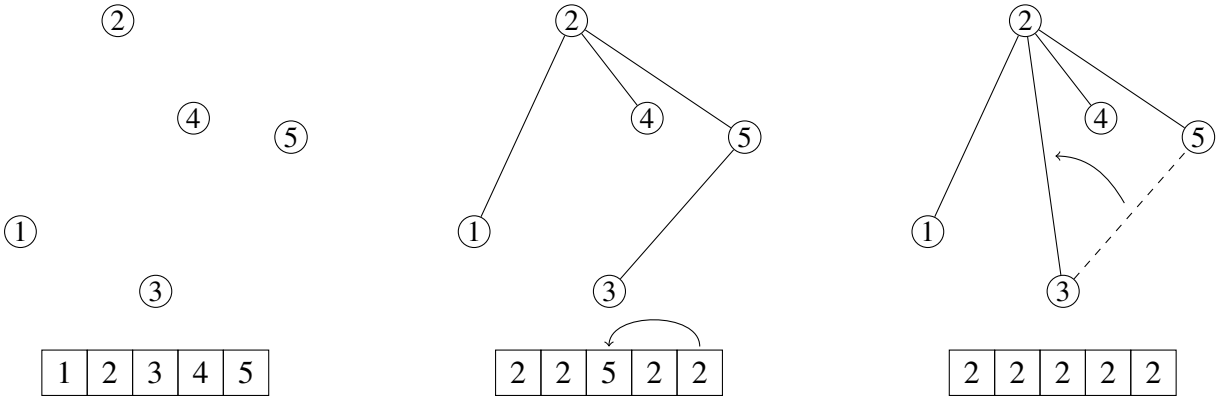

Fig. 2: Visualización de la transformación de un árbol arbitrario a un árbol con la raíz y el resto hojas. El recuadro debajo de cada grafo muestra el estado del vector de incidencia a. Antes de construir el árbol, cada nodo es su propia raíz (izquierda). En la construcción inicial del árbol se pueden tener un número arbitrario de niveles (centro). En cada iteración de la transformación se buscan todos los nodos cuyo padre no sea una raíz, en este caso el nodo 3. Estos nodos pasan a ser hijos del padre de su padre, tal que a(i)=a(a(i)). En este caso la transformación es a(3)=5 → a(3)=a(a(3))=2. De esta manera, tras $n-1$ iteraciones siendo n el número de niveles del árbol inicial, se consigue un árbol con un solo nivel (derecha).

Una primera versión del programa (`MST_v1`) implementa directamente el algoritmo (Alg.11). Se genera el grafo completo a partir de los puntos de la base de datos, y después se aplica la función para generar el MST. Se podan las ramas con una distancia mayor que la distancia máxima de enlace entre puntos y se identifican los distintos subgrafos resultantes. Hay que recalcar que el coste de construir un MST es en el peor de los casos casi cúbico $\Theta(n^3)$ y en el mejor es peor que un coste cuadrático $\Omega(n^2)$.

La operación de añadir una nueva arista es muy costosa, ya que para comprobar que cada

nuevo candidato no formará un ciclo hay que hacer $\Theta(v/2)$ iteraciones, siendo v el número de aristas del MST. Dado que en zonas con una gran densidad de puntos se tienen muchas aristas de longitud similar, el algoritmo de generación del MST pasará mucho tiempo comprobando aristas que no serán parte del árbol. Este coste de encontrar una nueva arista se hace considerable hacia el final del programa, en el que casi ninguna arista comprobada es válida para añadirse al MST. Para evitar este coste extra, se ha desarrollado una nueva versión del algoritmo (`MST_v1.1`) en el que no se genera un MST completo, sino que se buscan subgrafos conectados, siendo cada uno de ellos un MST. Para ello basta con generar un grafo inicial en el que solamente se añaden las aristas con una longitud $< b$, y después lanzar el mismo algoritmo de generación del árbol.

Aunque esta pequeña modificación sea mejor que la primera iteración del algoritmo, sigue sin ser útil para generar clústeres a partir de una base de datos con una gran cantidad de puntos. Por ello, se ha probado una nueva modificación (`MST_v2`), que intenta reducir al mínimo el número de cálculos del programa. Esta nueva versión genera un grafo solamente con las aristas de longitud $< b$, y después se lanza a calcular los clústeres buscando todos los puntos que están conectados entre sí. Aunque en esta versión se ahorren cálculos para generar los árboles, el coste de comprobación de la conectividad de los puntos es mayor. Esto es así por la necesidad de guardar todos los puntos visitados al no ser los subgrafos que se evalúan árboles. Pese a esta mejora, la complejidad computacional del algoritmo sigue siendo la misma. En la figura (Fig.A.2) del anexo se muestran algunos pasos de las distintas versiones del algoritmo desarrolladas.

Para realizar operaciones de búsquedas de ciclos y clústeres, se usa una estrategia de Depth First Search. Como ya se ha adelantado en la introducción, un árbol sigue siendo un árbol sin importar qué nodo se haya elegido como raíz. Esta propiedad permite recorrer un árbol de la misma manera empezando desde un nodo cualquiera. La estrategia Depth First Search aprovecha esto para recorrer un árbol pasando por cada punto de cada nivel, hasta encontrar el nodo que se estaba buscando.

Un problema de este algoritmo es encontrar una estrategia de paralelización que reduzca el tiempo de cómputo de forma significativa. El cálculo de las distancias entre puntos sí que es paralelizable, pero el tiempo para este paso es relativamente pequeño por sí mismo. Debido a la forma secuencial en la que se añaden nuevas aristas al MST, la generación del árbol no es paralelizable de ninguna manera. Al ser esta la operación más costosa del algoritmo, queda visto que la eficiencia de este algoritmo nunca va a llegar a ser adecuada para resolver problemas grandes. Los resultados de las pruebas se muestran en la gráfica (Fig.3).

Analizando esta gráfica se comprueba rápidamente que la generación de un MST, ya sea para todo el grafo (`MST_v1`) o para subgrafos (`MST_v1.1`) es una operación muy costosa. Además, se observa una gran variación del tiempo de cómputo entre distintas bases de datos. En ambos, los peores resultados se han obtenido para la base de datos de *worms*. Una peculiaridad de esta base de datos es que sus puntos tienen una distribución bastante uniforme en el espacio. Esto hace que se tengan que comprobar un mayor número de aristas hacia el final del algoritmo.

Descartado el algoritmo original MST, el *benchmark* hecho indica que la mejor opción es no generar un MST. Aún así, se sigue observando una diferencia en el tiempo de ejecución dependiendo del tipo de base de datos. Por este motivo, el ajuste parabólico realizado no se podrá usar para hacer una estimación fiable del tiempo de ejecución a priori. Sin embargo, si que se puede usar para comparar de forma aproximada el rendimiento frente al resto de algoritmos implementados. Haciendo un ajuste parabólico de todos los puntos obtenidos para

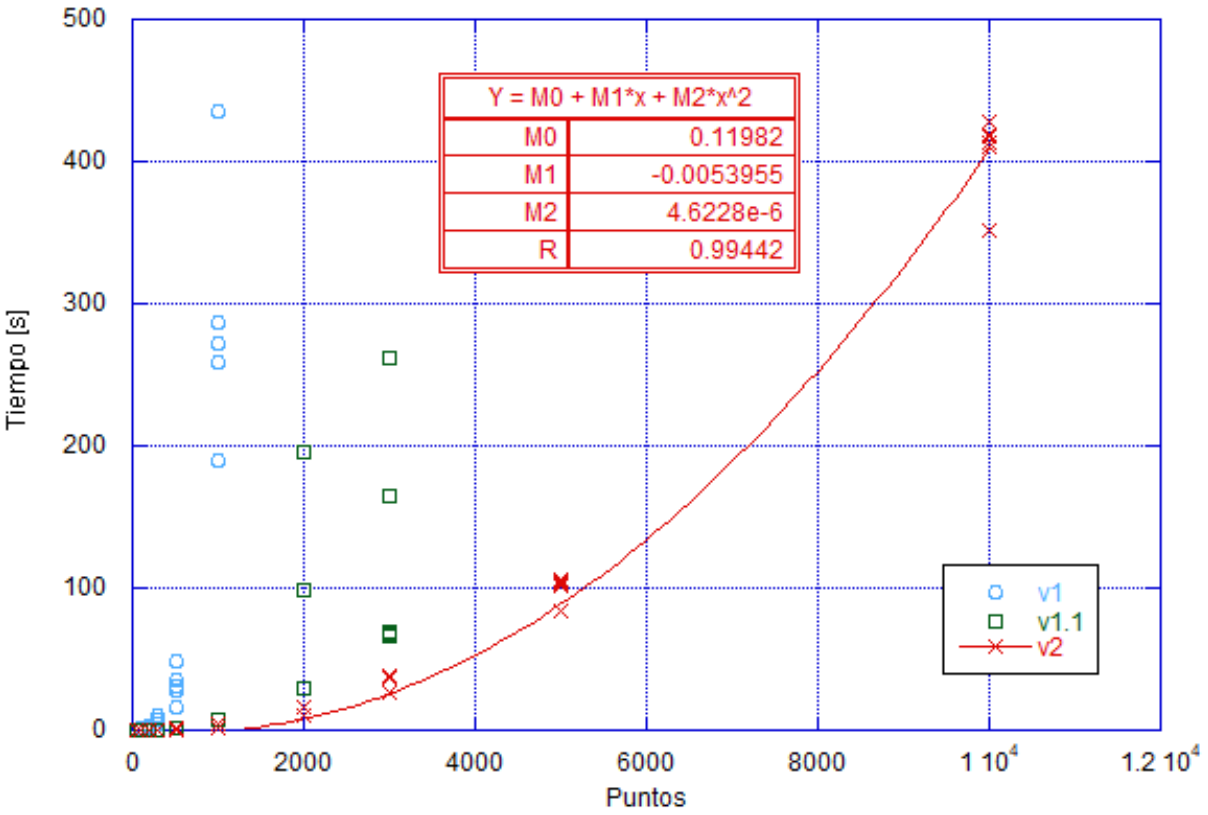

Fig. 3: Benchmarking de MST sobre distintas bases de datos y versiones, con un ajuste parabólico para la versión 2.

el *benchmark* usando `MST_v2`, la función que mejor se ajusta a estos es

$$T_{\text{MST\_v2}} = 0.120 - 5.40 \cdot 10^{-3} n + 4.62 \cdot 10^{-6} n^2 \ [s]. \tag{18}$$

3.2 DBSCAN

La implementación del algoritmo DBSCAN se ha hecho de manera directa a partir de lo desarrollado en la sección 2.3.1, salvo por pequeñas modificaciones para evitar cálculos redundantes, por ejemplo, del número de puntos en cada clúster.

Las modificaciones frente al algoritmo propuesto por los autores se han aplicado al algoritmo de creación del R*-tree. En el trabajo donde se plantea esta estructura de datos [14], se propone una serie de funciones y algoritmos orientados a trabajar en bases de datos de 2 dimensiones. Al no dar indicaciones de cómo trabajar en dimensiones superiores, se a tenido que modificar varios aspectos para adaptarlo a dimensiones superiores.

La función ChooseSplitAxis [14, Alg.CSA1] sirve para elegir el eje sobre el que se ordenarán los puntos de la celda para después elegir la partición. Para ello, se calcula el mínimo de la suma de las aristas de las celdas que se formarían al ordenar los puntos en cada una de las dimensiones y elegir las particiones con un tamaño mínimo. En el artículo original, esta operación busca minimizar el margen de las nuevas celdas, a través de su perímetro. En 3 dimensiones no es evidente el criterio que se debería usar. Se ha decidido minimizar el área de las caras del prisma. Para problemas de orden superior, simplemente se ha buscado minimizar el hipervolumen de las nuevas celdas.

De manera similar, ChooseSplitIndex [14, Alg.CSI1] se usa para elegir las dos particiones de una celda. En 2 dimensiones esto se hace minimizando la suma del área de las dos nuevas celdas. Al traducir esto a 3 o más dimensiones, se ha elegido usar como criterio de división minimizar la suma del volumen (o el hipervolumen) de las nuevas celdas.

El R*-tree original guarda la información de todos los puntos contenidos en una celda para acceder a los datos de forma rápida. Sin embargo, esto se hace a costa de aumentar el tamaño en memoria del árbol. El tamaño máximo en memoria que ocupa el árbol implementado de esta manera crece como $n \cdot l$, donde l es la profundidad del nivel más bajo del árbol. Buscando aplicar este algoritmo a bases de datos de millones de puntos, no parece una solución aplicable si la memoria RAM es un factor limitante. Por ello, se ha decidido modificar esta implementación para que cada punto aparezca solamente una vez en la celda más pequeña a la que pertenece. Aunque esto aumente la cantidad de nodos recorridos para encontrar un punto, el tamaño en memoria es prácticamente constante, independientemente del número de niveles del árbol.

Uno de los datos que se usan a la hora de hacer cálculos de distancias es el de los límites de cada celda. Para evitar cálculos redundantes, se guardan los límites de cada celda junto con la información del padre y los hijos del nodo.

Otra estrategia para ahorrar cálculos es la implementación de un $flag$ en los nodos que contienen los puntos. En el R*-tree original, la única condición para dejar de partir una celda es llegar a tener un número de puntos por debajo de umbral. Manteniendo esta condición, se ha añadido otra condición de parada acorde con el problema. Antes de decidir partir una celda, se calcula la longitud de su diagonal principal. Si esta longitud es menor que la longitud mínima de enlace b, querrá decir que todos los puntos de la celda están conectados entre sí. Añadir este $flag$ permite hacer el cálculo de expansión del clúster rápidamente. Si a la hora de buscar nuevos puntos se encuentra una celda con este $flag$, basta con comprobar que existen al menos $minVecinos$ conexiones para añadir todos los puntos al clúster como puntos de "core". En la figura (Fig.A.3) del anexo se muestra el efecto que tiene el parámetro $minVecinos$ en la identificación de los clústeres. Otra aplicación directa de este $flag$ es que si se empieza la expansión de un clúster a partir de un punto contenido en una de estas celdas, todos los puntos de la misma se añaden al clúster en un solo paso.

La implementación inicial de este algoritmo se ha hecho usando una estructura de R*-tree. Dado que para el algoritmo DTFoF se ha escrito una implementación del KD-tree, se ha usado este también para comparar los resultados entre ambos. En la figura (Fig.A.4) del anexo se muestra la diferencia en la distribución de las celdas entre los dos tipos de árbol. Se ha hecho un bechmark usando las mismas bases de datos que para el MST, esta vez llegando hasta los 90000 puntos. Los resultados obtenidos se muestran en la siguiente figura (Fig.4).

Dado que la versión del programa que usa el KD-tree parece más rápido, se ha decidido usar este para compararse con el resto. Aunque en la implementación de los autores usaban el R*-tree, dejaban abierta la posibilidad de usar cualquier otra estructura de árbol compatible. Esto se ha comprobado verificando que el número de clústeres obtenidos con las dos versiones es el mismo para el mismo conjunto de puntos. Se ha hecho un ajuste parabólico para el tiempo de la versión del problema con un KD-tree

$$T_{\mathrm{DBSCAN}} = -0.418 + 4.47 \cdot 10^{-4} n + 2.98 \cdot 10^{-9} n^2 \ [n]. \tag{19}$$

Una opción que da este programa es la de buscar clústeres de FoF. En este tipo de clústeres cualquier punto conectado a otro se considera parte del clúster, por lo tanto, para obtener obtener el mismo mismo resultado usando DBSCAN basta con que cada punto encontrado al expandir

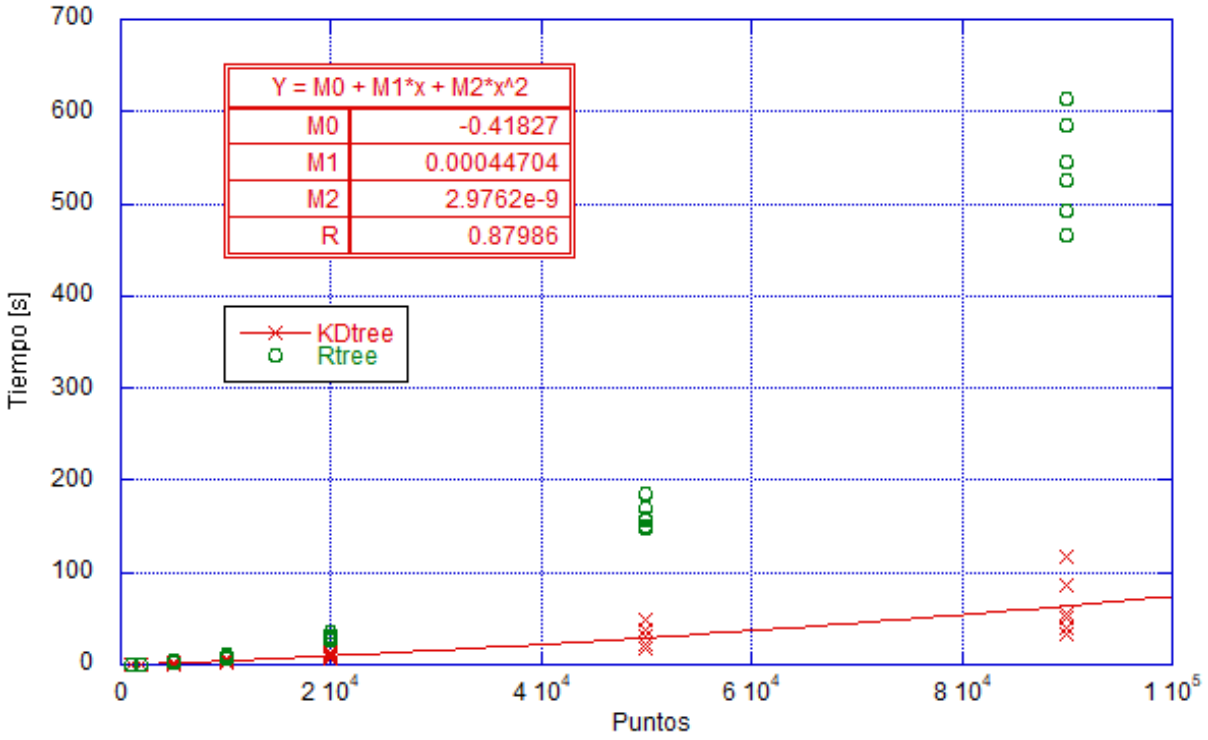

Fig. 4: Benchmarking de MST sobre distintas bases de datos y tipo de árbol usado, con un ajuste parabólico para KD-tree.

un clúster se añada como punto de core. Para ello, basta con que el parámetro de entrada *minVecinos* sea igual a 0. Para comprobar que efectivamente se obtienen los mismos clústeres, se han comparado el número de clústeres obtenidos con DBSCAN y DTFoF para un mismo parámetro b. Usando las mismas bases de datos, los resultados muestran que efectivamente el número de clústeres encontrados usando ambos algoritmos es el mismo.

3.3 DTFoF

La implementación del algoritmo DTFoF se ha hecho siguiendo el algoritmo desarrollado anteriormente en la sección 2.5.3. Se ha adaptado el algoritmo en algunos aspectos para que el programa pueda funcionar en un ordenador de sobremesa. En este sentido, además de la implementación del árbol en la que solamente se guarda cada punto una vez, se han añadido pequeñas mejoras.

La primera de ellas se aplica en la función ENUM (Alg.15), y hace uso del sistema de "flags" de los nodos del árbol explicado en la sección anterior. Si los nodos entre los que se están buscando parejas son el mismo, y además todos sus nodos están conectados, la asignación de las parejas de puntos se hace directamente sin comprobar de nuevo las distancias.

La segunda mejora es una pequeña modificación de la misma función. Para acelerar la operación de identificar cada uno de los clústeres encontrados, las parejas de puntos devueltas por ENUM se ordenan de menor a mayor. De esta manera, las raíces de los clústeres será siempre el punto con el índice menor. Este pequeño cambio hace menos costosa la operación de identificar y contar los clústeres.

Aplicando sucesivas mejoras al programa principal, se han desarrollado 3 versiones distintas

del siguiente algoritmo:

- `DTFoF_v0`: implementa el algoritmo de la referencia [18].
- `DTFoF_v1`: mejora de la versión `DTFoF_v0` que incluye la optimización de la búsqueda de parejas en los nodos totalmente conectados.
- `DTFoF_v1.1`: modificación de la versión 1 reordenando las parejas devueltas por ENUM.

En la referencia donde se desarrolla este algoritmo, se usa un KD-tree como estructura de datos para los puntos. El programa que genera el árbol se ha escrito siguiendo de forma directa lo desarrollado en la sección 2.5.3. Además de esto, dejan abierta la posibilidad de usar estructuras de árbol similares. Dado que para el algoritmo DBSCAN se había escrito ya la función para generar un R*-tree, se ha decidido aprovechar ese código para comparar ambas estructuras.

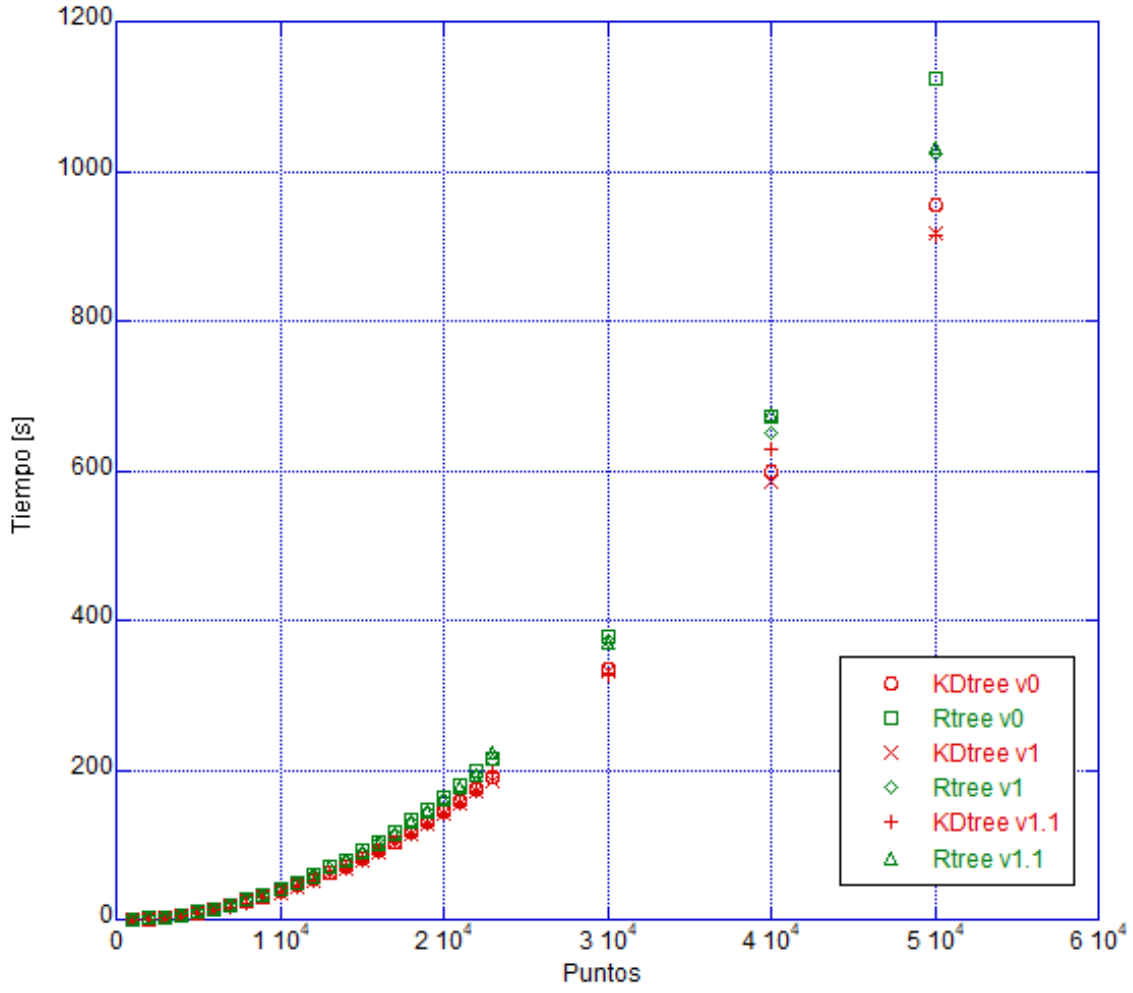

Fig. 5: Benchmarking de todas las versiones de DTFoF sobre la base de datos streams, con $b = 750$ y $nMin = 200$.

Para decidir qué versión y qué tipo de árbol da un mejor rendimiento, se ha hecho un *benchmarking* preliminar. Se ha usado una sola base de datos (*streams*) sobre el mismo número de puntos. Los resultados de esta prueba se muestran en la figura (Fig.5). Se observa inmediatamente que usar el R*-tree es más costoso que usar el KD-tree, principalmente debido a que la

generación del árbol es más costosa. Aunque la versión `DTFoF_v0` es más rápida para tamaños pequeños, a partir de 50000 puntos el coste aumenta más rápidamente que las versiones mejoradas. Por lo tanto, y dado que la pequeña mejora de `DTFoF_v1.1` parece ser ligeramente mejor para bases de datos grandes, esta es la que se ha elegido para realizar las pruebas a partir de este momento.

Para hacer una estimación del coste del programa con el número de puntos se ha hecho un *benchmarking* sobre las mismas bases de datos. Dado que el tipo de clúster de DTFoF es el de MST, se han usado los mismos parámetros de clusterización para cada base de datos. Los resultados se muestran en la figura (Fig.6). Se puede apreciar que hay una base de datos (*islands*) para la que el programa tarda más en calcular los puntos. Al tratarse de una base de datos peculiar, en la que los clústeres son pequeños y lineales, puede usarse como aproximación al peor caso. De todas formas, el coste computacional del programa parece seguir siendo cuadrático. La parábola que mejor se ajusta a los datos obtenidos de tiempo frente al tamaño del problema es

$$T_{\text{DTFoF\_v1.1}}(n) = 1.36 - 2.05 \cdot 10^{-4}\, n + 3.91 \cdot 10^{-7}\, n^2\ [s]. \tag{20}$$

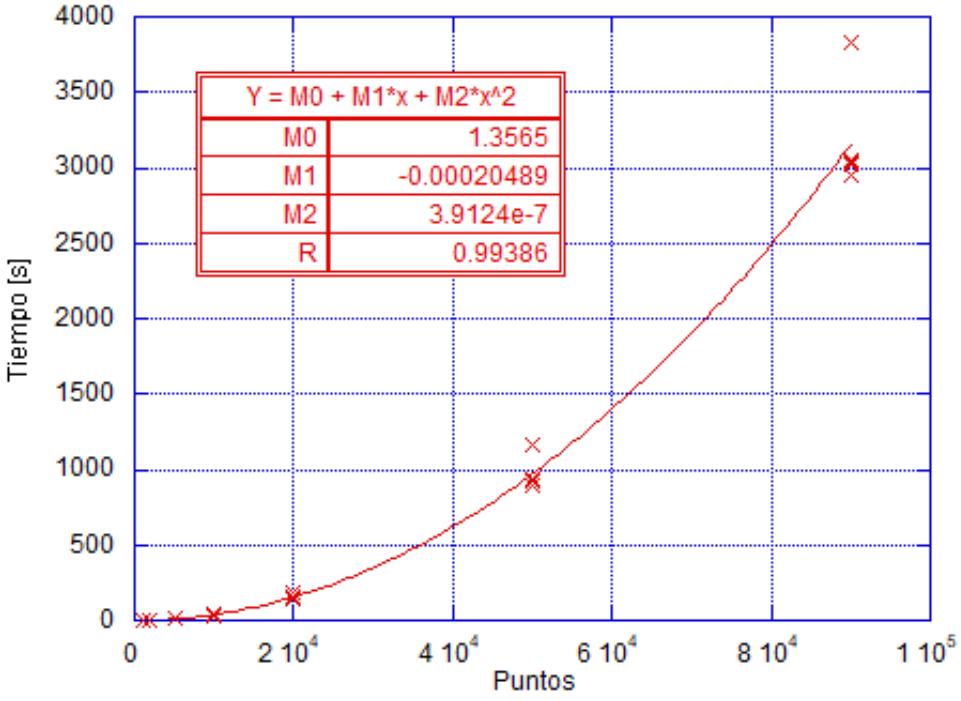

Fig. 6: Benchmarking de DTFoF sobre distintas bases de datos y ajuste parabólico.

3.4 Elección del mejor algoritmo para el cálculo de clústeres

Vistos los resultados obtenidos es necesario escoger uno de los algoritmos para lanzar la búsqueda de clústeres de galaxias.

Teniendo en cuenta el orden de magnitud del número de galaxias que se quiere analizar ($\sim 10^6$), es evidente que se tiene que descartar el algoritmo de MST, ya que su rendimiento empeora de forma drástica con el número de puntos. El simple hecho de necesitar al menos $\Omega(n^2/2)$ cálculos para obtener las distancias entre cada punto, hace que este algoritmo sea ineficiente para esta aplicación.

La elección entre usar el algoritmo DTFoF o DBSCAN implica elegir también el tipo de clúster que se buscará. Dado que los dos algoritmos aplican una definición de clúster distinta, no se pueden comparar los resultados de forma cuantitativa. La elección dependerá por lo tanto del uso que se le quiera dar a los clústeres encontrados. Sobre este problema, en la figura (Fig.A.5) del anexo se muestra la diferencia principal entre los dos algoritmos, y el problema que puede surgir al usar DTFoF en bases de datos con una gran densidad de puntos.

Algunos autores usan algoritmos de clusterización de Friends-of-Friends para reducir el número de candidatos sobre los que hacer un análisis en profundidad mucho más costoso. En [25] por ejemplo, para lanzar una búsqueda de clústeres basado en un cálculo complejo de la luminosidad, primero se realiza una pasada para encontrar clústeres de tipo FoF. Todos los clústeres encontrados, incluso los clústeres formados por solamente una pareja de galaxias, se usan para lanzar el siguiente paso. Sin embargo, al tener un solo parámetro de entrada (la distancia máxima de enlace b) es necesario hacer un ajuste fino del mismo para obtener resultados adecuados. Un valor de b pequeño hará que se encuentren una gran cantidad de clústeres de tamaño pequeño, la mayoría siendo solamente de 2 o 3 galaxias. Un parámetro b demasiado grande hará que se obtenga un solo clúster que abarque todo el universo. Este problema hace que DBSCAN parezca la mejor solución al problema, ya que el parámetro del número mínimo de vecinos permite un ajuste más bruto del parámetro b.

El rendimiento es también un factor importante. Como se ha visto, el tiempo que tarda el programa en dar un resultado crece aproximadamente con el cuadrado del tamaño del problema. Con este dato, y usando los ajustes hechos para cada programa (Eqs.18, 19, 20), se ha hecho una estimación del tiempo que tardaría en resolver un problema de un tamaño similar al que se trabajará en el siguiente capítulo:

$$n = 3 \cdot 10^6 : \begin{cases} T_{\text{MST\_v2}} & \sim 4 \cdot 10^7 s \approx 11500h, \\ T_{\text{DBSCAN}} & \sim 3 \cdot 10^4 s \approx 8h, \\ T_{\text{DTFoF\_v1.1}} & \sim 3 \cdot 10^6 s \approx 100h. \end{cases}$$

Viendo estos tiempos es evidente que la mejor opción es la de usar el programa que implementa DBSCAN. El orden de magnitud del tiempo necesario para los otros 2 es demasiado grande. Además, dado que DBSCAN permite también encontrar clústeres FoF ofrece más posibilidades que un algoritmo especializado como DTFoF.

4. Clústeres de galaxias

Uno de los objetivos de este trabajo es aplicar un algoritmo de clusterización a una base de datos de galaxias para encontrar todos los posibles clústeres y proto-clústeres. En la tesis de Yi-Kuan Chiang [26] se define un clúster como un halo de materia oscura limitado y virializado[1] en el presente ($z = 0$), con una masa mayor que 10^{14} $M_\odot$ h^{-1}. Un proto-clúster será un clúster en una época anterior ($z > 0$) que uniéndose con otros proto-clústeres formarán un clúster en el futuro. A parte de esta diferencia, en este trabajo no se hace distinción entre ambos conjuntos.

Para encontrar estos clústeres se tienen que analizar, entre otros factores, la densidad de masa, de plasma intracúmulo (ICM) y de materia oscura. Ni la materia oscura ni los halos que forma son observables de forma directa, por lo que la búsqueda de clústeres se centra en las dos primeras características. En concreto en la tesis, la búsqueda de los clústeres se centra en identificar regiones con una sobredensidad de masa, halo y número de galaxias. Estos parámetros se calculan como

$$\delta_m(x) = \frac{\rho(x) - \langle\rho\rangle}{\langle\rho\rangle},\ \delta_h(x) = \frac{n_h(x) - \langle n_h\rangle}{\langle n_h\rangle},\ \delta_{gal}(x) = \frac{n_{gal}(x) - \left\langle n_{gal}\right\rangle}{\left\langle n_{gal}\right\rangle}, \tag{21}$$

donde $n(x)$ es la densidad en un punto dado y $\langle n\rangle$ es la densidad promedio del entorno.

Además de este, existen multitud de métodos para abordar el problema. Uno de ellos consiste en lanzar simulaciones de la evolución del universo a partir de la situación en una época dada. Las simulaciones tendrán que tener en cuenta no solo la dinámica de las galaxias, sino que tendrán que dar cuenta de la expansión del universo a través de modelos cosmológicos como el ΛCDM [26, ch. 2.2.1]. Otros métodos buscan la información de la densidad de materia a través de otros datos, como pudiera ser la luminosidad de las galaxias. Este método analiza una gran cantidad de grupos pequeños de galaxias para estimar el tamaño del clúster a partir de la luminosidad y el radio aproximado de este [25].

A pesar de que estos métodos siguen los modelos cosmológicos para abordar esta tarea, tienen un problema que los hace difícilmente implementables sobre bases de datos completas. Como norma general, el orden de magnitud del número de galaxias de estas bases de datos varía desde 10^5 hasta 10^7. Lanzar este tipo de análisis multivariable sobre tantos puntos puede necesitar demasiado tiempo de cómputo y un ajuste fino de las variables de entrada igualmente costoso.

4.1 Clasificación de clústeres

Al igual que la mayoría de objetos del universo, los clústeres se pueden categorizar en varios tipos según criterios distintos. Dos de los criterios principales a la hora de clasificar los clústeres son el criterio de Abell y el de Bautz-Morgan.

En el catálogo de clústeres de George O. Abell [27] se usan varios criterios distintos para clasificar los clústeres: la riqueza o el número de galaxias del clúster, lo compacto que es el clúster, lo distante que es el clúster en relación a la magnitud de las galaxias más brillantes y la latitud respecto al plano galáctico. En el trabajo original esta última clasificación era necesaria

[1] Un sistema está virializado si la interacción gravitatoria entre sus elementos hace que este sea estable, es decir, que ninguno de sus elementos vaya a escapar en un futuro.

debido a los errores cometidos al observar estrellas, por lo que con unos instrumentos de medida mejores, esta clasificación está obsoleta.

La clasificación de Bautz-Morgan [28] no tiene en cuenta las propiedades del clúster sino las características morfológicas de las galaxias que lo forman. En concreto, según esta clasificación los clústeres se pueden identificar en 5 grupos según la morfología de la galaxia más brillante. Los clústeres tipo I contienen una galaxia supergigante de tipo cD[1], mientras que los de tipo III no contienen ninguna galaxia dominante. El resto de clústeres se clasificarán en tipos intermedios según lo cercanos que sean a cada uno de los criterios.

En este trabajo se va a usar la clasificación usada por Yi-Kuan Chiang en su tesis. Esta clasificación consiste en identificar un clúster según su tamaño y número de elementos y compararlos con 3 clústeres representativos. De mayor a menor tamaño los clústeres son Virgo, Coma y Fornax (Tab.1).

Tab. 1: Tamaño y número de galaxias de los 3 clústeres representativos en distintas épocas.

Tipo	Radio virial [MPc]			nº galaxias
	z=0	z=2 [26]	z=5 [26]	z=0
Coma	2.9 [29]	13.0	18.8	~ 4500 [29]
Virgo	1.72 [30]	9.0	13.2	1589 [31]
Fornax	0.7 [32]	6.4	9.6	590 (564 enanas) [32]

4.2 Cálculo de clústeres

Para calcular estos clústeres de galaxias se va a elegir uno de los algoritmos desarrollados anteriormente. Este algoritmo, además de dar resultados coherentes, tendrá que poder hacer los cálculos en un tiempo razonable dado el tamaño del problema. Aunque el algoritmo que se vaya a usar parta de un modelo matemático simple, los parámetros de entrada de estos algoritmos contendrán información básica acerca de los modelos usados en los otros métodos de clusterización de galaxias.

Una conclusión que se obtiene de la tesis de Yi-Kuan Chiang es que la definición del clúster se tiene que ajustar a la época del universo en la que se esté haciendo el análisis. Según más temprano sea el universo que se está observando, el número de galaxias del clúster disminuye mientras que su tamaño aumenta. Esto implica que el algoritmo que se use tiene que ser capaz de admitir fácilmente una longitud de enlace y un número de elementos por clúster variable. Esta característica la cumple adecuadamente el algoritmo DBSCAN, con tal de definir una función para calcular distancias que tenga en cuenta esta variable.

4.2.1 Base de datos

Para hacer los cálculos se ha usado la base de datos del SDSS DR16 [33]. En ella, se han accedido a los datos de *redshift*, ascensión recta y declinación de los objetos identificados como galaxias. Dado que para correr el programa se necesita conocer la posición de cada galaxia, se ha usado una función propia de la base de datos (`fCosmoDC`) para calcular la distancia comóvil

[1] Según el esquema de clasificación morfológica de Yerkes.

de cada galaxia con la tierra. Las distancias entre galaxias se pueden calcular directamente aplicando una métrica euclídea a estas posiciones. Ya que para calcular esta distancia la función necesita que los *redshift*s de los objetos sean positivos, la petición de datos hecha busca solamente las galaxias con $z > 0$. Las galaxias identificadas con un error de z demasiado grande ($z \gtrsim 0.15$) impiden diferenciar correctamente los clústeres [26, ch. 3.4], por lo que se han eliminado estas galaxias de la base de datos. La petición al servidor se ha hecho a través del servicio "CasJobs"[1] de la base de datos usando lenguaje SQL, tal y como se muestra en (Qry.1). El resultado es una base de datos con 3012775 galaxias distintas. A esta base de datos se le ha añadido de manera manual la Vía Láctea, colocándola en el origen de coordenadas.

Query 1 Instrucción SQL para obtener los datos del SDSS a través de CasJobs

```
SELECT DISTINCT ObjID
    ObjID, ra, dec, z, dbo.fCosmoDC(z,D,D,D,D,D)        ▹ D=DEFAULT
FROM
    SpecPhotoAll
WHERE
    z > 0 AND type=3 AND zErr<0.15                      ▹ type=3 son las galaxias
```

Una aproximación que se va a aplicar en este trabajo es la de suponer que todas las galaxias son iguales entre sí. En concreto se va a suponer que todas las galaxias tienen la misma masa. Un cálculo más exacto de los posibles clústeres tiene que tener en cuenta la masa de cada una de las galaxias, al ser este dato una importante herramienta para su identificación.

4.2.2 Elección de parámetros de clusterización

La elección correcta de los parámetros es una tarea importante a la hora de resolver este problema. Como ya se ha adelantado, el número de miembros y el tamaño de los clústeres dependen de la época en la que se estén mirando.

Para obtener la distancia mínima de enlace b para DBSCAN, se va a calcular la distancia media a la que se encuentra la galaxia más cercana a otra para distintos rangos de z (Fig.7). El cálculo se ha hecho asignando a cada punto obtenido la media de las distancias de las galaxias en cierto margen de z: ± 0.025 para la región de $z < 1$ y ± 0.25 para el resto. Estas franjas se han elegido para tener una muestra aproximadamente uniforme en cada una de ellas, aunque a partir de $z = 2.5$ cada punto tiene solamente del orden de 1000 datos de galaxias. Debido a la poca densidad de puntos disponibles, no se puede asegurar que los datos obtenidos sean representativos. Teniendo en cuenta que se esperaba una curva monótonamente creciente, se ha decidido ajustar los datos usando una interpolación de spline cúbica.

En el apartado 3.1 de la tesis, en concreto las figuras 3.1, 3.2 y la tabla 3.4, se muestran las relaciones entre la sobredensidad de galaxias y el *redshift* para los distintos tipos de clústeres y *windowing*. Usando esas relaciones y comparando los resultados de algunos clústeres previamente clasificados, se dan unas estimaciones para los valores de sobredensidad de galaxias en cada tipo de clúster con un *windowing* de 13MPc. Los valores de δ_{gal} usados para los clústeres tipo Coma, Virgo y Fronax son 2.8, 1.6 y 1.0 respectivamente [26, p. 54].

A partir de estos datos y el valor de b, se obtendrá el parámetro $minVecinos$. Para ello se va a suponer que los clústeres se encuentran superpuestos sobre una distribución aleatoria de

[1] Accesible desde http://skyserver.sdss.org/CasJobs/

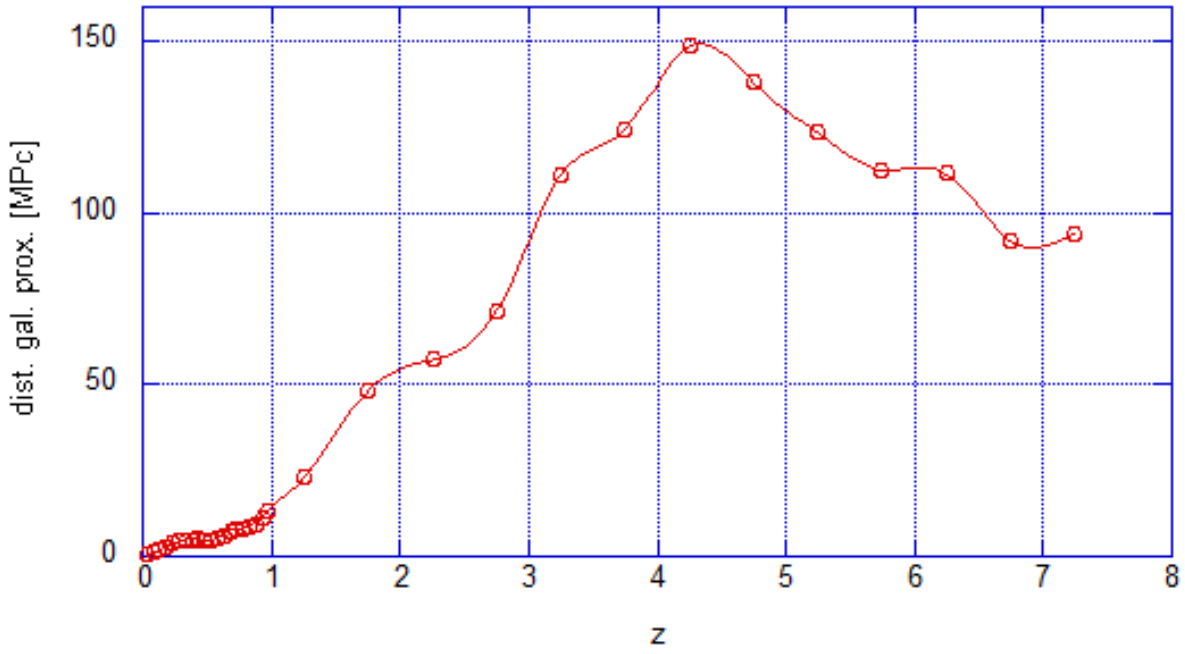

Fig. 7: Distancia media entre galaxias más cercanas en función del redshift e interpolación de spline cúbica.

puntos. La 2PCF (two-point correlation function) da cuenta de la diferencia de densidad de probabilidad de encontrar una galaxia a una distancia dada de otra respecto a la distribución de Poisson [34, Sec.V.B]. Así, la probabilidad de encontrar una galaxia a un distancia entre r y $r + dr$ es

$$dP(r) = \left\langle n_{gal} \right\rangle (1 + \xi(r))dV, \tag{22}$$

con $\xi(r)$ la 2PCF. En [35] se da el siguiente ajuste para la 2PCF

$$\xi(r) = \left(\frac{5.06 \pm 0.12\ [MPc]}{r} \right)^{1.862 \pm 0.034} . \tag{23}$$

Se puede estimar el número medio de galaxias que se encuentran a una distancia < b de otra integrando la ecuación (Eq.22)

$$\overline{n_{vec}} = \int_0^b \left\langle n_{gal} \right\rangle (1 + \xi(r))4\pi r^2 dr = \left\langle n_{gal} \right\rangle \left(\frac{4\pi\ b^3}{3} + 226.045\ b^{1.138} \right). \tag{24}$$

A partir de este número medio de vecinos por galaxia, e imponiendo la condición de que en un clúster exista una sobredensidad de galaxias, el parámetro de *minVecinos* de DBSCAN se puede obtener como

$$minVecinos = \left\langle n_{gal} \right\rangle \delta_{gal} \left(\frac{4\pi\ b^3}{3} + 226.045\ b^{1.138} \right). \tag{25}$$

La densidad media de galaxias en un entorno depende fuertemente del tipo de observaciones que se hayan hecho. Dadas las limitaciones de los observatorios, en la base de datos del SDSS solo se tiene información de algunas regiones del cielo. Además, no todas las regiones se han analizado con los mismos instrumentos, por lo que la densidad de datos puede variar. Esto hace

que para calcular de forma precisa la densidad de galaxias de un entorno haya que hacerlo punto por punto. Dado que para lanzar el programa se usa una estructura de árbol, se puede acceder rápidamente a la información del entorno de cada punto. En concreto, para hacer el cálculo de la densidad media, se usará la información de las celdas los niveles superiores a la que pertenece la galaxia. En la tesis se trabaja con un un *windowing* máximo de 25MPc para calcular las sobredensidades, por lo que para calcular la densidad media en un punto habrá que escrutar un entorno amplio. Se ha elegido hacer este cálculo sobre una esfera de radio 10 veces mayor que este valor (250MPc), dando como resultado una muestra de tamaño similar a la descrita en el capítulo 2.2.1 de la tesis.

El cálculo de la densidad en el entorno de cada punto es demasiado costoso. Para poder realizar los cálculos en un tiempo razonable, se ha decidido calcular la densidad en el entorno solamente del punto sobre el que se empieza a expandir cada punto, fijando también el resto de parámetros. Esta concesión agrava el problema de encontrar clústeres de forma eficaz en los bordes de la base de datos. Si el primer punto de la expansión está en una región cercana a un borde, el cluster se sobreexpandirá hacia regiones con mayor densidad de datos.

A pesar de ello, los cálculos hechos se tomarán con todos los datos disponibles para minimizar este efecto lo máximo posible sobre las regiones en las que sí se tiene una densidad de datos aproximadamente constante. Para evitar la expansión incorrecta de un clúster a partir de estos bordes, se ha limitado el número de veces en la que se buscan nuevos candidatos (paso 6 Alg.7) a 10.

4.3 Resultados obtenidos

Usando los parámetros obtenidos en el apartado anterior (Sec.4.2.2) se han lanzado la búsqueda de los 3 tipos de clústeres sobre la base de datos completa (Sec.4.2.1). Los resultados para cada tipo de clúster se muestran en las figuras (Figs.A.6, A.7 y A.8) del anexo. Se han obtenido 5449 clústeres tipo COMA, 7914 tipo VIRGO y 9631 tipo FORNAX, con un número medio de galaxias por clúster de 74, 59 y 53 respectivamente.

Es evidente que el número medio de galaxias obtenidas es muy inferior al esperado según los valores de la tabla (Tab1). Una razón de esto puede ser que estos clústeres se han observado con mayor detalle, detectando miembros de los cuales la base de datos usada no tiene información.

Una de las conclusiones a las que se llega viendo la distribución de los clústeres es que, pese al intento de minimizar este problema, se detectan clústeres demasiado grandes en el borde de la base de datos. Este problema se ve acrecentado sobre el eje z el sistema de coordenadas J2000. Se observa claramente que sobre este eje se detectan más clústeres que en el resto del espacio. Dadas las limitaciones de la base de datos, en este eje se tiene un borde de la base de datos dada por los ángulos muertos de ascensión recta.

Algunos de los clústeres obtenidos están formados por un número demasiado pequeño de galaxias, teniendo incluso clústeres con 1 solo miembro. Esto es debido a que dados unos parámetros distintos, una galaxia puede formar o no un clúster. Por el método de clusterización usado, el orden en el que se hace la búsqueda de nuevos clústeres afecta al resultado. Parece que estos clústeres pequeños aparecen principalmente en zonas adyacentes a otros clústeres previamente identificados. Este nuevo clúster, al no poder sobreescribir la pertenencia de una galaxia a un cluster, no puede seguir expandiéndose. Sin embargo, y dado que son producto de una sobredensidad mayor de galaxias, se han mantenido como clústeres válidos. Un cálculo

punto a punto de los parámetros de clusterización resolvería este problema.

4.4 Comparación de resultados

Una práctica común de los catálogos de objetos astronómicos es comparar los resultados obtenidos con algún trabajo previo. Para este trabajo se ha decidido usar la base de datos SPIDERS del SDSS [36]. Esta base de datos contiene información, entre otros, de 2740 clústeres de galaxias detectados a partir de observaciones en el espectro de los rayos X. Una ventaja importante que presenta SPIDERS es que se ha generado a partir de la misma base de datos SDSS DR16 (Sec.4.2.1). Esto hace que no exista ningún clúster en una región no cubierta por alguna de las dos bases de datos.

Una peculiaridad de la base de datos SPIDERS es que los clústeres no se dan a partir de las galaxias que lo forman, sino a partir del centro del clúster. Para poder afirmar si se ha detectado uno de estos clústeres, se ha hecho una aproximación de la extensión de los clústeres identificados. Se han calculado los límites de un prisma rectangular que contiene todas las galaxias de un clúster. Si alguno de los puntos de SPIDERS cae dentro de estos prismas, se tendrá una coincidencia.

Se ha repetido el cálculo descrito para cada una de las bases de datos generadas. El número de coincidencias entre las bases de datos es de 928 clústeres para los 3 tipos distintos, es decir, se han identificado el 33.9% de los clústeres obtenidos por SPIDERS.

5. Resultados y trabajo futuro

En este trabajo se han estudiado diversos algoritmos de clusterización. Con el objetivo de aplicarlos a la identificación de clústeres de galaxias, se han implementado 3 de ellos: MST, DTFoF y DBSCAN. Tras discutir cuál de ellos cumple con los requisitos necesarios para este trabajo, el algoritmo escogido ha sido DBSCAN. Aplicándolo a una base de datos real, se han conseguido identificar clústeres de galaxias de distintos tipos según su densidad.

Hay que recordar que los clústeres obtenidos son solamente candidatos. Para poder verificar que un candidato cumple las condiciones como para clasificarlo como clúster, es necesario hacer un análisis posterior. El método de detección desarrollado en este trabajo puede servir para acelerar el proceso de selección de candidatos. La variedad y forma en la que se ha implementado la entrada de los parámetros de clusterización permite un ajuste fino de estos, lo que lo hace más versátil que otros algoritmos más simples. La entrada de parámetros variables podría ser de utilidad en otro tipo de problemas en los que no se pueden usar parámetros de clusterización constantes.

Los criterios usados para definir un clúster de galaxias en este trabajo, aunque basado en trabajos previos, no permiten afirmar con certeza que los clústeres encontrados sean reales. Tal y como se ha comentado en el capítulo 4, para hacer esta identificación adecuadamente es necesario analizar un amplio rango de medidas, las cuales no se han tenido en cuenta en este trabajo. Encontrar clústeres de galaxias propiamente es una tarea a medio camino entre el análisis computacional y la identificación manual observacional. Este TFG se ha centrado solamente en el aspecto computacional, buscando una aplicación práctica de los algoritmos de clusterización. Determinar la validez de los clústeres identificados requeriría un trabajo más propio de la cosmología, lo cual está fuera del alcance de este trabajo.

Para tener un catálogo completo de candidatos de clústeres en una región del universo sería necesario tener una densidad de datos elevada, tanto en la región objetivo como en un entorno suficientemente grande. Las bases de datos disponibles no suelen ser completas. Como se ha visto con la base de datos usada, hay algunas regiones del espacio con mayor cantidad de datos que otras. Para evitar este problema, se deberían combinar distintas bases de datos. Con esto, se conseguiría tener una densidad de datos más uniforme, además de poder eliminar detecciones de galaxias espurias.

El algoritmo de clusterización desarrollado en este trabajo tiene varios aspectos con margen de mejora. Una de las mayores aproximaciones que se han hecho para calcular los parámetros de clusterización ha sido la de suponer que la masa de todas las galaxias es idéntica. Añadir esta variable al algoritmo podría mejorar la identificación de clústeres, limpiando de los resultados el "ruido" introducido por las galaxias demasiado pequeñas.

En este sentido, se podrían añadir datos espectográficos de las galaxias. Las galaxias cercanas con un espectro similar tienen una probabilidad mayor de haberse formado en un mismo clúster. Además, se podrían detectar las galaxias formadas fuera del clúster, y que sin formar parte de este, estén lo suficientemente cerca como para que se le hayan asignado. Con el mismo objetivo, se podrían añadir observaciones en el espectro de los rayos X. Usando mediciones en este espectro, es posible detectar el polvo del intraclúster. Comparando la distribución de este polvo con la distribución de galaxias, se podrían desechar los clústeres formados por agrupaciones aleatorias de galaxias.

Uno de los problemas que tienen las bases de datos, que no se ha tenido suficientemente en cuenta, es el error en las mediciones, en concreto las de *redshift*. Tal y como se ha comentado en la sección 4.2.1, cuanto mayor es el error en el *redshift* menor es el margen de sobredensidad con el que se pueden detectar los clústeres. Para afinar aún más los valores calculados de la sobredensidad de galaxias, se podría tener en cuenta la mediana de este error en el entorno de cada punto.

La realización de este trabajo ha servido principalmente para reafirmar la capacidad de estudiar algoritmos de computación e implementarlos de manera eficaz. Se han aplicado competencias adquiridas en todas las asignaturas en las que se han trabajado métodos matemáticos y técnicas computacionales, lo que ha permitido implementar de forma fácil y eficiente los algoritmos descritos en el trabajo. Se confirma además la capacidad de adaptación a un lenguaje de programación nuevo con facilidad, en este caso Matlab, añadiéndolo así a la lista de los lenguajes usados a lo largo de la doble titulación.

Se han estudiado además algunos conceptos propios de la cosmología desde un punto de partida en el que no tenía prácticamente ningún conocimiento previo. Aún así, se ha logrado analizar y extraer de manera efectiva la información necesaria para realizar el trabajo. Los conocimientos adquiridos pueden servir de entrada a este ámbito de la física, en el que todavía quedan muchas puertas abiertas a la investigación futura.

Bibliografía

[1] D. N. Spergel *et al.*, "First Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Determination of Cosmological Parameters," Astrophys. J. Suppl. Ser., vol. 148, no. 1, 2003, pp. 175-194.

[2] C. J. Conselice *et al.*, "The Evolution of Galaxy Number Density at Z < 8 and its Implications," Astrophys. J., vol. 830, no. 2, 2016, pp. 83.

[3] R. Diestel, "Graph Theory", 5th ed., Springer, 2017.

[4] A. Kao, and S. R. Poteet, "Natural Language Processing and Text Mining," Springer, 2007, ch. 1, pp. 1-7.

[5] S. Battiato *et al.*, "A cluster-Based Boosting Strategy for Red Eye Removal," in Computational Intelligence in Image Processing. A. Chatterjee, and P. Siarry, Springer, 2013, ch. 12, pp. 217-249.

[6] M. Ester *et al.*, "A Density-Based Algorithm for Discovering clusters in Large Spatial Databases with Noise," in Proc. 2nd Int. Conf. Knowl. Discovery and Data Mining (KDD-96), 1996, pp. 226-231.

[7] A. Vathy-Fogarassy, and J. Abonyi, "Graph-Based clustering and Data Visualization Algorithms", Springer, 2013.

[8] G. Hamerly, and J. Drake, "Accelerating Lloyd's Algorithm for k-Means clustering," in Partitional clustering Algorithms. M. E. Celebi, Springer, 2015, ch. 2, pp. 41-78.

[9] L. Kaufman, and P.J. Rousueeuw, "Partitioning Around Medoids (Program PAM)," in Finding Groups in Data: an Introduction to cluster Analysis. John Wiley & Sons, 1990, ch. 3, pp. 68-125.

[10] R. T. Ng, and J. Han, "Efficient and Effective clustering Methods for Spatial Data Mining," in Proc. 20th VLDB, 1994, pp. 144-155.

[11] L. Vendramin, M. C. Naldi and R. J. G. B. Campello, "Fuzzy Clustering Algorithms and Validity Indices for Distributed Data," in Partitional clustering Algorithms. M. E. Celebi, Springer, 2015, ch. 5, pp. 147-192.

[12] J. M. Marin, K. Mengersen, and C. P. Robert, "Bayesian Modelling and Inference on Mixtures of Distributions," in Handbook of Statistics Series: Bayesian Thinking, Modeling and Computation, vol. 25, ed. 1. North Holland, 2011, ch. 16.

[13] A. Vathy-Fogarassy, A. Kiss, and J. Abonyi, "Hybrid Minimal Spanning Tree and Mixture of Gaussians Based clustering Algorithm," in FoIKS 2006, Feb. 2006, pp. 313–330.

[14] N. Beckmann *et al.*, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," in Proc. ACMS IGMOD Int. Conf. Manage. Data, 1990, pp. 322-331.

[15] A. Hinneburg, and D. A. Keim, "An Efficient Approach to clustering in Large Multimedia Databases with Noise," in Proc. 4th Int. Conf. Knowl. Discovery and Data Mining (KDD-98), 1998, pp. 58-65.

[16] W. Wang, J. Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," in Proc. 23rd VLDB, 1997, pp. 186-195.

[17] J.B. Kruskal Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," in Proc. Am. Math. Soc., vol. 7, no. 1, 1956, pp. 48-50.

[18] Y. Feng, and C. Modi, "A fast algorithm for identifying friends-of-friends halos," in Astron. Comput., vol. 20, 2017, pp. 44-51.

[19] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," in ACM Trans. Math. Softw., vol. 3, no. 3, sep. 1977, pp. 209-226.

[20] M. E. Celebi, and H. A. Kingravi, "Linear, Deterministic, and Order-Invariant Initialization Methods for the K-Means clustering Algorithm," in Partitional clustering Algorithms. M. E. Celebi, Springer, 2015, ch. 3, pp. 79-98.

[21] C. Braune, S. Besecke, and R. Kruse, "Density Based clustering: Alternatives to DBSCAN," in Partitional clustering Algorithms. M. E. Celebi, Springer, 2015, ch. 6, pp. 193-214.

[22] M. Stonebraker *et al.*, "The SEQUOIA 2000 Storage Benchmark," Proc. ACMS IGMOD Int. Conf. Manage. Data, 1993, pp. 2-11. Accessed: 7/4/2020. http://s2k-ftp.cs.berkeley.edu/sequoia/benchmark/

[23] S. Sieranoja and P. Fränti, "Fast and general density peaks clustering," in Pattern Recognit. Lett., vol. 128, 2019, pp. 551-558.

[24] P. Fränti, and S. Sieranoja, "K-means properties on six clustering benchmark datasets," in Appl. Intell., vol. 48, no. 12, 2018, pp. 4743-4759. Accessed: 8/4/2020. http://cs.joensuu.fi/sipu/datasets/

[25] X. Yang, H.-J. Mo, F. C. van den Bosch, and Y.P. Jing, "A Halo-Based Galaxy Group Finder: Calibration and Application to the 2dFGRS," in Mon. Not. R. Astron. Soc., vol. 356, 2005, pp. 1293-1307.

[26] Y. K. Chiang, "Galaxy Proto-clusters as an Interface between Structure, cluster, and Galaxy Formation," Ph.D. dissertation, Univ. Texas, Austin, 2016.

[27] G. O. Abell, "The Distribution of Rich clusters of Galaxies," in Astrophys. J. Suppl., vol.3, 1958, pp. 211-288.

[28] L. P. Bautz, and W. W. Morgan, "On the Classification of the Forms of clusters of Galaxies," in Astrophys. J., vol. 162, 1970, pp. L149-L153.

[29] D. Hammer *et al.*, "Deep Galex Observations of the Coma cluster: Source Catalog and Galaxy Counts," in in Astrophys. J. Suppl., vol. 190, no. 1, 2010, pp. 43–57.

[30] G. L. Hoffman, B. M. Lewis, and E. E. Salpeter, "The Large-Scale Distribution of Late-Type Galaxies Between Virgo and The Great Wall," in Astrophys. J., vol. 441, 1995, pp. 28-50.

[31] S. Kim *et al.*, "The Extended Virgo cluster Catalog," in Astrophys. J. Suppl., vol. 215, no. 2, 2014, pp. 22-51.

[32] A. Venhola *et al.*, "The Fornax Deep Survey with the VST: IV. A size and magnitude limited catalog of dwarf galaxies in the area of the Fornax cluster," in Astron. Astrophys., vol. 620, 2018, pp. A165:1-31.

[33] R. Ahumada *et al.*, "The Sixteenth Data Release of the Sloan Digital Sky Surveys: First Release from the APOGEE-2 Southern Survey and Full Release of eBOSS Spectra," 2019, arXiv:1912.02905. Accessed: 4/6/2020. https://skyserver.sdss.org/

[34] B. Jones *et al.*, "Scaling Laws in the Distribution of Galaxies,"in Rev. Modern Phys., vol. 76, no. 4, 2005, pp. 1211-1266.

[35] Y. P. Jing, H. J. Mo, and G. Görner, "Spatial correlation function and pairwise velocity dispersion of galaxies: CDM models versus the Las Campanas Survey," Astrophys. J., vol. 494, no. 1, 1998, pp. 1-12.

[36] N. Clerc *et al.*, "SPIDERS: the spectroscopic follow-up of X-ray-selected clusters of galaxies in SDSS-IV," Mon. Not. Roy. Astron. Soc., vol. 463, iss. 4, pp. 4490–4515. Accessed: 2/6/2020. https://www.sdss.org/dr16/data_access/value-added-catalogs/?vac_id=spiders-x-ray-galaxy-cluster-catalogue-for-dr16/