UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DELFIN++: NUEVA VERSIÓN DE ALGORITMO DE BÚSQUEDA DE VACÍOS COSMOLÓGICOS EN 3D

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

VALERIA LORETO GUIDOTTI CONTRERAS

PROFESORA GUÍA:
NANCY HITSCHFELD KAHLER

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
DANIEL PEROVICH GEROSA

Este trabajo ha sido parcialmente financiado por Proyecto Fondecyt N°1181506 y Proyecto Fondecyt N°1211484

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN
POR: VALERIA LORETO GUIDOTTI CONTRERAS
FECHA: 2021
PROF. GUÍA: NANCY HITSCHFELD KAHLER

DELFIN++: NUEVA VERSIÓN DE ALGORITMO DE BÚSQUEDA DE VACÍOS COSMOLÓGICOS EN 3D

Los vacíos cósmicos son regiones del espacio cuya densidad es significativamente menor a la media, y contienen muy pocas o ninguna galaxia. Descubiertos en los primeros catálogos de galaxias en la década de 1970, los vacíos se han convertido en estructuras fundamentales para comprender y caracterizar la estructura a gran escala del universo. Debido a su importancia, en las últimas décadas ha aumentado la cantidad de información, herramientas y algoritmos disponibles para la detección y estudio de los vacíos.

El algoritmo DELFIN (**Del**aunay Edge Void **Fin**der), implementado para 2 y 3 dimensiones, se desarrolló como una propuesta para la búsqueda de vacíos a partir de la teselación de Delaunay de galaxias, en donde, a partir de aristas localmente más largas, se construyen los polígonos más grande posibles para caracterizarlos como vacíos. En este trabajo se presenta una nueva versión de DELFIN en 3 dimensiones, basada en el trabajo de R. Alonso (2016), la cual presenta problemas como la fragmentación de vacíos en dos o más poliedros y falta de información sobre los resultados, como la densidad y las galaxias que componen un vacío.

Para desarrollar DELFIN++ como una extensión de DELFIN, se evalúa la calidad, extensibilidad y robustez de su implementación. En DELFIN++ se propone trabajar con la densidad de los vértices y aristas de la teselación, y generar poliedros -que representan a los vacíos- a partir de la agrupación de tetraedros con elementos de baja densidad. En esta extensión, se permite identificar información sobre los vacíos como sus galaxias, su densidad interna, y descriptores de la distribución de las partículas como el tensor de inercia y la elipticidad.

DELFIN++ se separa de la construcción del poliedro más grande posible a partir de las aristas más largas, pero se continúa el trabajo con las teselaciones de Delaunay. Gracias a esto, se genera una implementación eficiente del algoritmo, la cual completa su ejecución en tiempo $O(n \log n)$, que además es extensible y sigue patrones de diseño orientado a objetos. El algoritmo se evalúa con datos generados artificialmente y con datos observacionales, comparando los resultados con un catálogo de vacíos obtenidos por un algoritmo que trabaja sobre campos de densidad (Sutter, 2015). Con los datos artificiales se obtiene una mejora en los resultados respecto a DELFIN, mientras que con datos observacionales se encuentran vacíos en posiciones similares a los reportados por el catálogo de referencia, y que tienden a ser más pequeños que estos últimos.

El algoritmo y su investigación asociada pueden verse beneficiados por algunas extensiones, por lo que, para finalizar, se incluyen propuestas para mejorar la implementación y evaluar de manera más exhaustiva los resultados obtenidos por el algoritmo.

Kashin

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Los *vacíos cosmológicos* son grandes volúmenes en el espacio donde la densidad de galaxias es significativamente menor a la media. Se encuentran rodeados de paredes, filamentos y *clusters* de galaxias. Desde su descubrimiento[14, 17] se ha hecho bastante claro que son el principal elemento de la estructura a gran escala (LSS, por sus siglas en inglés) del universo. Debido a su tamaño, se ha propuesto que los vacíos son estructuras formadas al principio de la historia del universo, y en la expansión de los vacíos, la materia encontrada entre ellos se comprime, formándose las paredes y filamentos en sus fronteras, logrando una distribución de materia en la que existen regiones de baja densidad predichas por el actual modelo cosmológico aceptado *Cold Dark Matter* (CDM) con constante cosmológica (ΛCDM).

El rol de los vacíos ha sido considerado como esencial para la compresión de la organización de la distribución de materia cósmica[29]. Existen tres motivos principales para llevar a cabo el estudio de los vacíos cosmológicos. Primero, ayuda a comprender la formación y dinámica de la estructura a gran escala del universo, sirviendo como información empírica para contrastarla con predicciones teóricas. En segundo lugar, los vacíos contienen una gran cantidad de información sobre parámetros cosmológicos globales; el estudio de sus formas, tamaños, flujos y distorsiones en el corrimiento al rojo aporta información acerca de estos parámetros y sobre la naturaleza de la energía oscura. Finalmente, provee un ambiente más limpio y aislado para el estudio de la formación y evolución de galaxias, junto con la influencia del ambiente cósmico en este proceso y en la morfología galáctica.

En las últimas décadas ha existido un aumento en la cantidad de información disponible sobre galaxias, en particular obtenida por sondeos de corrimiento al rojo de galaxias. El *2dF Galaxy Redshift Survey* (2dFGRS) es un sondeo de este tipo llevado a cabo entre el 1997 y 2002, publicado en junio de 2003. El año 2000 comenzó el proyecto *Sloan Digital Sky Survey* (SDSS), dedicado a investigar el espacio con imágenes en el espectro visible y de corrimiento al rojo, creando mapas 3-dimensionales del universo. La última liberación de datos de SDSS es la número 16 (SDSS Data Release 16), publicada en diciembre de 2019, corresponde a la cuarta publicación de la cuarta fase de *Sloan Digital Sky Survey* (SDSS-IV) y contiene observaciones hasta Agosto de 2018.

El aumento de datos entregados por estos sondeos ha permitido estudiar los vacíos metó-

dicamente, algo que previamente no era posible porque los vacíos utilizan una gran fracción del espacio, lo cual es difícil de representar con poca información. Para estos estudios metódicos se han desarrollado distintos algoritmos de búsqueda de vacíos. Estos algoritmos reciben una sección de un sondeo y generan un catálogo con un conjunto de vacíos presentes en la sección. A partir de los catálogos los investigadores recuperan propiedades de los vacíos y pueden comparar los diversos algoritmos, ya sea por los catálogos producidos o mediante la búsqueda de vacíos con los mismos datos de entrada.

Se han desarrollados varios algoritmos que usan distinta información base, pero todos buscan entregar los mismos resultados. Entre los tipos de información se encuentran materia oscura, campos de densidad de materia oscura[10, 23, 25], galaxias[12, 16], halos o una combinación de estos. Además de la información base, difieren en las geometrías utilizadas en el método de búsqueda; regiones vacías en la distribución de galaxias[12, 16], regiones convexas vacías en un conjunto de puntos, construcción de esferas a partir de mínimos de densidad, conexión de celdas con baja densidad en una grilla, regiones irregulares con baja densidad alrededor de mínimos locales de densidad[10], basados en transformación divisoria[23, 25], entre otros. Esto genera diversos tipos de vacíos, algunos cúbicos, otros como uniones de esferas y otros como uniones de poliedros.

Uno de los algoritmos de búsqueda de vacíos es DELFIN[2, 15] (DELaunay edge void FINder), el cual detecta vacíos mediante la construcción de un campo de densidades a partir de una teselación de Delaunay[1], donde los puntos (que representan galaxias o partículas de materia oscura) se conectan a sus vecinos naturales. Comenzando en arcos localmente más largos en la triangulación, DELFIN construye polígonos vacíos o casi vacíos más grande posible a su alrededor.

Este algoritmo está implementado en 2D y 3D, y presenta problemas de fragmentación al momento de identificar vacíos pues exige que estos estén libre de puntos en su interior. Como resultado genera una cantidad mucho mayor de vacíos y más pequeños que los realmente existentes. Esto fue tratado en el trabajo de Alonso, R.[2], donde se agregan métodos para decidir si un par de vacíos deben ser unidos o mantenerse separados, pero no logra unir todos los vacíos fragmentados por la estrategia inicial.

En este trabajo se busca analizar el *software* producido por Alonso para evaluar su calidad y escalabilidad. También se busca incluir algunas mejoras propuestas para DELFIN en 3D, para lograr resultados más precisos tanto en la morfología de cada vacío como en la cantidad de vacíos obtenidos. Además se pretende mejorar la presentación de los resultados del algoritmo, entregando información relevante de los vacíos de un modo más comprensible, debido a que la entregada actualmente no es de utilidad para los usuarios finales (astrónomos).

[1]Se considera una triangulación óptima, en el sentido que cada punto del conjunto de entrada tendrá aristas que forman triángulos en 2D o tetraedros en 3D, que lo unen con sus puntos más cercanos. Los largos de las aristas de los triángulos o tetraedros aportan información sobre la densidad local.

1.1. Objetivos

Objetivo General

El objetivo de este trabajo de título es generar y validar una nueva versión del algoritmo identificador de vacíos cosmológicos DELFIN en 3-dimensiones, que mejore su robustez en el reconocimiento de vacíos a partir de la información sobre la densidad local de los elementos geométricos de la teselación de Delaunay.

Objetivos Específicos

1. Analizar la calidad del *software* usando métricas estándares para *software* orientado a objetos.

2. Crear tests unitarios para probar las funcionalidades existentes cada vez que el *software* sea ampliado con otras nuevas.

3. Incluir un constructor de vacíos que imponga condiciones de unión de tetraedros según métricas de densidad por vértice y/o arco en comparación con la densidad promedio de la muestra.

4. Comparar los resultados obtenidos por el nuevo constructor de vacíos con los obtenidos originalmente por DELFIN.

5. Agregar nueva información al *output* entregado por el algoritmo, tal como la densidad promedio de la muestra, la densidad de las galaxias y la identificación de las galaxias que componen los bordes de los vacíos.

6. Mejorar la identificación y almacenamiento de vacíos que se encuentran en el borde de la muestra de datos.

7. Validar el algoritmo con datos reales obtenidos de sondeos astronómicos para los cuales se conocen los vacíos existentes.

8. Generar acceso al algoritmo y a los resultados obtenidos por el mismo sobre distintos sondeos astronómicos.

9. Realizar un análisis de escalabilidad de DELFIN y DELFIN++ usando datos artificiales.

1.2. Metodología

En este trabajo se lleva a cabo la extensión de DELFIN con nuevas métricas que permiten la identificación de vacíos. Todo el trabajo se sigue basando en la teselación de Delaunay de un conjunto de puntos que representan galaxias. Los pasos principales de este trabajo son:

1. Revisar el algoritmo actual, análisis del código fuente y análisis de escalabilidad.

2. Ampliar la cobertura de los tests unitarios sobre la implementación actual, además de

preparar test para las extensiones.

3. Diseñar e implementar la integración de métricas y constructores de vacíos en el flujo y clases del software.

4. Evaluar resultados de los nuevos constructores sobre datos artificiales y comparar estos resultados con los obtenidos por la versión actual de DELFIN.

5. Realizar análisis de escalabilidad del software en memoria y tiempo utilizando datos artificiales.

6. Evaluar resultados sobre datos de catálogo y comparar con resultados de otros algoritmos sobre estos mismos datos.

7. Generar acceso a los resultados obtenidos sobre los datos de catálogo, además de disponer el código fuente para uso público.

Durante todo el proceso se utilizarán herramientas que permitan trabajar con código de C++, en particular aquellas que faciliten el proceso de compilación, test unitarios y análisis de código. Para las validaciones se utilizarán datos artificiales que permitan evaluar casos particulares como la existencia de vacíos en el borde de la muestra de datos. Se utilizarán datos del catálogo de SDSS para comparar los resultados con otros algoritmos.

1.3. Descripción general de la solución

Para llevar a cabo las extensiones de software, se escoge implementar distintas métricas de densidad para distintos elementos geométricos de la teselación de Delaunay. Para los puntos de la teselación, se implementan dos métricas: densidad respecto al volumen de los tetraedros a los que pertenece el punto, y densidad respecto a la longitud de las aristas que el punto delimita. Para las aristas de la teselación también se implementan dos métricas de densidad: densidad respecto al volumen de los tetraedros a los que pertenece la arista, y densidad respecto al volumen de los mismos tetraedros y longitud del arco.

A partir de estas métricas se definen dos tipos de identificadores de vacíos: basados en densidad de puntos y basado en densidad de arcos. A diferencia de DELFIN, los vacíos no se expanden por los tetraedros vecinos con elementos menos densos buscando el polígono más grande posible, sino que los elementos se evalúan en orden de densidad creciente, y los tetraedros que comparten el elemento correspondiente se unen para generar vacíos, ya sea uniendo los tetraedros a vacíos pre-existentes o creando un nuevo vacío.

Además del algoritmo, se integran métodos que permiten entregar información sobre la forma de los vacíos, como su centro de masas, elipticidad, tensor de inercia junto con sus valores y vectores propios. Otros datos añadidos a la información de vacíos entregada por el software son los miembros de cada vacío, la densidad central del vacío y la densidad de cada galaxia (según la métrica seleccionada).

1.4. Contenidos

Este documento se organiza de la siguiente forma: El capítulo 2 describe herramientas geométricas comúnmente utilizadas en los algoritmos de búsqueda de vacíos, además de describir algoritmos desarrollados previamente en tal área. El capítulo 3 describe en mayor detalle el problema abordado, exponiendo las principales motivaciones para realizar análisis y extensiones sobre DELFIN. El capítulo 4 presenta el estudio y análisis previo de la implementación de DELFIN. En el capítulo 5 incluye una descripción de la solución alcanzada en este trabajo, exponiendo las métricas utilizadas, las modificaciones y extensiones realizadas al trabajo realizado por Alonso[2]. El capítulo 6 presenta la validación del nuevo algoritmo, evaluando los resultados con datos artificiales y datos de catálogo, además de incluir estudios de escalabilidad. En el capítulo final se resume el trabajo realizado y se describen posibles trabajos a realizar en futuras investigaciones.

Capítulo 2

Antecedentes

En este capítulo se discutirán distintos métodos desarrollados para encontrar vacíos, sus influencias y contribuciones, para luego llegar al desarrollo de DELFIN. Previo a esto, para poder entender mejor los distintos métodos, se introducirán las herramientas geométricas utilizadas en algoritmos de búsqueda de vacíos, y particularmente en DELFIN: el diagrama de Voronoi y la teselación de Delaunay.

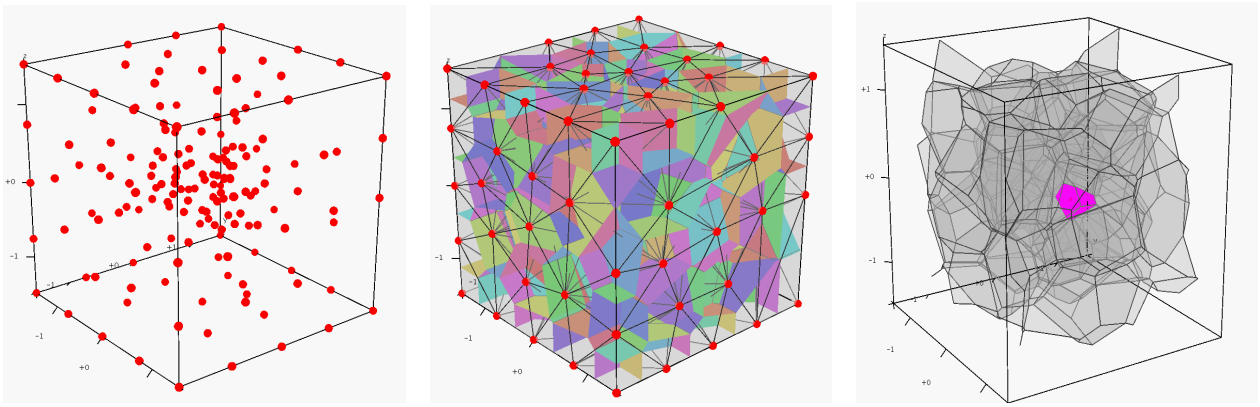
2.1. Diagrama de Voronoi y Triangulación de Delaunay

Dado H un espacio d-dimensional y S un conjunto finito de sitios (puntos en H), la *teselación de Dirichlet* o *diagrama de Voronoi* (DV) de S corresponde a la partición de H en regiones, una para cada punto en S, donde cada punto p en el espacio pertenece a la región asociada con el punto en S más cercano a p. En otras palabras, p pertenece a la región asociada con $s \in S$ *ssi* no hay un sitio $\hat{s} \in S$ más cercano a p que s. Cada región de Voronoi es una región convexa.

La *triangulación de Delaunay* (TD) es el grafo dual del diagrama de Voronoi (ver figura 2.1). Dado un conjunto de sitios S en un espacio d-dimensional H, la TD es una malla formada por d-símplices (triángulos en 2 un espacio 2-d, tetraedros en un espacio 3-d) cuyos arcos unen pares de sitios que comparte un arco de Voronoi. En una triangulación de Delaunay se satisface la siguiente restricción: ningún punto $s \in S$ se encuentra en el interior de la d-esfera circunscrita a un d-símplice de TD(S). Por definición, esta restricción permite que en la superficie de la esfera se ubiquen puntos, y de hecho la esfera debe tener en su superficie al menos los d+1 vértices del símplice circunscrito.

Algunas propiedades de la teselación del espacio generada por la malla de símplices son: La unión de todos los símplices forma la envoltura convexa de los puntos. Para cada sitio s existe un arco bs en la triangulación que lo une con su vecino más cercano b. Si una esfera que su superficie pasa por dos sitios, no contiene ningún sitio en su interior, el segmento que conecta los dos puntos es un arco de la triangulación de Delaunay de los puntos dados.

El largo de los arcos en una triangulación de Delaunay nos entrega información sobre la densidad local. Cada arco e en la triangulación es una cuerda para alguna esfera B sin sitios



(a) Conjunto de puntos (b) Teselación de Delaunay (c) Regiones de Voronoi

Figura 2.1: Conjunto de 164 puntos en un cubo (a), la teselación o tetraedrización de Delaunay de dichos puntos en donde las líneas representan las aristas de los tetraedros, y el diagrama de Voronoi de los puntos, en donde las caras de Voronoi se encuentran coloreadas de forma aleatoria (b), y las regiones de Voronoi delimitadas con una región resaltada (c).

en su interior. Como una cuerda, e debe ser de igual o menor longitud que el diámetro d_B de B, implicando la existencia de una esfera vacía de diámetro mayor o igual a $|e|$.

Para cada esfera B de radio r_B circunscrita a un tetraedro de Delaunay, si su centro se encuentra dentro de la envoltura convexa del conjunto de puntos, entonces existe un arco en la triangulación de largo al menos $l = \frac{2\sqrt{6}}{3} r_B$, es decir, el largo del arco de un tetraedro regular con B como su esfera circunscrita.

Los diagramas de Voronoi son comúnmente utilizados para identificar regiones de alta densidad dado un conjunto de galaxias. Un DV, compuesto de regiones con puntos en su centro, es una mejor teselación para encontrar regiones de alta densidad en comparación con una TD, en la que los símplices son regiones vacías. Una TD también captura la diferencia de densidades entre regiones, y a diferencia de los DV, se pueden identificar de forma más natural las regiones de baja densidad.

Comparando las triangulaciones de Delaunay con los diagramas de Voronoi se encuentra que en las TD los elementos base son símplices definidos (según la dimensión), mientras que en los DV son poliedros convexos que si bien se generan en menor cantidad que los símplices de las TD, son más difíciles y costosos de evaluar.

Una deficiencia de las TD respecto a los DV es que con un cambio en los puntos, la triangulación puede cambiar considerablemente; los arcos (2-d) o caras (3-d) son intercambiados entre símplices para generar nuevos símplices que cumplan con la restricción de Delaunay y el intercambio se puede propagar por símplices vecinos, llegando a generar cambios notorios. Esto no es un problema en los DV, donde un cambio en los vecinos de un punto se ven reflejados por una transformación más fluida en la que se genera un nuevo poliedro, cambiando los arcos de los poliedros adyacentes.

2.2. Trabajo Previo

Los vacíos cosmológicos son amplias regiones vacías en la red cósmica, descubiertos por primera vez hace más de cuarenta años (Gregory y Thompson 1978 [14]; Jõeveer et al. 1978 [17]; Kirshner et al. 1981 [20]), presentan propiedades intrigantes, una de ellas es que contienen una estructura compleja de materia oscura (Gottlober et al. 2003 [13]; Aragon-Calvo y Szalay 2012 [4]). El estudio de los vacíos se ha convertido en un área de estudio importante en cosmología (y más precisamente en cosmografía), ya que son una herramienta útil para comprender la estructura a gran escala del universo; ofrecen un entorno más limpio para estudiar las relaciones entre galaxias y materia oscura, son regiones del espacio útiles para investigar fenómenos astrofísicos como la evolución de las galaxias.

Si bien los primeros vacíos fueron identificados por mera inspección visual, rápidamente se hizo necesaria la información objetiva sobre los mismos. Existen catálogos públicos de vacíos identificados en sondeos de galaxias por desplazamiento hacia el rojo (*galaxy redshift surveys* en inglés), principalmente el *Sloan Digital Sky Survey* (SDSS), pero si bien la información sobre la forma y tamaño de los vacíos se ha refinado, todavía pueden existir mejoras, por lo que se siguen desarrollando métodos de búsqueda de vacíos. Existen variadas publicaciones que presentan estos métodos, los cuales están basados en variadas técnicas que serán discutidas más adelante, pero la mayoría de los códigos se mantienen privados. Es por esto y para promover al desarrollo de comunidades, es que sigue siendo importante el proveer códigos abiertos para la búsqueda de vacíos.

Los primeros algoritmos de búsqueda de vacíos se diseñaron para encontrar regiones en las que no hubiesen galaxias. A medida que se obtenía más información, se adoptaron nuevas estrategias que se ajustaran a las definiciones alternativas: se observó que el interior de los vacíos no es efectivamente vacío, sino que contiene una baja densidad de galaxias.

A partir de esto, los trabajos recientes sobre vacíos cósmicos utilizan parámetros asociados a la densidad, por ejemplo el parámetro de sobredensidad δ:

$$\delta = \frac{\rho - \bar{\rho}}{\bar{\rho}} \tag{2.1}$$

donde $\bar{\rho}$ es la densidad promedio de la muestra, y ρ es la densidad promedio del vacío descrito. Con este parámetro, lo vacíos sin galaxias en su interior tienen una sobredensidad igual a -1. En general, los valores de sobredensidad utilizados en las distintas definiciones de vacíos son $-1 \leq \delta \leq -0{,}5$.

A continuación se describen distintos algoritmos para encontrar vacíos, explicando cómo funcionan y destacando sus contribuciones al área.

2.2.1. Identificadores de vacíos basados en geometría

2.2.1.1. Voidsearch

Este algoritmo por Kauffmann y Fairall (1991)[19] comienza construyendo una malla cubica, donde el tamaño de cada celda es la mitad de la distancia promedio entre galaxias. Se comienza formando un vacío con la creación del cubo más grande posible a partir de cubos

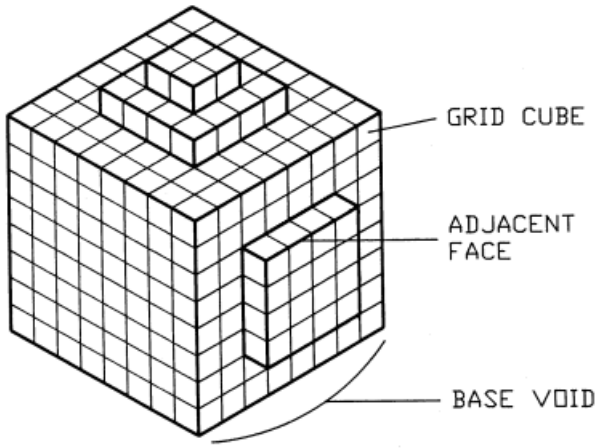


Figura 2.2: Una ilustración del procedimiento de la localización de vacíos en el algoritmo VOIDSEARCH. Un ejemplo del vacío cúbico base con caras adyacente. Figura obtenida de *Kauffmann G. & Fairall A.P.* (1991)[19].

de la malla que no se han utilizado. A este cubo se le agregan capas de cubos adyacentes a una de sus caras (ver figura 2.2), con la restricción de que la capa cubra al menos 2/3 del área de la cara del vacío, repitiéndose hasta que no se puedan agregar más capas. Este proceso se repite hasta que no se puedan construir más cubos. Este es uno de los primeros localizadores de vacíos, en este no se permiten galaxias en el interior de los vacíos ni que estos se intersequen.

2.2.1.2. Void Finder

Este algoritmo publicado por El-Ad y Piran (1997)[11] es la base para otros algoritmos como los de Hoyle & Vogeley y Foster & Nelson. Basado en distribuciones de Voronoi, consiste en dos etapas: la construcción de la pared y la localización de vacíos.

Para la etapa de construcción de la pared, definen a una galaxia de la pared como una galaxia que tiene al menos n otras galaxias de pared en un radio l alrededor de ella. Una galaxia que no satisface esta definición se considera una galaxia de campo. La separación de galaxias de la pared l la definen como sigue: Si la distancia al n-ésimo vecino más cercano de una galaxia es l_n, y en una muestra el promedio de estas distancias es $\bar{l_n}$ y la desviación estándar es σ_n, el radio l de separación es $l = \bar{l_n} + \beta\sigma_n$, con $\beta = 1{,}5$. Además utilizan $n = 3$, ya que afirman que permite filtrar cadenas largas y delgadas de galaxias.

En la etapa de localización se buscan esferas sin galaxias de pared en su interior. A partir de estas esferas se construyen los vacíos: un vacío está compuesto de distintas esferas -de diferentes radios, para cubrir todo el volumen del vacío- superpuestas. Los vacíos se generan a partir de expansión de una esfera vacía hasta que una galaxia se encuentre en su borde. Luego se expande en la dirección definida por esa galaxia hacia el centro de la esfera, hasta que otra galaxia se encuentre en la superficie. Una vez más, se expande en la dirección definida por el punto medio de las dos galaxias hacia el centro de la esfera, hasta que una tercera galaxia se encuentre en la superficie. Finalmente, se expande en la dirección de la normal al plano definido por las tres galaxias hacia el centro de la esfera, hasta que una cuarta galaxia

se encuentre en la superficie de la esfera.

Las esferas maximales se unen en varias iteraciones, deteniendo el proceso cuando se alcanza una resolución de vacío deseada, definida como el diámetro d de una esfera mínima usada como un bloque de construcción de vacío. Se denota la resolución de vacío usada en la i-ésima iteración como $d_{\rm i}$. Los valores para $d_{\rm i}$ van decreciendo, y utilizando $\xi = 0{,}85$ se define $d_{\rm i+1} = \xi\ d_{\rm i}$. En la i-ésima iteración, todas las esferas no utilizadas de diámetro mayor a $\xi d_{\rm i}$ son detectadas. Para esto, el método previo de detectar esfera vacías maximales es utilizado en una malla con celdas de tamaño $l_{cell} = \xi d_{\rm i}/\sqrt{3}$, comenzando de celdas que no estuviesen contenidas en vacíos encontrados previamente. Entonces, el tamaño de las celdas garantiza que todas las esferas maximales de diámetro mayor a $\xi d_{\rm i}$ sean encontradas; si se encuentran esferas más pequeñas, estas son descartadas. Luego las esferas son unidas bajo las siguientes condiciones:

- Al menos una esfera en el grupo debe tener diámetro mayor a $d_{\rm i}$, el diámetro más grande del grupo será llamado d_{max}.
- Se pueden agregar esferas mientras la intersección de su superficie con la superficie de otra esfera del grupo sea una circunferencia de diámetro mayor a ξd_{max}.

Finalmente, las esferas de diámetro $d > d_{\rm i}$ que no han sido agregadas a un grupo se unen a un vacío construido anteriormente.

Como para construir los vacíos solo se utiliza la información de las galaxias de pared, se pueden detectar vacíos que contengan galaxias de campo. Además, las esferas maximales corresponden a las esferas circunscritas de Delaunay, y la intersección en la superficie de las esferas son los círculos circunscritos en la cara compartida en la triangulación de Delaunay.

Hoyle & Vogeley

El algoritmo de Hoyle y Vogeley (2002)[16] se basa en el algoritmo *void finder* descrito arriba. Toma las etapas de construcción de la pared y de localización de vacíos, pero el proceso no es iterativo. En una sola pasada se detectan las esferas en una malla con tamaño de celda $l_{cell} = l_3$, haciendo crecer esferas maximales desde el centro de cada celda vacía.

Una vez que todas las esferas maximales son detectadas, el proceso de construcción de vacíos es diferente. Las esferas se evaluan de la más grande a la más pequeña. Hay tres posibilidades:

- Si la esfera no se superpone a vacíos por más de η_1, se convierte en un nuevo vacío (esto siempre ocurre para la esfera más grande).
- Si se superpone a solo un vacío por más de η_1, se convierte en un nuevo miembro de ese vacío.
- Si se superpone a más de un vacío por η_1, entonces la esfera se descarta.

Seleccionan $\eta_1 = 10\,\%$, afirmando que los vacíos obtenidos con este valor aparecen visualmente como regiones distintas. Para no identificar huecos entre paredes galácticas como

vacíos, agregan un umbral como tamaño mínimo de vacíos, así solo se evalúan las esferas más grandes que el umbral.

Como último paso se realiza una mejora en el volumen de los vacíos, si las esferas menores al umbral se intersecan por más de $\eta_2 = 50\,\%$ con un vacío, se unen a este; si se superpone con más de un vacío, se descarta. Se utiliza ese valor para aumentar el tamaño del vacío sin cambiar la forma esférica del mismo.

Foster & Nelson

El algoritmo de Foster y Nelson (2009)[12] se basa en el algoritmo de Hoyle y Vogeley. Hacen un cambio ligero al valor $l = \bar{l_n} + \beta\sigma_n$ de la definición de El-Ad & Piran, utilizando $\beta = 2{,}0$ en vez de 1,5, y reducen el tamaño de la celda a la mitad, permitiendo que esferas más pequeñas también sean parte de un vacío.

La diferencia principal con Hoyle y Vogeley es una distinción más detallada de las posibilidades que pueden tener las esferas al ser evaluadas. En la primera etapa se evalúan las esferas en el siguiente orden:

1. Si la esfera se superpone a dos o más vacíos por más de β_3 a cada uno, la esfera se descarta.

2. Si la esfera se superpone a un (1) vacío por al menos β_1, se convierte en miembro del vacío.

3. Si la esfera no se superpone a algún vacío, se convierte en un nuevo vacío (esto siempre ocurre para la esfera más grande).

Luego de esto viene la mejora del volumen del vacío, evaluando las esferas con tamaño menor al umbral definido:

1. Si la esfera se superpone a dos o más vacíos por más de β_3 a cada uno, se descarta.

2. Si la esfera se superpone a solo un (1) vacío por al menos β_2, se convierte en miembro del vacío.

En esta evaluación se utilizan los valores $\beta_3 = 2\,\%$, $\beta_1 = 10\,\%$ y $\beta_2 = 50\,\%$. Al igual que en el algoritmo de Hoyle y Vogeley, el método de expansión del volumen de los vacíos permite el crecimiento de vacíos esféricos.

2.2.2. Identificadores de vacíos basados en transformación divisoria

WVF

El algoritmo *Watershed Void Finder* (WVF), desarrollado por Platen et al. (2007)[25] utiliza suposiciones sobre la escala, estructura y forma de los vacíos. Está basado en la transformación divisoria (*watershed transform* en inglés), una técnica utilizada en procesamiento de imágenes para realizar segmentaciones. A continuación se presenta una descripción sintetizada del algoritmo.

1. Se utiliza un estimador de campo de teselación de Delaunay (DTFE por sus siglas en inglés) para definir un campo de densidad sobre los puntos de la muestra. La densidad ρ en cada punto $x_{\rm i}$ se define como $\rho(x_{\rm i}) = \frac{1+D}{V(\mathcal{W}_{\rm i})}$, donde $D = 3$ es la dimensión del espacio, y $V(\mathcal{W}_{\rm i})$ es la suma del volumen de los tetraedros de Delaunay que tienen $x_{\rm i}$ como uno de sus vértices. La densidad en otros puntos se define por funciones lineales. Si los vértices son $x_0, ..., x_3$, entonces la densidad en un punto $x = \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$ en el tetraedro es :

$$\rho(x) = \alpha_0 \rho(x_0) + \alpha_1 \rho(x_1) + \alpha_2 \rho(x_2) + \alpha_3 \rho(x_3) \tag{2.2}$$

2. Luego se construye una cuadrícula, donde el tamaño de las celdas está en el orden de la separación entre las partículas. Para cada celda, se elige un número aleatorio de Montecarlo de muestras y la densidad de la celda se define como el promedio de la densidades de esas muestras.

3. La cuadrícula es filtrada usando un filtro ordenado por rango de vecinos naturales, en el que la densidad de cada muestra es reemplazada por la mediana, mínimo o máximo de su vecino natural.

4. Se discretiza la imagen dividiendo uniformemente la distribución de densidad acumulada y reemplazando cada densidad por el valor correspondiente en la partición.

5. Se reduce el ruido utilizando una operación de apertura y cierre de 2 píxeles (respectivamente equivalentes a sustracción de Minkowski seguida de una suma de Minkowski, y una suma de Minkowski seguida de una sustracción).

6. Se aplica la transformación divisoria por inundación: comenzando en los píxeles rodeados exclusivamente por píxeles con mayor densidad (mínimos regionales), el terreno se inunda a niveles crecientes, agregando los puntos inundados a la cuenca o base del mínimo correspondiente. Cuando todos los píxeles han sido inundados, quedan *parches de vacíos*, los cuales teselan toda la región.

7. Se corrigen los parches, para lo cual se eliminan todos los límites de la división cuya densidad es menor que la subdensidad típica de los vacíos $\Delta = -0{,}8$ (ver figura 2.3).

Una de las propiedades más interesantes de este algoritmo es que detecta vacíos relativamente libre de parámetros. Esta propiedad lo hace potencialmente capaz de detectar vacíos de cualquier tamaño. El uso de un campo de densidad también le permite ignorar la posición de los puntos y centrarse en su vecindad. Estas propiedades se comparten con otros algoritmos basados en la transformación divisoria.

ZOBOV

El algoritmo *ZOnes Bordering on Voidness* (ZOBOV) publicado por Neyrinck (2008)[23], se comporta de manera similar a WVF, pero utiliza las regiones de Voronoi del conjunto de puntos.

Comienza obteniendo el campo de densidad de partículas de materia oscura utilizando un estimador de campo de teselación de Voronoi (VTFE por sus siglas en inglés). VTFE es

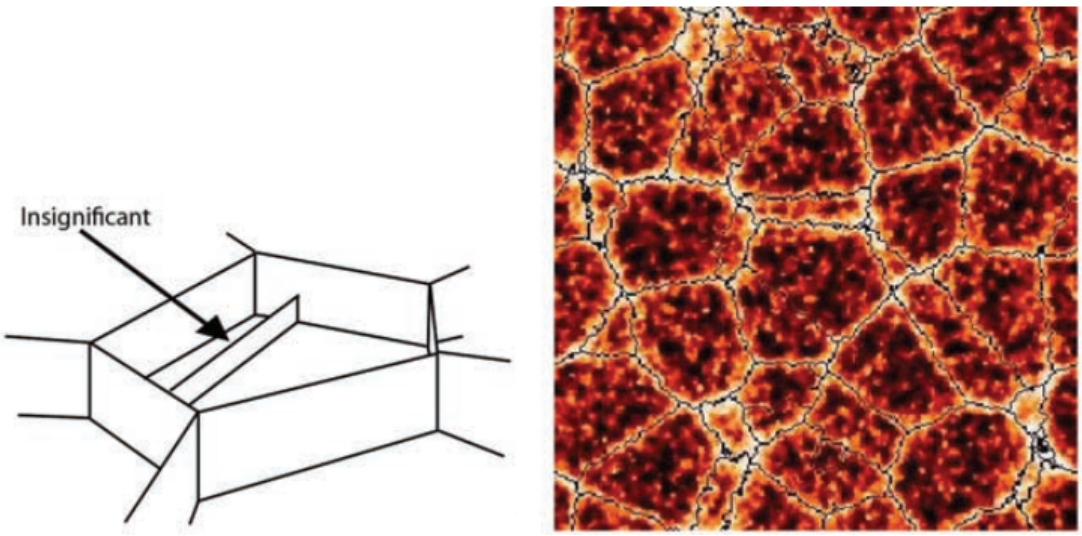


Figura 2.3: El concepto de transformación divisoria por jerarquía. No todas las líneas producidas en la división son relevantes. Se descartan si no cumplen con un criterio en particular (ej. si tienen un contraste menor a cierto umbral). Solo se mantienen los segmentos divisores relevantes. En la imagen de la izquierda se observa la división computada luego de descartar los límites con un contraste menor a 0.8. Figura obtenida de *Platen E.* (2007)[25].

similar a DTFE, pero la densidad estimada para cada partícula p es $1/V(p)$, donde $V(p)$ es el volumen de la región de Voronoi asociada con la partícula p. Luego aplica un proceso de transformación divisoria donde cada región de Voronoi se une a su vecino de menor densidad, resultando en regiones llamadas *zonas*.

Las zonas se unen inundando cada zona z, hasta que una más profunda que z esté por ser inundada. La profundidad o altura corresponde a la densidad de la celda de Voronoi menos densa de la zona, por lo que una zona menos densa sería más profunda que una con alta densidad. Las zonas inundadas cuando se detiene el proceso de unión corresponden a vacíos. Si la zona con la que se inicia la inundación es la más profunda, el proceso se detiene antes de inundar la última zona (contiene a todas las zonas exceptuando la menos profunda).

Este algoritmo genera vacíos superpuestos; de hecho, por cada superposición, uno de los vacíos es un subconjunto del otro. Esta propiedad permite formar jerarquías de vacíos, lo que puede dar información sobre el proceso de formación de vacíos. Por ejemplo, un vacío formado por la fusión de dos vacíos puede identificarse en esta jerarquía como dos vacíos diferentes, con uno de ellos como el padre del otro en esta jerarquía.

VIDE

El algoritmo utilizado por *Void IDentification and Examination toolkit* (VIDE) de Sutter et al. (2015)[28] está basado ZOBOV, extendiéndolo y mejorándolo. En particular, agregan variaciones para los valores de densidad de las regiones de Voronoi si los sondeos utilizados son limitados por magnitud o por volumen.

En la etapa de unir las zonas para crear vacíos, se agrega un umbral basado en la densidad, donde las zonas adyacentes se unen solo si la densidad de la pared que las divide es menor

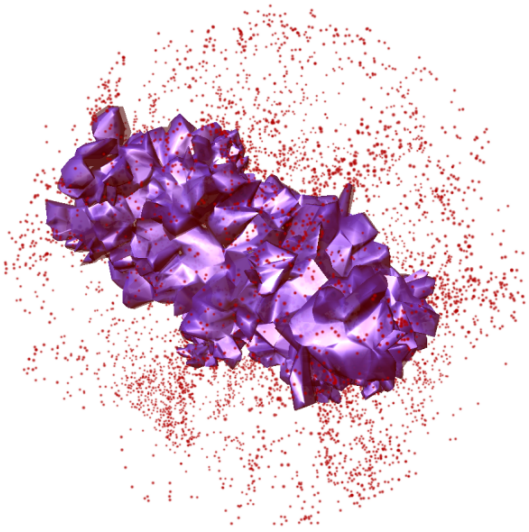


Figura 2.4: Ejemplo de vacío encontrado por VIDE. Las regiones de Voronoi que definen al vacío están en morado con las galaxias en rojo. El radio efectivo del vacío es 20 h^{-1}Mpc en una región esférica de 50 h^{-1}Mpc. Figura obtenida de *Sutter et al.* (2015)[28].

que 0,2$\bar{n}$, donde $\bar{n}$ es la densidad promedio de las partículas. Para sondeos limitados por el volumen, la densidad promedio de las partículas se calcula dividiendo el volumen del sondeo por el número de galaxias. Para sondeos limitados por magnitud, la densidad promedio de partículas se estima de la función de selección radial. Estos criterios de densidad permiten que los vacíos no se extiendan a estructuras con sobredensidad, además limita la profundidad de la jerarquía de vacíos.

Todos los vacíos tienen un solo padre, pero potencialmente muchos hijos, y los hijos de un padre ocupan distintos subconjuntos de volúmenes del padre, los cuales se encuentran separados por cumbres bajas. Sin la aplicación del corte por densidad menor a 0,2$\bar{n}$, ZOBOV identifica un gran vacío que contiene a todo el volumen, por lo que genera solo un árbol de jerarquía de vacíos. Con VIDE se generan múltiples jerarquías.

Los vacíos encontrados por VIDE son agregaciones de celdas de Voronoi que adquieren una forma esférica (ver figura 2.4), comparten una misma cuenca o región de baja densidad, y están limitadas por el mismo conjunto galaxias de pared, las que tienen una densidad mayor al vacío. Estos vacíos incluyen a las partículas de las paredes en su definición, por lo que no tienen una densidad promedio pre-definida.

2.2.3. El Algoritmo DELFIN

En esta sección se explicará primero el algoritmo en 2-d, continuando con la versión en 3-d y el motivo de las mejoras propuestas. La implementación de estos algoritmos es en C++ para la versión en 3-d, y en Python para la versión en 2-d. Para la última se busca generar una optimización en un trabajo de título de 2021[1].

[1]Marjorie Pulgar. *Optimización de un algoritmo para encontrar vacíos poligonales en un conjuntos de puntos en el plano*. Magíster en Tecnologías de la Información. 2021

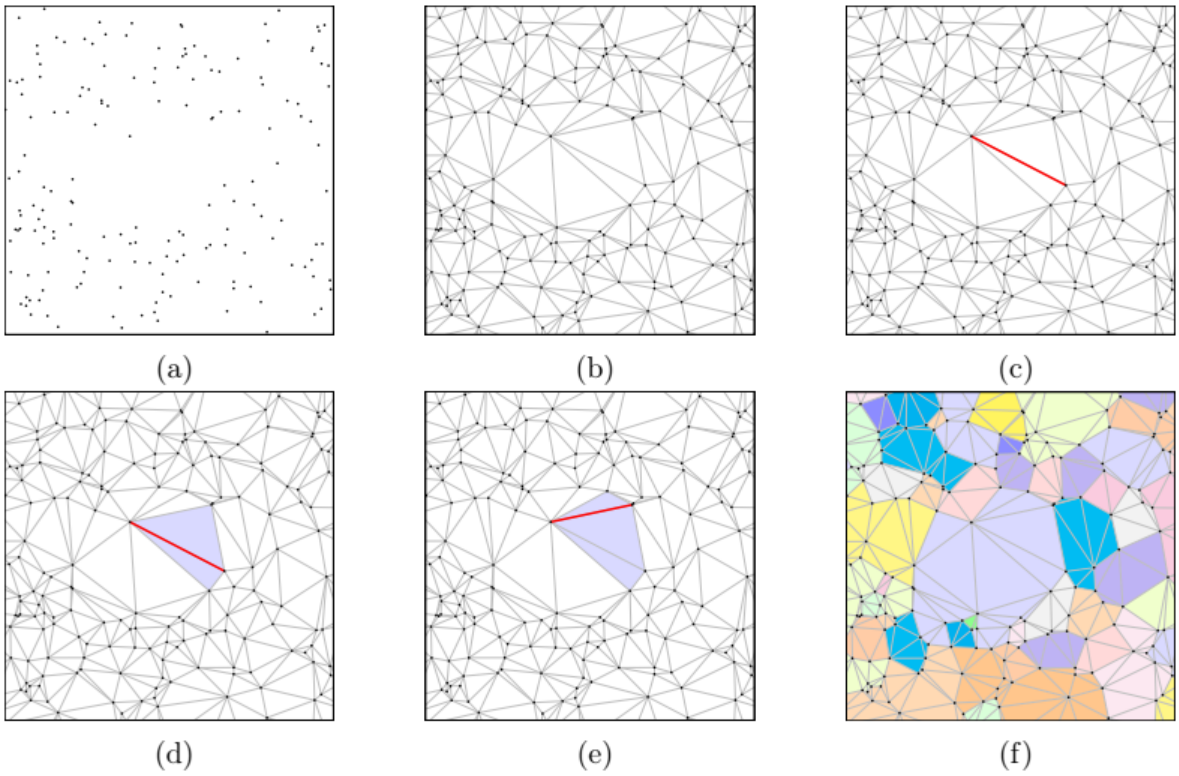


Figura 2.5: Pasos de la identificación vacíos realizada por DELFIN. Se muestra la generación de una partición a partir de puntos y de la unión de triángulos siguiendo a las aristas más largas (línea roja en las figuras). En la figura 2.5.f se muestra la partición del espacio en candidatos a vacíos. Figura obtenida de *Alonso R.* (2016)[2].

DELFIN en 2-d

El algoritmo identificador de vacíos de Hervías et al. (2014)[15] y Alonso (2018)[3] DELFIN, detecta vacíos al identificar grandes regiones vacías en el espacio a partir de una triangulación de Delaunay de un conjunto de puntos que representan galaxias. Las regiones con menor densidad tienen un espacio más grande disponible para una esfera vacía en su interior.

El algoritmo DELFIN comienza a construir vacíos por mínimos locales de densidad, los cuales están definidos sobre los arcos de la triangulación de Delaunay en vez de los puntos. Esto se basa en que los arcos más largos de la triangulación garantizan la existencia de un círculo vacío que tiene a uno de esos arcos como cuerda. Los mínimos locales se definen entonces como los arcos que son los más largos de ambos triángulos a los que pertenecen.

A partir de un arco mínimo local, une los dos triángulos que lo comparten. Estos dos triángulos se consideran una semilla de un vacío. Luego, a cada semilla se le unen los triángulos adyacentes si la arista que comparten con el triangulo que pertenece al vacío es su arista más larga. De esta forma, todos los arcos más largos de los triángulos en el vacío se encuentran en el interior del mismo (i.e. no son aristas del borde del vacío).

Esta estrategia genera una nueva partición del conjunto de puntos, en que conjuntos de

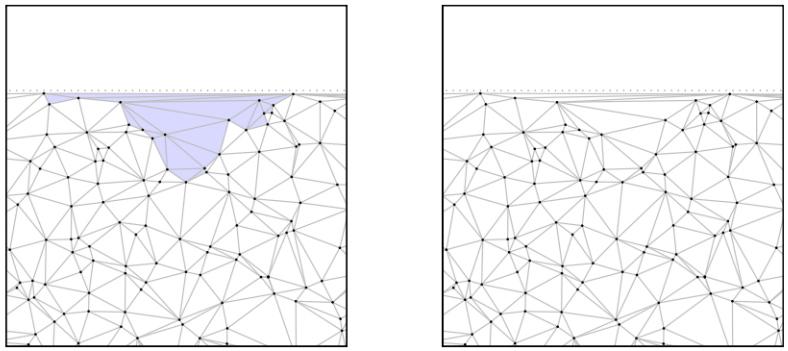
triángulos se agrupan alrededor de arcos más largos locales (ver figura 2.5). Estos grupos, antes llamados semillas de vacíos, serán tratados como candidatos a vacíos. Cada candidato a vacío se describe como un polígono definido por los arcos que están una sola vez en este conjunto de triángulos (su frontera).

Los candidatos a vacíos vecinos son unidos para formar uno más grande de acuerdo a ciertos criterios geométricos. En particular, se utiliza un criterio propuesto por algunos buscadores de vacíos[12], en el que se expresa la condición de que dos candidatos se unen ssi uno de los arcos que comparten es más largo que un parámetro r_3 definido como

$$r_3 = \mathrm{d}_3 + \lambda\sigma_3 \tag{2.3}$$

donde d_3 es la distancia promedio de cada punto a su tercer vecino más cercano, λ es un parámetro que toma el valor de 2,0, y σ_3 es la desviación estándar para la distancia al tercer vecino más cercano.

Para que no todo polígono sea considerado un vacío, se descartan las regiones cuya área es menor que un umbral definido por el usuario. Además se descartan los vacíos que contienen uno o más de sus arcos en la superficie de la cobertura convexa de los datos (ver figura 2.6).



(a) (b)

Figura 2.6: Manejo de vacíos en el borde. (a) Un vacío en el borde. (b) El resultado luego de la aplicación del filtro que descarta los vacíos con aristas en los bordes. La línea punteada indica el límite del conjunto de datos. Figura obtenida de *Alonso R.* (2016)[2].

DELFIN en 3-d

De manera similar al caso en 2-dimensiones, dado un conjunto de puntos en el espacio de 3-dimensiones, se genera una malla de tetraedros que satisfacen la condición de Delauney. Los tetraedros obtenidos son ordenados por su arista más larga. Luego, los arcos localmente más largos se utilizan como punto de partida para un nuevo vacío. En 3-d, un arco localmente más largo se define como el arco más largo para todos los tetraedros que lo contienen. Todos los tetraedros que lo comparten se agrupan en una semilla o candidato de vacío.

Los tetraedros se agrupan iterativamente, agregando tetraedros vecinos que no pertenecen a una semilla y tienen su arco más largo compartido con la semilla que se está formando. A diferencia del caso en 2-d, los tetraedros agregados al candidado puede no compartir caras con

los tetraedros del vacío al que se unió. Este proceso se repite para todos los arcos localmente más largos, hasta que el arco evaluado tenga un largo menor a un largo umbral l_{min} definido para detener el proceso de formación de candidatos. Luego de la formación de candidatos, este algoritmo incluye métodos para filtrar vacíos cuyo volumen V es menor a un umbral V_{min}.

Posterior a esto, el algoritmo realiza la agrupación de candidatos a vacíos utilizando el criterio de unir ssi el arco que comparten es más largo que el parámetro r_3 definido en la ecuación 2.3. Para terminar, se descartan todos los vacíos con un arco en la cobertura convexa de los datos.

Este algoritmo presenta problemas de fragmentación de vacíos, además de producir vacíos falsos en la cercanía de la cobertura convexa de los datos. Por estos problemas, la versión actual de DELFIN no puede aún ser utilizada para detectar vacíos reales. En un estudio de validación astrofísica[1] se menciona la necesidad de modificar el algoritmo para que entregue información relevante de los vacíos, especialmente que no se alteren los gradientes de densidad de las muestras, y sea posible realizar los estudios astrofísicos relacionados con vacíos cosmológicos. Esto está relacionado a identificar la densidad de puntos que hay en cada vacío.

Capítulo 3

Problema

3.1. Planteamiento y relevancia

El principal problema abordado es que la versión actual de DELFIN aún no puede ser utilizada para detectar vacíos en datos observacionales. Los resultados entregados por el algoritmo y la información que proporciona sobre los vacíos que encuentra no resulta útil para el análisis astrofísico requerido.

En el estudio de Hervías et al. (2014)[15], el algoritmo DELFIN en 3-d presentaba problemas de fragmentación de vacíos en componentes adyacentes más pequeños, este problema se aprecia en la figura 3.1. Para tratar este problema, en la versión actual de DELFIN, se lleva a cabo la unión de candidatos de vacíos (potencialmente fragmentos de un vacío de mayor tamaño) mediante el criterio del tercer vecino más cercano. Sin embargo, en la validación de resultados obtenidos por Alonso (2016)[2](comparados con los obtenidos por el método de Foster & Nelson), ocasionalmente se presentan vacíos fragmentados, producto de que los arcos compartidos entre candidatos son demasiado cortos. Estos resultados son consecuencia del criterio utilizado, por lo que debe ser ajustado para obtener resultados de distinta calidad en la unión de candidatos de vacíos.

En pruebas realizadas con datos observacionales (en particular con datos de SDSS Data Release 10[27]), los resultados obtenidos por DELFIN presentan más inconvenientes que la fragmentación de vacíos. Uno de los problemas presentados es la partición del espacio en poliedros, resultante de una relativamente baja unión de candidatos a vacíos (ver figura 3.2). Este caso ocurre cuando no se ingresan los parámetros adecuados y se evalúan arcos muy pequeños en la etapa de generación de candidatos, por lo que su densidad es lo suficientemente grande como para que los vacíos se mantengan separados. Si bien este puede ser considerado un problema de usuario, la complejidad en parámetros significa una barrera de entrada para el uso y comprensión del algoritmo.

El manejo de vacíos en el borde de los datos también genera problemas. En particular, ocurre la pérdida de candidatos de vacíos con arcos en la cobertura convexa. Debido a que en la generación de candidatos se evalúan los arcos por longitud decreciente para generar candidatos, los tetraedros de la cobertura convexa -al ser más alargados y tener arcos más

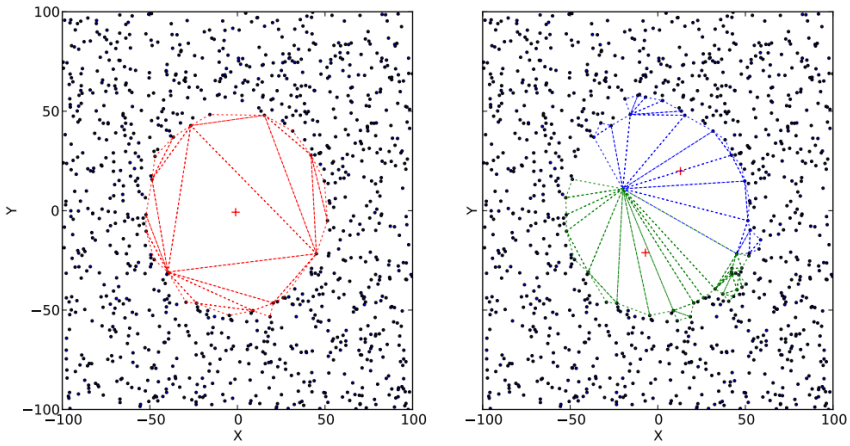


Figura 3.1: En la izquierda, el vacío encontrado en un conjunto de puntos. En la derecha, el mismo conjunto de puntos, pero con un punto en el interior del vacío. Se encuentran dos vacíos adyacentes. Las cruces son los centroides de los vacíos. Figura obtenida de *Hervías et al.* (2014)[15].

largos- se terminan convirtiendo en candidatos. De esta forma, los vacíos que se encuentren en su completitud dentro del espacio de los datos, pero que estén en la cercanía del borde, a menudo se descarten porque se unen a un candidato que tiene un arco en el borde.

Hasta ahora se han mencionado los resultados obtenidos sin especificar cual es la información que se tiene de estos. La utilidad de la información es otro problema que debe ser abordado. Actualmente los datos entregados por cada vacío son: centroide, volumen y radio de una esfera con volumen equivalente al del vacío. Esta información no solo es poco detallada, sino que también es poco relevante para los estudios astrofísicos relacionados con vacíos cosmológicos, ya que a través de ellos no se puede acceder a información primordial de cada vacío.

Como se mencionó brevemente en la introducción, uno de los motivos para estudiar los vacíos cosmológicos es que el estudio de sus formas, tamaños y flujos aporta información sobre parámetros cosmológicos globales. Es por esto que datos como la posición, forma, tamaño, densidad del vacío y las galaxias que lo componen, son esenciales para su identificación y posterior estudio.

Para tratar los problemas mencionados hasta ahora se deben llevar a cabo extensiones en el software, por lo que se presenta la necesidad de estudiar la calidad de DELFIN como software orientado a objetos y su extensibilidad a nuevos criterios geométricos. Complementario a esto, se debe llevar a cabo un análisis del uso de memoria, tanto en escalabilidad en función de los datos de entrada, como en la detección de problemas en el manejo de memoria. Ambos análisis se vuelven necesarios para evitar o resolver fugas de memoria y lograr la mejor eficiencia posible al procesar datos.

Se ha mencionado que la exploración y estudio de los vacíos es esencial para comprender la distribución de la materia cósmica, y que pueden servir como una herramienta poderosa para sondear la energía oscura. Es el descubrimiento de la energía oscura que ha impulsado

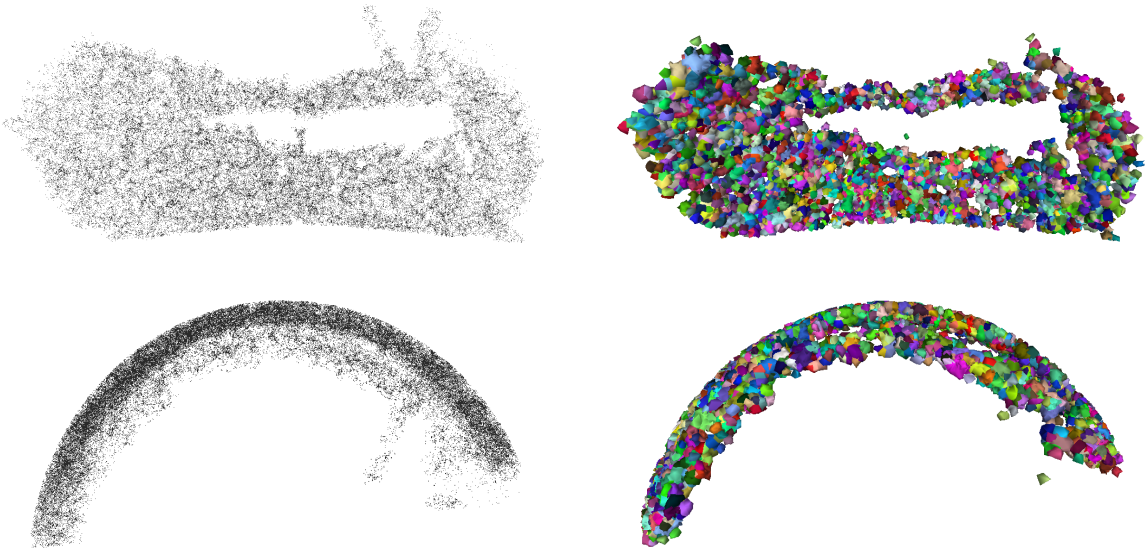


Figura 3.2: Conjunto de puntos de muestras de CMASS limitada por volumen entre *redshift* $z \geq 0{,}45$ y $z \leq 0{,}5$ de SDSS Data Release 10. En la derecha se encuentran los poliedros identificados como vacíos por la versión actual de DELFIN al evaluar arcos con longitud mayor o igual a 10 Mpc.

el aumento del interés de astrónomos para explorar vacíos. Este interés enfatiza la relevancia de la existencia de algoritmos para apoyar estos estudios. De esta manera se vuelve relevante contar con una solución a los problemas indicados para proporcionar un algoritmo funcional que utilice triangulaciones de Delaunay para abordar la búsqueda de vacíos, lo cual es de interés debido a que una cantidad significativa de los algoritmos disponibles utilizan crecimiento de esferas a partir de campos de densidad o uniones de regiones de Voronoi (poliedros).

3.2. Requisitos

La solución entregada se desarrolla a partir del software DELFIN previamente existente. A partir de este se trabaja en los problemas en dos ámbitos, que se pueden definir como: desempeño del software y extensión del software con una nueva versión del algoritmo. Los principales requisitos de la solución orientada al desempeño y calidad de software (no funcionales) son:

1. El porcentaje de cobertura de código por los tests debe superar el 75 %.
2. El acoplamiento de clases debe ser bajo.
3. El sistema debe evitar la baja cohesión de clases.
4. El programa no debe tener fugas de memoria.
5. El programa debe ser ejecutable en un sistema operativo de tipo Unix.

Los principales requisitos asociados a las capacidades del software extendido (funcionales),

relacionados a la nueva versión del algoritmo, en la cual se han de resolver los problemas de los resultados en la identificación de vacíos, son:

1. Asociar densidades a las galaxias, definiéndolas a partir de elementos de la teselación de Delaunay de los datos.

2. Identificar vacíos agrupando tetraedros en función de la densidad de los elementos de la teselación de Delaunay.

3. Permitir definir la densidad límite para la cual un elemento geométrico ya no se considera parte de un vacío.

4. Reconocer y descartar a los vacíos falsos generados en el borde de los datos por la baja densidad.

5. Entregar información que permite caracterizar los vacíos encontrados:

    - centroide
    - galaxias pertenecientes
    - volumen
    - densidad
    - radio de esfera equivalente
    - elipticidad

Con estos requisitos se define la estructura general de la solución, basada en la construcción de vacíos a partir de información de densidad en la teselación de Delaunay. En cuanto a los requisitos de calidad y desempeño, se debe considerar que se busca calidad y robustez considerando que se trabaja con un software orientado a objetos, es por esto que se incluyen métricas como cohesión y acoplamiento.

También se debe tener en cuenta que se busca trabajar con un programa que haga buen uso de la memoria, por lo que un requisito mínimo es identificar y eliminar las posibles fugas de memoria. Además, se debe considerar que existen millones de datos observacionales y de simulación disponibles para el estudio de vacíos cosmológicos, por lo que es importante que los tiempos de ejecución del programa sean razonables para la escala de datos que se trabaja. Se considera un buen resultado el ceñirse a los tiempos obtenidos por Alonso (2016), para quien la ejecución se ajustaba a un tiempo linealítmico, $O(n \log n)$.

Capítulo 4

Evaluación

Previo a realizar las extensiones del programa para integrar la nueva versión del algoritmo, se realiza un análisis y evaluación del software, teniendo en cuenta que se deben seguir las reglas de un diseño orientado a objetos. En este capítulo se presenta a DELFIN en su estado inicial, descrito mediante un análisis estático de código, el estado de los test unitarios, y un análisis de escalabilidad y manejo de memoria.

4.1. Análisis de calidad de software

Para el estudio de calidad se realiza un análisis estático del código fuente, evaluando métricas de calidad y vulnerabilidad de software. Las métricas principalmente utilizadas son: complejidad, acoplamiento, cohesión, número de clases, tamaño de clases, número de hijos por clase y número de métodos por clase [8, 18].

En la figura 4.1 se presenta el diagrama de clases de DELFIN. Es un total de 12 clases, entre ellas se encuentran clases para representaciones de elementos geométricos, clases que construyen y que unen vacíos. La clase PointDictionary contiene a los puntos que representan a las galaxias con sus coordenadas cartesianas. Las clases que contienen el nombre "Facet" son las que representan a los tetraedros. De la misma forma, las clases que contienen el nombre "Edge" son las que representan las aristas de la teselación.

Las herramientas utilizadas para el análisis estático de código son *CppDepend*[1] (en su versión de prueba), *SourceMonitor*[2] y *Sourcetrail*[3]. En los resultados entregados por *CppDepend*, además de las métricas se incluyen estados para reglas de diseño orientado a objetos. El resultado de este reporte contiene las siguientes reglas violadas:

- Los constructores de las clases abstractas deben ser declarados como *protected* o *private*.

[1]CoderGears. CppDepend, Version 2021.1. [Computer software] `https://www.cppdepend.com`. January 2021.

[2]Campwood Software. SourceMonitor, Version 3.5. [Computer software] `https://www.campwoodsw.com`. April 2020.

[3]Coati Software. Sourcetrail, Version 2020.2. [Computer software] `https://www.sourcetrail.com`. June 2020

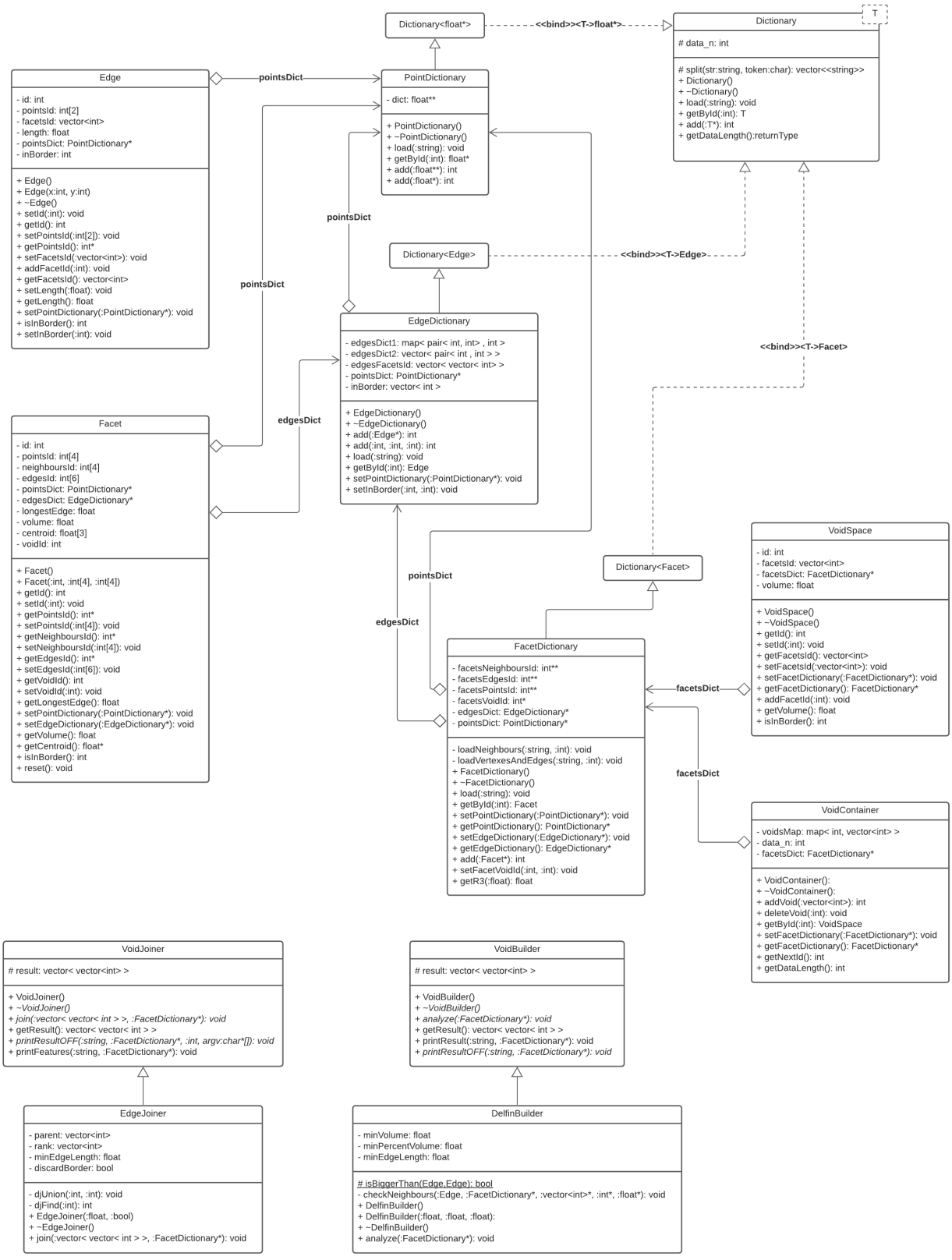


Figura 4.1: Diagrama de clases de la implementación de DELFIN

- Las clases Dictionary, PointDictionary, EdgeDictionary, FacetDictionary, Edge, Facet, VoidSpace y VoidContainer tienen variables de instancia no inicializadas en su constructor.

- La clase FacetDictionary no tiene un constructor de copia, lo que es recomendado porque la clase tiene asignaciones dinámicas de memoria/recursos.

En la tabla 4.1 se presentan los resultados de cada clase en métricas específicas. La métrica Type Rank es computada aplicando el algoritmo PageRank de Google. Los resultados presentados en la Tabla 4.1 indican que las clases PointDictionary, Edge, Dictionary y EdgeDictionary son las más *importantes* (por cantidad de referencias en el proyecto) y deben ser testeadas entera y cuidadosamente, porque la existencia de defectos en estas clases puede generar resultados catastróficos.

Según el porcentaje de código comentado se observa que la clase FacetDictionary tiene un porcentaje muy alto de comentarios (>40 %). Por otro lado, clases como VoidContainer, EdgeDictionary, PointDictionary, VoidBuilder, VoidSpace y Edge necesitan más comentarios, pues sus valores son menores al 20 % recomendado.

Name	**Type Rank**	**# Lines Of Code**	**# IL Instructions**	**# Lines Of Comment**	**% Comment**
Dictionary	1.96	11	0	0	9
PointDictionary	3.31	38	5	11.63	13
Edge	2.12	25	0	0	17
EdgeDictionary	1.75	34	6	15	12
Facet	1.35	81	3	3.57	29
FacetDictionary	1.63	130	26	16.67	43
VoidBuilder	0.33	53	0	0	16
DelfinBuilder	0.15	68	15	18.07	27
VoidJoiner	0.33	63	1	1.56	19
EdgeJoiner	0.15	33	9	21.43	17
VoidSpace	0.48	22	0	0	16
VoidContainer	0.15	17	2	10.53	11

Tabla 4.1: Métricas de calidad de código de la implementación de DELFIN.

4.1.1. Cohesión y Acoplamiento

En programación orientada a objetos se considera que la alta cohesión en un módulo y bajo acoplamiento entre módulos esta relacionado con un código de buena calidad.

Una entidad con poca cohesión es una que debe dividirse en una o más sub-entidades pues aquellos tipos o módulos no-cohesivos son difíciles de reutilizar y extender, y un intento de reutilizarlos puede introducir vulnerabilidades en el software [8].

A partir de los resultados presentados en la Tabla 4.2 se observa que por los valores obtenidos en la métrica LCOM la clase Dictionary es la más vulnerable, y que clases como Edge, EdgeDictionary y Facet pueden necesitar modificaciones para mejorar el código en su

extensibilidad [18].

Las clases en que se definen más métodos y variables son Facet, Edge y FacetDictionary (ver Tabla 4.4). Es por esto que las clases que más dificultades pueden tener para ser comprendidas y mantenidas son Facet y Edge, pues las clases donde LCOM >0.8, Número de métodos >10 y Número de variables >10 pueden ser problemáticas porque son poco cohesivas. Si bien sus valores de LCOM se encuentran bajo 0.8, se reitera la importancia de que sean enteramente testeadas.

Name	**Lack Of Cohesion Of Methods**	**Lack Of Cohesion Of Methods HS**	**Association Between Classes**
Dictionary	0.86	1	7
PointDictionary	0.33	0.4	28
Edge	0.67	0.71	5
EdgeDictionary	0.7	0.8	22
Facet	0.71	0.75	3
FacetDictionary	0.5	0.54	50
VoidBuilder	0.5	0.6	29
DelfinBuilder	0.06	0.07	36
VoidJoiner	0.5	0.6	30
EdgeJoiner	0.45	0.56	21
VoidSpace	0.55	0.6	12
VoidContainer	0.52	0.58	18

Tabla 4.2: Métricas de falta de cohesión de métodos y asociación entre clases de la implementación de DELFIN.

En cuanto al acoplamiento del código, se observan buenos resultados en métricas de herencia como la Profundidad del Árbol de Herencia (DIT) y el Número de Hijos (NOC), presentados en la Tabla 4.4. Los valores de DIT para todas las clases son 0 ó 1, lo que significa que el acoplamiento entre clases debido a herencia es bajo. Para NOC el máximo es 3 en la clase Dictionary que es una clase plantilla y las otras dos únicas clases con herencia son VoidBuilder y VoidJoiner, con 1 hijo cada una. Estos valores en NOC indican que existe poco acoplamiento por métodos y variables de instancia heredadas.

Según lo observado en la Tabla 4.3, las clases con mayor acoplamiento total son Facet y FacetDictionary. Estas clases son importantes para el sistema pues son las que identifican a los tetraedros, a partir de los cuales se generan los vacíos. Como son clases con alta responsabilidad en el sistema y necesitan ser capaces de trabajar con múltiples otras clases para obtener resultados, su nivel de acoplamiento no es necesariamente malo [18].

Otra métrica importante en el acoplamiento es la inestabilidad, que mide la relación entre acoplamiento eferente y el acoplamiento total. Un índice de inestabilidad de 1 indica es que altamente inestable y un índice de 0 significa que es estable. Clases como Dictionary, PointDictionary y Edge, que presentan alto acoplamiento aferente y bajo eferente son clases estables. Las clases con mayor inestabilidad son DelfinBuilder, EdgeJoiner y VoidContainer. En este caso la inestabilidad se explica por su nulo acoplamiento aferente. DelfinBuilder y EdgeJoiner son

clases que se encuentran al final del flujo, calculan y contienen los resultados finales (con la información de los vacíos), lo que explica que necesiten más clases para funcionar y no existan clases que dependan de ellas. VoidContainer es una clase no utilizada en el sistema y debe ser evaluado si se mantendrá en el mismo.

Name	**Afferent Coupling**	**Efferent Coupling**	**Total Coupling**	**Inestability**
Dictionary	3	0	3	0.00
PointDictionary	6	1	7	0.14
Edge	6	1	7	0.14
EdgeDictionary	4	3	7	0.43
Facet	7	3	10	0.30
FacetDictionary	6	7	13	0.54
VoidBuilder	1	4	5	0.80
DelfinBuilder	0	7	7	1.00
VoidJoiner	1	4	5	0.80
EdgeJoiner	0	6	6	1.00
VoidSpace	1	2	3	0.67
VoidContainer	0	2	2	1.00

Tabla 4.3: Métricas de acoplamiento de clases de la implementación de DELFIN.

Name	**# Instance Methods**	**Nb Static Methods**	**# Fields**	**# Children Classes**	**Depth Of Inheritance Tree**
Dictionary	7	0	1	3	0
PointDictionary	8	0	1	0	1
Edge	16	0	6	0	0
EdgeDictionary	10	0	5	0	1
Facet	21	0	10	0	0
FacetDictionary	15	0	6	0	1
VoidBuilder	8	0	1	1	0
DelfinBuilder	7	1	3	0	1
VoidJoiner	8	0	1	1	0
EdgeJoiner	7	0	4	0	1
VoidSpace	12	0	4	0	0
VoidContainer	10	0	3	0	0

Tabla 4.4: Métricas de miembros y herencia para la implementación de DELFIN.

4.1.2. Complejidad de código

La complejidad ciclomática (CC) de código permite identificar los métodos más difíciles de mantener y con mayor riesgo a tener defectos. Esta métrica también nos ayuda a reconocer la cantidad de casos de prueba a realizar por cada método.

En particular hay que revisar las clases FacetDictionary y EdgeJoiner pues es en donde

se encuentran los métodos con mayor complejidad del proyecto (ver Tabla 4.5). VoidJoiner, al tener una CC de 10 también debe ser revisada, pero no con tanta urgencia como FacetDictionary y EdgeJoiner que tienen una CC de 15, entrando en la categoría de código poco reutilizable y bastante complejo o desafiante para probar [18].

Considerando la complejidad por herencia, las métricas DIT y NOC se encuentran en valores bajos, lo que indica una baja complejidad; mientras mayor sean los valores de DIT para una clase, mayor es el número de métodos que puede heredar, haciéndola más compleja y dificultando la predicción de su comportamiento [8].

Name	**Avg Stmts/Method**	**Max Complexity**	**Max Depth**	**Avg Depth**	**Avg Complexity**
Dictionary	2.8	3	2	0.96	1.5
PointDictionary	5.8	5	5	1.48	1.83
Edge	1.7	2	2	0.74	1.07
EdgeDictionary	4.7	3	2	1.15	1.29
Facet	3.1	4	4	1.16	1.42
FacetDictionary	10.3	15	7	1.93	3.17
VoidBuilder	17.7	6	4	2.24	3.67
DelfinBuilder	11.0	6	4	2.24	3.67
VoidJoiner	21.7	10	5	2.62	5
EdgeJoiner	10.8	15	5	1.67	5.20
VoidSpace	1.6	3	3	0.80	1.36
VoidContainer	2.1	2	2	0.67	1.14

Tabla 4.5: Métricas de complejidad de código de la implementación de DELFIN.

4.2. Cobertura de código

La cobertura del código y la cobertura de los tests son métricas importantes para conocer qué porciones de código no han sido ejecutadas y determinar si hay suficientes tests unitarios. Un alto porcentaje de cobertura de código al ejecutar los tests reduce las posibilidades de errores no identificados. Esto va de mano con los resultados obtenidos en el análisis estático, porque las clases con mayor complejidad y mayor Type Rank necesitan ser testeadas más cuidadosamente, ya que fallas en esas clases pueden introducir vulnerabilidades al código. Para estas mediciones se trabaja con *gcov*[4] y *lcov*[5].

En una ejecución de código con datos artificiales y utilizando parámetros de filtro de vacíos por volumen y largo mínimo del arco más largo los resultados de ejecución de código se encuentran en la Tabla 4.7, con valores detallados por archivo en la Tabla 4.6.

Las clases VoidSpace y VoidContainer nunca son utilizadas. Para el resto de las clases siempre se ejecutan al menos el 66.7 % de las funciones. Se debe, por tanto, evaluar la utilidad de VoidSpace y VoidContainer al extender el software con el constructor de vacíos por densidad

[4] `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`
[5] `http://ltp.sourceforge.net/coverage/lcov.php`

de arcos, ya que la información entregada en el *output* será modificada y deberá incluir, entre otros, la densidad de cada vacío, lo que podría ser almacenado en VoidSpace.

VoidBuilder es una clase con 66.6 % de funciones ejecutadas y con un porcentaje bajo de líneas ejecutadas (48 %). Las líneas que no fueron ejecutadas corresponden a una función para escribir los resultados de vacíos encontrados. Esta función no se utiliza actualmente porque la información esta siendo entregada por EdgeJoiner, sin embargo podría ser aprovechada al agregar la nueva clase constructora de vacíos.

El total de cobertura por tests unitarios incluidos para DELFIN es de 33.7 % en relación a las líneas de código (ver Tabla 4.9). Solo 6 archivos del código fuente tienen funciones ejecutadas en los test unitarios, y en las clases Dictionary, PointDictionary y FacetDictionary el porcentaje de funciones ejecutadas no supera el 50 % (ver Tabla 4.8).

Si bien tener alto porcentaje de cobertura de código por los tests no garantiza que esté libre de defectos, se recomienda que el porcentaje de cobertura se encuentre entre 75-85 %, pero idealmente superior a 80 % para cerciorar la calidad del software mediante la detección y corrección de defectos [30].

A partir de los resultados de cobertura por test, junto con los resultados de complejidad e *importancia* por clase se concluye que las clases que necesitan más pruebas son FacetDictionary, PointDictionary, Facet y Edge. Conociendo la complejidad de cada método se sabe cuántos caminos independientes hay que probar, y por lo tanto cuántas pruebas unitarias agregar o crear, número que es mayor para las clases con alta complejidad como FacetDictionary.

Archivo	**Líneas**		**Funciones**	
DelfinBuilder.cpp	92.2 %	71 / 77	85.7 %	6 / 7
Dictionary.hpp	100.0 %	14 / 14	78.6 %	11 / 14
Edge.cpp	75.5 %	37 / 49	80.0 %	12 / 15
EdgeDictionary.cpp	95.1 %	39 / 41	87.5 %	7 / 8
EdgeDictionary.hpp	0.0 %	0 / 1	0.0 %	0 / 1
EdgeJoiner.cpp	100.0 %	58 / 58	100.0 %	6 / 6
Facet.cpp	86.7 %	85 / 98	78.9 %	15 / 19
FacetDictionary.cpp	96.4 %	134 / 139	92.3 %	12 / 13
PointDictionary.cpp	72.7 %	32 / 44	71.4 %	5 / 7
VoidBuilder.cpp	48.4 %	31 / 64	66.7 %	2 / 3
VoidBuilder.hpp	100.0 %	2 / 2	66.7 %	2 / 3
VoidContainer.cpp	0.0 %	0 / 25	0.0 %	0 / 7
VoidJoiner.cpp	97.3 %	73 / 75	66.7 %	2 / 3
VoidJoiner.hpp	100.0 %	2 / 2	66.7 %	2 / 3
VoidSpace.cpp	0.0 %	0 / 34	0.0 %	0 / 11
geom3d.cpp	64.0 %	16 / 25	75.0 %	3 / 4
main.cpp	85.7 %	36 / 42	100.0 %	1 / 1

Tabla 4.6: Cobertura de código para los archivos fuente de DELFIN.

	Alcance	Total	Cobertura
Líneas	630	790	79.7 %
Funciones	86	125	68.8 %

Tabla 4.7: Resumen de la cobertura de código para las líneas y funciones de DELFIN.

Archivo	Líneas		Funciones	
DelfinBuilder.cpp	0.0 %	0 / 77	0.0 %	0 / 7
Dictionary.hpp	85.7 %	12 / 14	35.7 %	5 / 14
Edge.cpp	71.4 %	35 / 49	73.3 %	11 / 15
EdgeDictionary.cpp	85.4 %	35 / 41	50.0 %	4 / 8
EdgeDictionary.hpp	0.0 %	0 / 1	0.0 %	0 / 1
EdgeJoiner.cpp	0.0 %	0 / 58	0.0 %	0 / 6
Facet.cpp	61.2 %	60 / 98	73.7 %	14 / 19
FacetDictionary.cpp	59.7 %	83 / 139	46.2 %	6 / 13
PointDictionary.cpp	61.4 %	27 / 44	42.9 %	3 / 7
VoidBuilder.cpp	0.0 %	0 / 64	0.0 %	0 / 3
VoidBuilder.hpp	0.0 %	0 / 2	0.0 %	0 / 3
VoidContainer.cpp	0.0 %	0 / 25	0.0 %	0 / 7
VoidJoiner.cpp	0.0 %	0 / 75	0.0 %	0 / 3
VoidJoiner.hpp	0.0 %	0 / 2	0.0 %	0 / 3
VoidSpace.cpp	0.0 %	0 / 34	0.0 %	0 / 11
geom3d.cpp	0.0 %	0 / 25	0.0 %	0 / 4

Tabla 4.8: Cobertura de código por los test de DELFIN.

	Alcance	Total	Cobertura
Líneas	252	748	33.7 %
Funciones	43	124	34.7 %

Tabla 4.9: Resumen de cobertura de código por los tests de DELFIN.

4.3. Análisis de escalabilidad

Se analiza la implementación de DELFIN sobre datos con n puntos en un cubo de $N \times N \times N$ ($N = 200h^{-1}$Mpc), usando un conjunto de datos de vacíos esféricos generados artificialmente. Los experimentos fueron repetidos para valores de $n = 1000 \times 2^0, 1000 \times 2^1, ..., 1000 \times 2^{10}$ usando un contraste de densidad[6] igual a 10.

El sistema utilizado consta de un procesador Intel Core i7-4710HQ de 4 núcleos, 16 (8x2) GB de memoria y Windows 10 Home versión 2004 como sistema operativo, ejecutando las

[6]Contraste de densidad es la relación entre el número de galaxias por unidad de volumen en el espacio exterior respecto al interior de los vacíos.

pruebas en WSL2 con Ubuntu 20.04. Para medir el consumo de memoria se utiliza la herramienta *massif* de Valgrind[22].

Tal como menciona Alonso R. [2] en su evaluación de desempeño del algoritmo (que corresponde a lo presentado en la figura 4.5), el tiempo se ajusta bien a la curva $O(n \log n)$. Al realizar la regresión lineal sobre los pares de tiempo respecto al número de puntos, se obtiene una pendiente $m = 1{,}29\text{e}{-03}$, con un coeficiente de correlación igual a $R^2 = 99{,}7\,\%$, y los residuos se encuentran, a lo más, a 42 segundos de los valores ajustados (figura 4.6).

En este trabajo, además de la medición de tiempo de ejecución, se mide cuánto demora cada etapa: la construcción de malla de tetraedros (figura 4.2), la ejecución del algoritmo DELFIN que construye los pre-vacíos (figura 4.3) y la unión de los pre-vacíos con el criterio de largo de arcos (figura 4.4). En todos los casos los tiempos se ajustan a una curva enelogarítmica.

El mayor porcentaje de tiempo de ejecución está determinado por el tiempo de construcción de los pre-vacíos el cual, según muestra la figura 4.7, aumenta a medida que se incrementa el número de puntos: para $n = 1000$ es el 79.5 % del tiempo total y para $n = 1000 \times 2^{10}$ es el 89 % del total. El tiempo de creación de la malla de tetraedros es siempre mayor al tiempo que el algoritmo tarda en unir a los pre-vacíos. En ambos casos el porcentaje del tiempo total que utilizan se va reduciendo.

El aumento del consumo máximo de memoria a medida que aumenta el número de puntos también es lineal (ver figura 4.8). La pendiente de la regresión lineal sobre los pares de número de puntos y el consumo de memoria es $m = 2{,}12\text{e}{-03}$ y el coeficiente de correlación es $R^2 = 99{,}9\,\%$. Respecto a los valores observados, el ajuste entrega residuos menores a 3 MiB cuando el número de puntos es 1000×2^8 o menor, y para 1000×2^9 y 1000×2^{10} puntos el residuo es de 39,9 MiB y 19,5 MiB respectivamente, que corresponde al 3,5 % y 0,9 % de la memoria observada.

El mayor consumo de memoria ocurre siempre cuando se ejecuta la construcción de prevacíos (por DelfinBuilder), y corresponde a la memoria asignada para los arcos de los tetraedros y su identificación. Durante toda la ejecución del análisis realizado por DelfinBuilder el consumo de memoria se mantiene relativamente cercano al máximo reportado. Cabe tener en cuenta que para efectos de este ejemplo, el número de arcos y tetraedros aumenta linealmente con respecto al número de puntos, lo que ayuda a explicar el comportamiento linealítmico de la memoria utilizada por DELFIN.

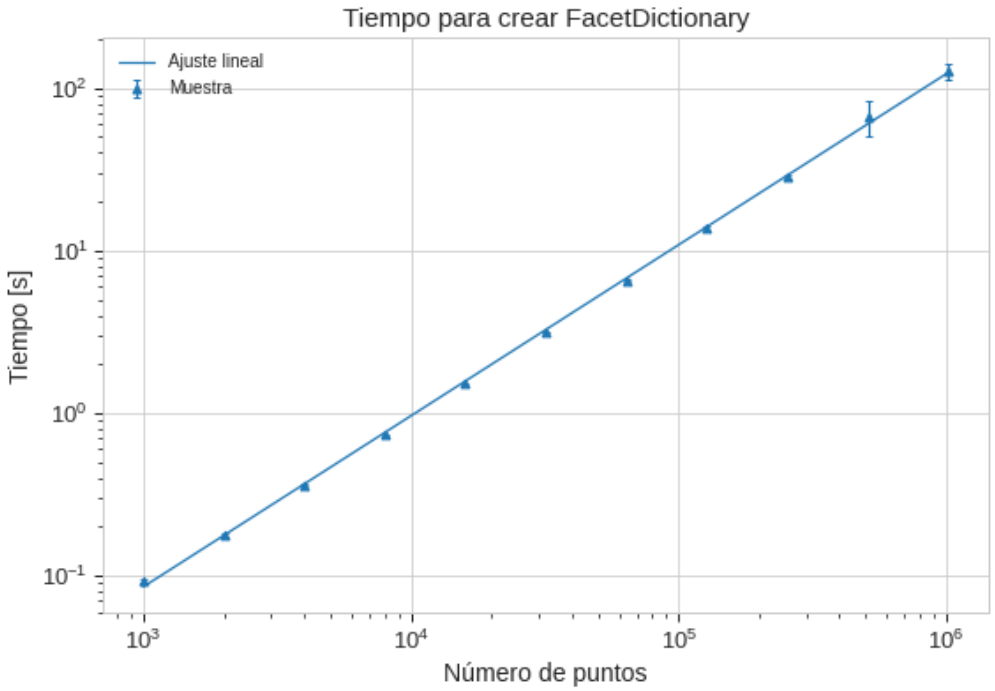


Figura 4.2: Tiempo que demora la creación de los tetraedros al ejecutar DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

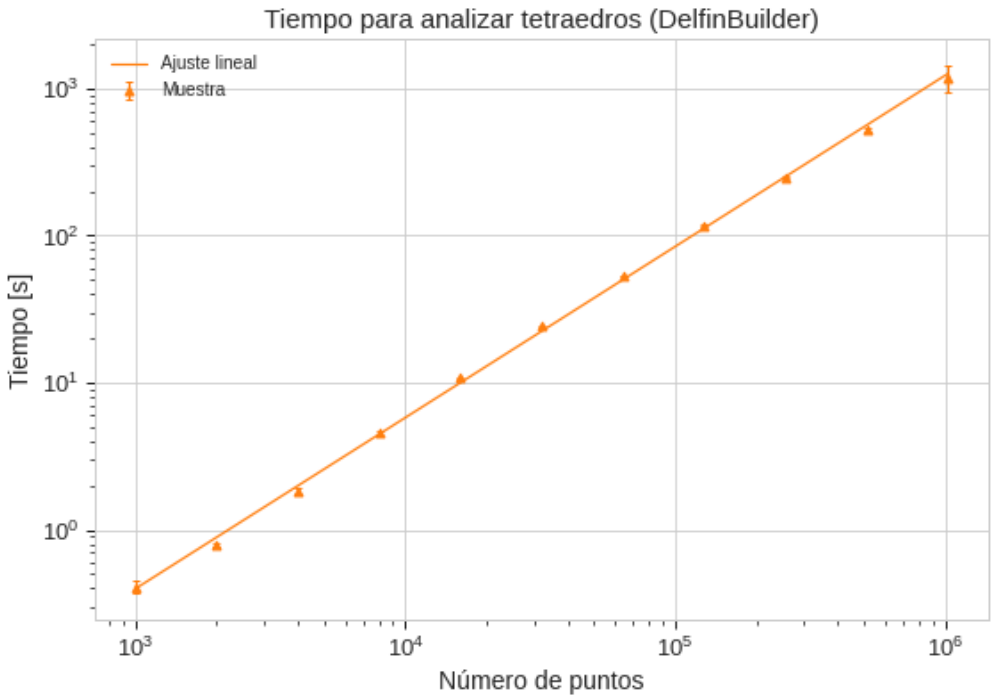


Figura 4.3: Tiempo que demora el análisis y creación de pre-vacíos al ejecutar DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

Figura 4.4: Tiempo que demora la unión de pre-vacíos al ejecutar DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

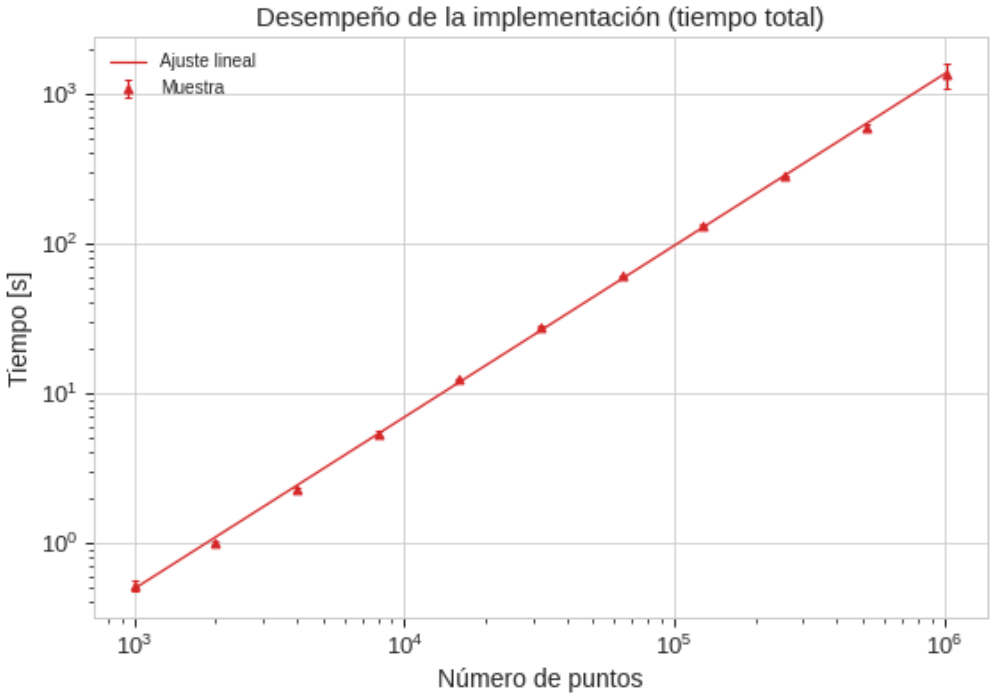


Figura 4.5: Tiempo total de ejecución de DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

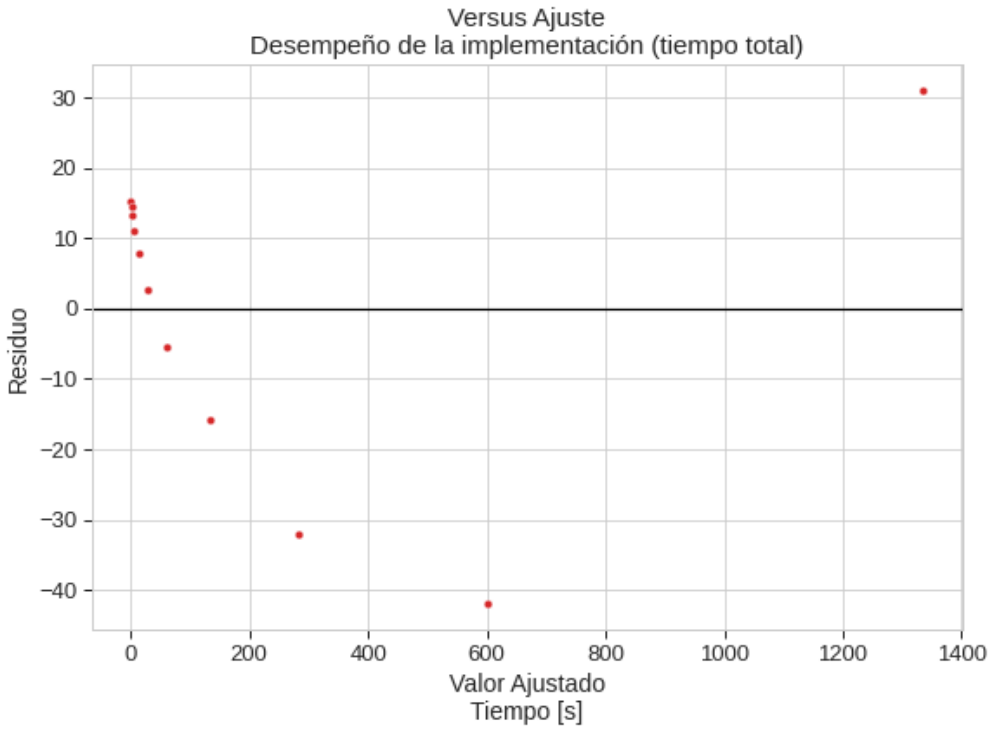


Figura 4.6: Residuos del ajuste lineal del tiempo total de ejecución de DELFIN respecto al número de puntos, sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

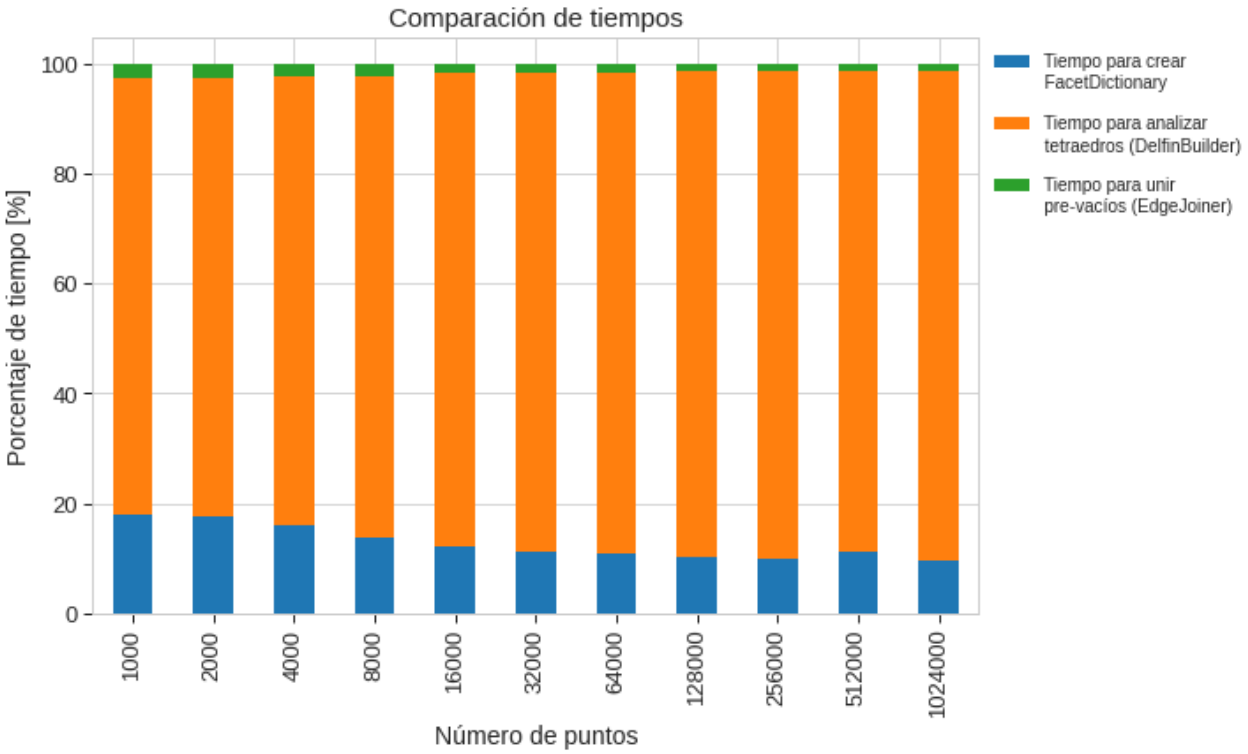


Figura 4.7: Comparación de los tiempos de ejecución de las distintas etapas de DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

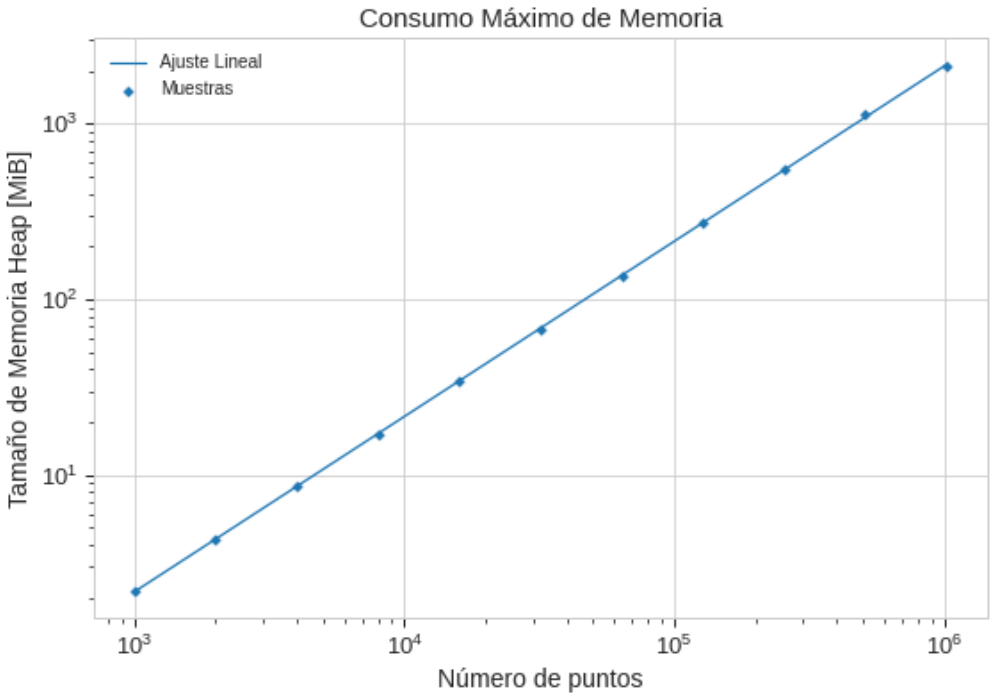


Figura 4.8: Consumo máximo de memoria al ejecutar DELFIN sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

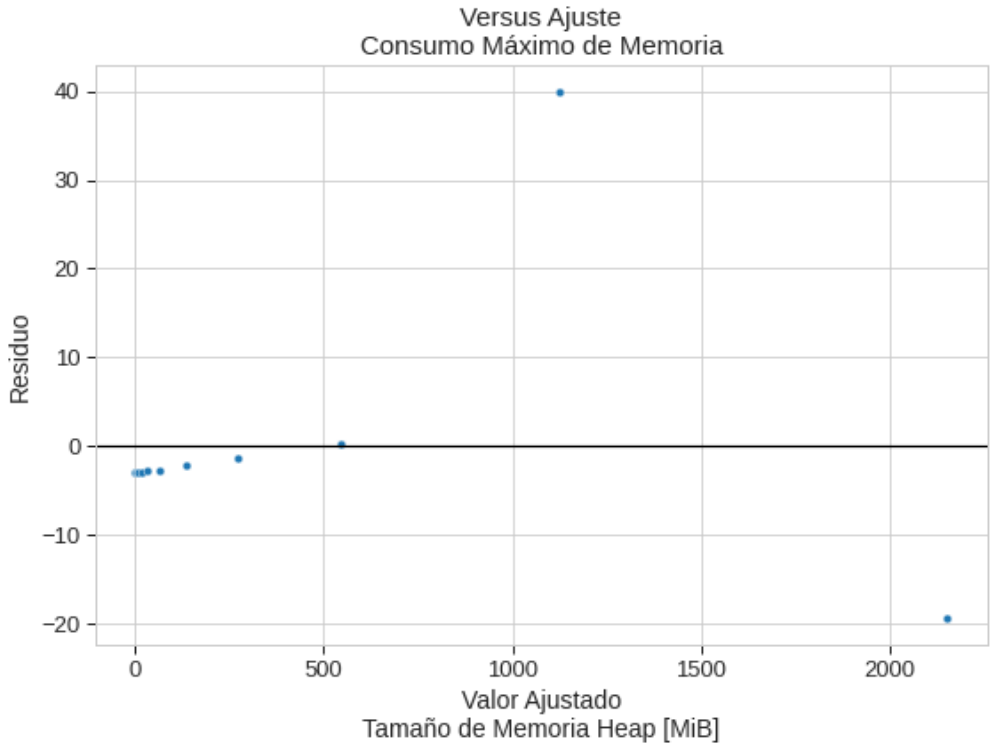


Figura 4.9: Residuos del ajuste lineal del consumo máximo de memoria en la ejecución de DELFIN respecto al número de puntos, sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000$ puntos.

4.4. Manejo de memoria

Además del aumento del uso de memoria a medida que se incrementa la cantidad de puntos de la malla, es crucial que exista una buena gestión de memoria, lo que es de particular importancia en lenguajes que no tienen *recolector de basura* como C++. Con la herramienta *Memcheck* de *Valgrind* se evalúa la existencia de errores relacionados a fugas de memoria en DELFIN.

```
==518==
==518== HEAP SUMMARY:
==518==     in use at exit: 3,555,908 bytes in 15,671 blocks
==518==   total heap usage: 84,429,986 allocs, 84,414,315 frees, 2,449,054,612 bytes allocated
==518==
==518== LEAK SUMMARY:
==518==    definitely lost: 1,559,116 bytes in 7,836 blocks
==518==    indirectly lost: 1,980,408 bytes in 7,834 blocks
==518==      possibly lost: 16,384 bytes in 1 blocks
==518==    still reachable: 0 bytes in 0 blocks
==518==         suppressed: 0 bytes in 0 blocks
==518==
==518== For lists of detected and suppressed errors, rerun with: -s
==518== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

Figura 4.10: Resultados de *Memcheck* sobre DELFIN.

En la figura 4.10 se muestra el resumen de los resultados del análisis de *Memcheck* para una ejecución de DELFIN sobre un conjunto de 52236 puntos. Se detectan tres errores, en donde hay seguridad de la pérdida de 3539524 bytes y otro error que indica la pérdida de 16384 bytes debido a la un puntero ubicado dentro de una cadena de uno o más punteros.

De los resultados en detalle se concluye que las fugas ocurren por un mal manejo de memoria en las clases DelfinBuilder y FacetDictionary. Eventualmente, estas fugas de memoria pueden causar problemas, sobre todo si se trabaja en un sistema con una cantidad limitada de memoria y el programa aumenta constantemente el uso de memoria. Debido a que DELFIN puede trabajar con muestras de millones de galaxias (por ejemplo, la muestra del proyecto Aspen-Amsterdam[10] tiene 12 millones de partículas), es de gran importancia que las fugas de memoria sean eliminadas.

Capítulo 5

Solución

En los capítulos 3 y 4 se describieron los problemas de los resultados y de la implementación DELFIN. En esta sección se presenta la nueva versión, DELFIN++, la cual basa la búsqueda de vacíos a partir de la densidad de elementos, y que incluye información para caracterizar a los vacíos, en posición, forma y otros descriptores de importancia.

5.1. Métricas de densidad

Las métricas de densidad incorporadas se definen a partir de los elementos geométricos entregados por la tetraedrización de Delaunay. Los elementos de los que se tiene información son los puntos, arcos y tetraedros, para los cuales se pueden calcular sus propiedades métricas como longitud para arcos, áreas de superficie, volumen y centroide de tetraedros.

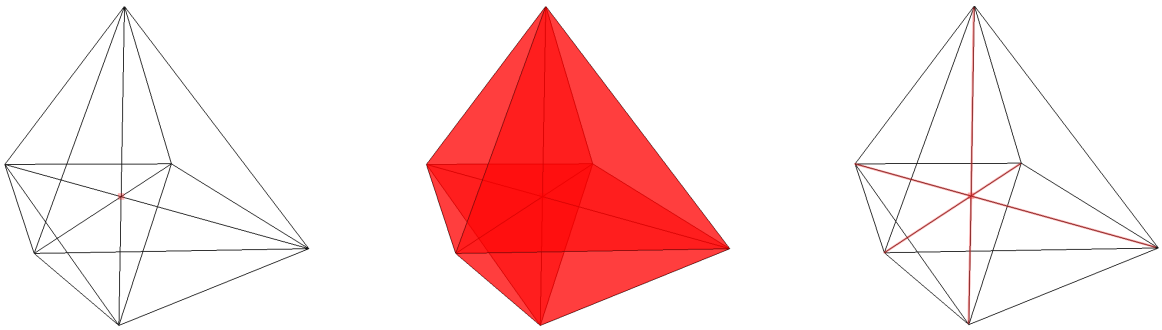
Se desarrollan cuatro métricas de densidad, dos correspondientes a densidad de puntos de la tetraedrización y dos de densidad de arcos. No solo se necesita la información de densidad como un descriptor de muestra, sino que el algoritmo de búsqueda de vacíos se basa en estas métricas para la obtención de resultados, así que se implementan diferentes métricas para evaluar su desempeño en la recuperación de vacíos.

La primera métrica corresponde a la relación del volumen que ocupan los tetraedros a los que pertenece el punto. Sea s un punto en el conjunto S en un espacio 3-dimensional, D la triangulación de Delaunay de S y $T(D, s)$ el conjunto de tetraedros de la triangulación D que tienen a s como uno de sus vértices, entonces esta métrica se define como

$$\frac{1}{\sum_{t \in T(D,s)} Vol(t)}$$

donde $Vol(t)$ es el volumen del tetraedro t.

La segunda métrica, también vinculada a la densidad de puntos, corresponde a la relación del largo de los arcos a los que pertenece el punto. Sea $E(D, s)$ el conjunto de aristas de la



(a) (b) (c)

Figura 5.1: Elementos considerados para las métricas de densidad de puntos. (a) La malla de tetraedros, con el punto para el que se calculan la densidades destacado en rojo. (b) En rojo los tetraedros considerados para calcular la densidad por volumen del punto. (c) Se destacan los arcos considerados para calcular la densidad por longitud de aristas del punto.

triangulación de Delaunay D que está limitada por el punto s, esta métrica corresponde a

$$\frac{1}{\sum_{e \in E(D,s)} Len(e)}$$

donde $Len(e)$ es la longitud de la arista e.

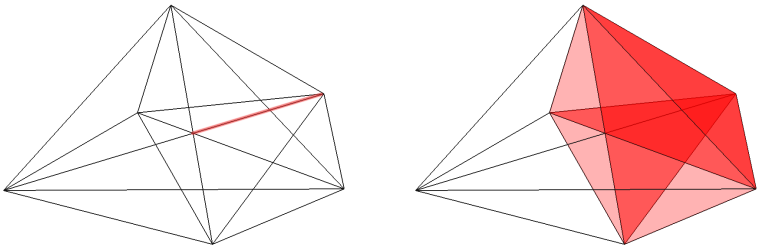
En la figura 5.1 se destacan los elementos a partir de los que se calcula la densidad de un punto. Cuando se calcula la densidad por volumen, se considera todo el espacio en el cual no existe otro punto (figura 5.1 (b)). A diferencia de la teselación de Voronoi, en donde un poliedro o región se asocia a un solo punto del conjunto de datos, los tetraedros de la teselación de Delaunay se asocian a cuatro, por lo que el volumen de un 3-símplex se considera más de una vez al calcular la densidad de los puntos. La densidad por longitud de arco (figura 5.1 (c)) considera las distancias a los vecinos más cercanos del punto. Al igual que con los tetraedros, la longitud de un arco se considera más de una vez.

Se implementan dos métricas de densidad de aristas de la triangulación. Sea e una arista de la triangulación de Delaunay D y $T(D, e)$ el conjunto de tetraedros de D que tienen a e como una de sus aristas (ver figura 5.2), se define una métrica que depende únicamente del volumen de estos tetraedros:

$$\frac{1}{\sum_{t \in T(D,e)} Vol(t)}$$

y otra similar que es la relación entre la longitud de la arista y la suma del volumen de sus tetraedros:

$$\frac{Len(e)}{\sum_{t \in T(D,e)} Vol(t)}$$



(a) (b)

Figura 5.2: Elementos considerados para las métricas de densidad de arcos. (a) La malla de tetraedros, destacando en rojo el arco para el que se calcula la densidad. (b) Se destacan los tetraedros asociados al arco, cuyo volumen se suma para calcular la densidad.

5.2. Algoritmo

El algoritmo depende directamente de las métricas de densidad, y se basa en agrupar los tetraedros que envuelven al elemento mientras este tenga una densidad los suficientemente baja. Existen dos versiones del algoritmo, una trabaja con densidad de puntos y la otra con densidad de aristas de la teselación. Desde ahora se hará referencia a la versión que trabaja con densidad de puntos, pero la lógica, a diferencia de la implementación, es equivalente.

Primero, se obtiene la teselación de Delaunay del conjunto de datos y se ordenan los puntos por densidad ascendente. Luego, por cada punto se evalúan los tetraedros a los que pertenece para que sean parte de un vacío. Si ninguno de estos tetraedros es parte de un vacío, se crea uno nuevo con todos ellos. Si entre los tetraedros hay uno o más que son parte del mismo vacío v, el resto de tetraedros sin vacío asociado se une a v. El tercer caso es cuando al menos dos de los tetraedros del punto son parte de vacíos diferentes, aquí se unen todos los tetraedros de todos esos vacíos a el primero y el resto de vacíos se descartan.

Secuencialmente se evalúan los puntos de la forma mencionada, hasta que la densidad del punto sea mayor que la densidad umbral. Esta densidad debe ser lo suficientemente baja para que no ocurra que todo el volumen definido por los datos se particione en poliedros y que el resultado final sea similar a la tetraedrización de Delaunay.

A las regiones resultantes se les aplican distintos filtros. El primer método consiste en eliminar todos los vacíos cuyo volumen V es menor a un umbral V_{min}. El segundo es un método que consiste en filtrar vacíos según la calidad de los tetraedros en su superficie. La métrica utilizada para medir la calidad de tetraedros es Aspect Ratio Gamma (ARG)[21]. Cuando se computa ARG se detectan los casos más comunes de tetraedros inválidos y de mala calidad: planos, agujas e invertidos. El ARG se define de la siguiente forma:

$$R = \left(\frac{1}{6} \sum_{\mathrm{i}=0}^{5} ||l_{\mathrm{i}}||^2 \right)^{\frac{1}{2}} \Rightarrow \mathrm{ARG} = \frac{R^3\sqrt{2}}{12 \cdot V} = \frac{R^3}{8{,}48528 \cdot V},$$

donde l_i, $\forall i = 0, ..., 5$ son los largos de sus arcos y V es el volumen del tetraedro. El rango de ARG es $[1, \infty)$, y el rango óptimo para que un tetraedro sea de calidad es $[1, 3]$.

Algoritmo 1: Algoritmo DELFIN++ de búsqueda de vacíos por densidad de puntos

```
Leer la teselación de Delaunay;
Leer densidad_umbral;
Leer volumen_umbral;
Calcular densidad por punto;
Ordenar puntos por densidad ascendente;
lista_vacios = ∅;
foreach punto p en la lista de puntos ordenada do
    if la densidad de p es mayor a densidad_umbral then
        break;
    end
    nuevo_vacio = True;
    Obtener los tetraedros t_1, ..., t_n a los que pertenece p;
    tetraedros_p = [t_1, ..., t_n] ;
    foreach tetraedro t en tetraedros_p do
        if t pertenece a un vacío v de lista_vacios then
            foreach tetraedro r en tetraedros_p do
                if r no es parte de un vacío then
                    Agregar r al vacío v;
                else if r pertenece al vacío w, w ∩ v = ∅ then
                    v = v ∪ w;
                    lista_vacios = lista_vacios\w;
            end
            nuevo_vacio = False;
            break;
        end
    end
    if nuevo_vacio then
        lista_vacios = lista_vacios ∪ poliedro(tetraedros_p);
    end
end
Eliminar de lista_vacios a aquellos vacíos con volumen menor a
 volumen_umbral, con un arco en el borde de la cobertura convexa o con 90 % de
 tetraedros en su superficie de mala calidad;
```

La regla utilizada para este segundo método para filtrar vacío es si al menos el 90 % de la superficie se obtiene por tetraedros de mala calidad, con ARG > 3, entonces el vacío se descarta. En particular este método se desarrolla para tratar con vacíos muy cercanos al borde de la cobertura convexa, pero no tienen arcos que toquen tal borde. En una tetraedrización de Delaunay, los tetraedros cercanos a los bordes tienden a ser alargados o planos, ocupando más superficie que los tetraedros interiores. Si un vacío es formado por estos tetraedros debe ser descartado porque se altera la información de forma y posición.

Como continuación de este método, se incluye un post proceso de vacíos que tienen entre 65 % y 90 % de su superficie de tetraedros de mala calidad. Este post proceso también se incluye para tratar con vacíos que se encuentran cercanos al borde pero a diferencia del caso anterior, el menor porcentaje de mala superficie indica una menor deformación y está principalmente dado por una sobre extensión del vacío, por lo que se realiza una disminución de su tamaño eliminando los tetraedros de mala calidad de su superficie.

La figura 5.3 es una representación en 2 dimensiones de la secuencia seguida en la generación de vacíos sobre una muestra aleatoria de 150 puntos, en el cual se utiliza la métrica de densidad de puntos según el área de sus triángulos (equivalente al volumen en 3-d) y el valor utilizado como densidad umbral es el 20 % del promedio de la densidad de puntos. La figura 5.3a muestra el conjunto inicial de puntos. La figura 5.3b muestra la triangulación de Delaunay de los puntos. En la figura 5.3c se observa la detección del punto menos denso (en rojo) y sus triángulos asociados, los cuales son la base para el primer vacío. En la figura 5.3d se detecta el segundo punto menos denso, del cual dos triángulos pertenecen al vacío recién creado, por lo que el resto de sus triángulos se agregan al mismo vacío. Lo mismo ocurre para el punto destacado en la figura 5.3e. Las figuras 5.3f y 5.3g muestra casos en que ninguno de los triángulos del punto procesado es parte de un vacío, así que en ambos casos se genera uno nuevo. En la figura 5.3h se procesa un nuevo punto, para el que se anexan sus triángulos al vacío de color azul, debido a que dos de estos pertenecían previamente a tal vacío. Finalmente, en la figura 5.3i se muestra el último punto procesado antes de exceder el umbral de densidad, cuyos triángulos se agregan al primer vacío generado.

Para obtener una versión resumida del algoritmo, consultar el algoritmo 1.

5.3. Información de forma y tamaño de vacíos

Un problema tratado en este trabajo es la dificultad de compresión de los resultados y la falta de información que permita identificar los vacíos, sus galaxias y características. Entre los datos previamente entregados por DELFIN se encuentra el centroide del vacío, calculado como:

$$X = \frac{1}{\sum_k V_k} \sum_k \mathbf{x}_k V_k, \tag{5.1}$$

donde V_k es el volumen del tetraedro k en el vacío, y $\mathbf{x}_k$ es el centroide del mismo tetraedro. Además de esto se entrega la información del volumen V del vacío, el cual se obtiene sumando el volumen de todos los tetraedros que son parte de él. A partir de V se obtiene el radio equivalente de una esfera con el mismo volumen:

$$R = \sqrt[3]{\frac{3}{4\pi} V}.$$

Se agrega la información de densidad interna de los vacíos, la cual es independiente de las métricas calculadas sobre los elementos geométricos de la tetraedrización y del algoritmo de búsqueda. Para este resultado se calcula la cantidad de galaxias o puntos interiores, los cuales corresponden a los puntos que tienen todos sus tetraedros en el vacío. La densidad interna se define como:

$$\delta = N_{interior}/V, \tag{5.2}$$

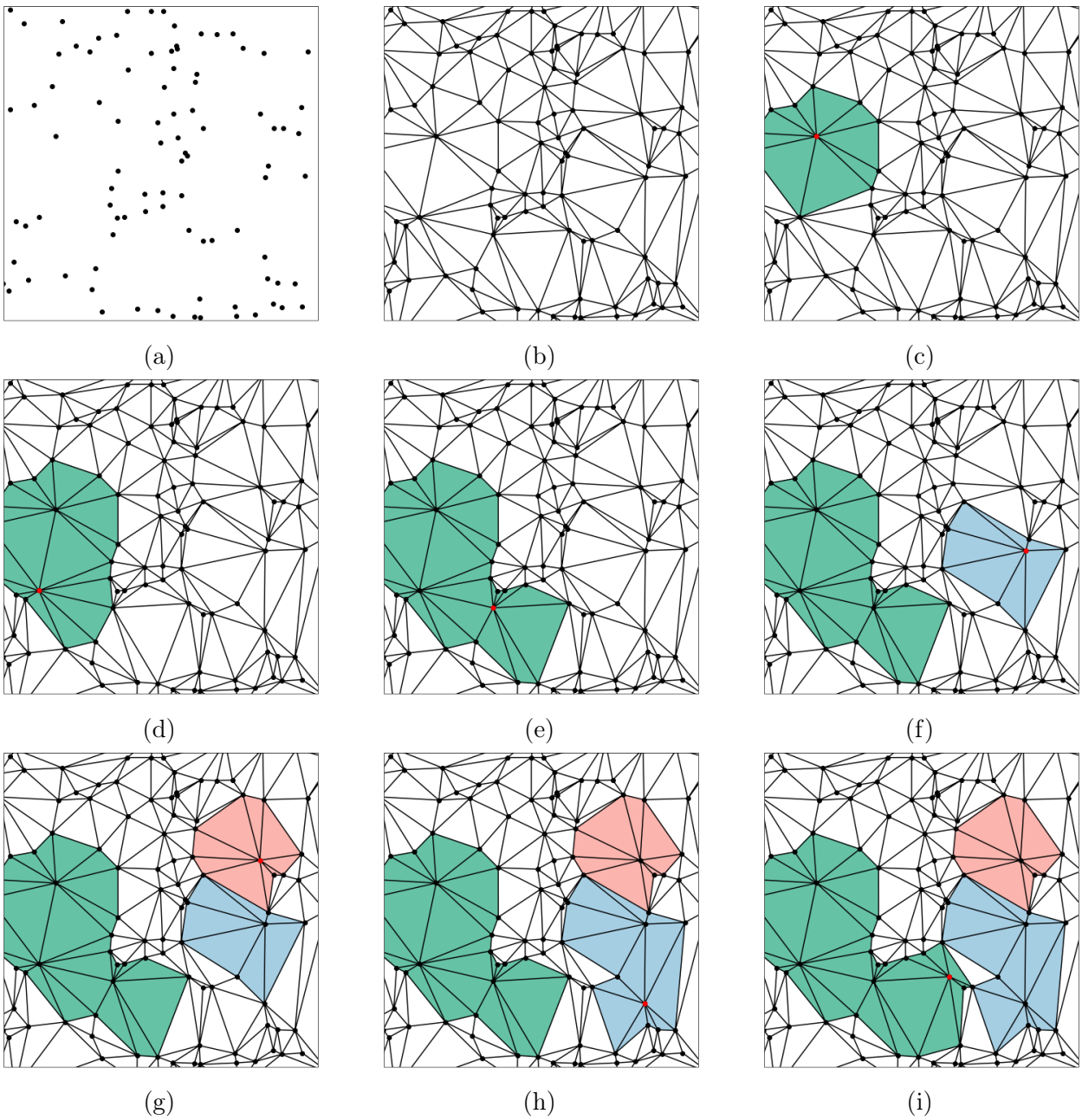


Figura 5.3: Representación en 2-d del paso a paso de la búsqueda de vacíos de DELFIN++ utilizando métricas de densidad de puntos. Se muestra el punto evaluado y los símplices que se agregan a un vacío.

donde $N_{interior}$ es el número de puntos interiores.

En DELFIN++ también se computa información de la forma de los vacíos, para lo cual se utilizan todos las galaxias miembros y se construye el tensor de inercia:

$$M_{xx} = \sum_{\mathrm{i}=1}^{N_p} (y_{\mathrm{i}}^2 + z_{\mathrm{i}}^2) \tag{5.3}$$

$$M_{xy} = -\sum_{\mathrm{i}=1}^{N_p} x_{\mathrm{i}} y_{\mathrm{i}}, \tag{5.4}$$

donde N_p es el número de galaxias miembros, x_{i}, y_{i}, y z_{i} son las coordenadas de la galaxia i relativas al centro de masas del vacío, el cual se computa considerando un sistema de masas puntuales:

$$\mathbf{R} = \frac{1}{N_p} \sum_{\mathrm{i}=1}^{N_p} r_{\mathrm{i}}, \tag{5.5}$$

donde r_{i} es la posición de la galaxia i. El resto de las componentes del tensor de inercia se obtienen por permutación cíclica. Luego de calcular el tensor, sus valores propios (λ_1, λ_2 y λ_3) y sus vectores propios correspondientes ($\mathbf{v}_{\lambda_1}$, $\mathbf{v}_{\lambda_2}$ y $\mathbf{v}_{\lambda_3}$ respectivamente) se obtiene la elipticidad:

$$\varepsilon = 1 - \left(\frac{\lambda_1}{\lambda_3}\right)^{1/4}, \tag{5.6}$$

donde λ_1 y λ_3 son los valores propios más pequeño y más grande respectivamente.

Con estos datos se estructuran los distintos archivos que contienen los resultados obtenidos por DELFIN++, basándose en el formato propuesto por VIDE. A continuación se presentan los encabezados de los mismos.

Propiedades y posición

ID vacío	x_{centro}	y_{centro}	z_{centro}	volumen (V)	radio (R)	densidad interna (δ)

Este archivo incluye información posicional del vacío, indicando las coordenadas de su centroide X: x_{centro}, y_{centro} y z_{centro} (ec. 5.1). El volumen, radio de la esfera con volumen equivalente (ec. 5.3) y densidad interna (ec. 5.2) se calculan como se mencionó previamente.

Forma

ID vacío	ε	λ_1	λ_2	λ_3	$\mathbf{v}_{\lambda_1}x$	$\mathbf{v}_{\lambda_1}y$	$\mathbf{v}_{\lambda_1}z$	$\mathbf{v}_{\lambda_2}x$	$\mathbf{v}_{\lambda_2}y$	$\mathbf{v}_{\lambda_2}z$	$\mathbf{v}_{\lambda_3}x$	$\mathbf{v}_{\lambda_3}y$	$\mathbf{v}_{\lambda_3}z$

Respecto a la forma del vacío se incluye su elipticidad ε, los valores y vectores propios del tensor de inercia. En esta notación, $\mathbf{v}_{\lambda_1}x$ es la coordenada x del autovector correspondiente al autovalor λ_1.

Galaxias

ID vacío	ID galaxia	x	y	z	densidad

Por último, se incluye información de los miembros de los vacíos, contando galaxias internas y aquellas en el borde del vacío. Esto permite reconstruir los vacíos si es que fuera necesario, tanto en una triangulación de Delaunay, como en diagramas externos que puedan construirse a partir del conjunto de datos.

5.4. Implementación

La implementación del algoritmo DELFIN en 3-d está escrita en C++, por lo que todas las implementaciones de extensiones y pruebas se realizan en ese lenguaje. Para obtener la teselación de Delaunay de los puntos se utiliza *qdelaunay*[7], una herramienta que utiliza el algoritmo quickhull para producir el cierre convexo de los datos. Proyecta los datos a un paraboloide, levantándolos a la suma de los cuadrados de las coordenadas. Las regiones de Delaunay se obtienen al descartar la dimensión extra del cierre convexo de los datos proyectados, manteniendo la información de vecinos y aristas.

La versión previa del algoritmo corre en tiempo $O(t \log t)$ en el número de tetraedros t: por motivos de eficiencia, en vez de trabajar directamente con tetraedros, se ordenan las aristas e por largo [2]. En cada teselación de Delaunay en 3-d se tiene que $3 + 3t \geq e$, y respecto al número de vértices n, t es $O(n^2)$. Sin embargo, en la práctica las teselaciones de Delaunay tienen un número lineal de tetraedros.

Para calcular la densidad de puntos por volumen de tetraedros se visita cada tetraedro (t repeticiones), y el resto de las métricas se implementa visitando cada arista, calculando su largo y/o volumen de tetraedros (e repeticiones). Luego, el algoritmo continúa ordenando los elementos por densidad ascendente, lo que se hace en tiempo $O(n \log n)$. Posterior a esto se procede a visitar los vértices o aristas una vez cuando se evalúan de menor a mayor densidad. Al trabajar con vértices, cada tetraedro puede ser visitado dos veces por cada uno de sus vértices, una vez para ver si pertenece a un vacío y la segunda para agregarlo a un vacío, así que puede visitarse a lo más 8 veces ($8t$ repeticiones). De la misma forma, al trabajar con aristas, cada tetraedro puede visitarse a lo más 12 veces ($12t$ repeticiones, 2 por cada una de sus 6 aristas). En la práctica no se realizan tantas repeticiones debido a que suele utilizarse una baja densidad umbral, deteniendo el proceso mucho antes de evaluar la totalidad de los elementos. Para descartar los vacíos se trabaja con los tetraedros, visitándolos una cantidad constante de veces, tanto para calcular su volumen, computar su ARG, evaluar si se encuentra en el borde del dominio de los datos o eliminarlo del vacío (si se utiliza el post-proceso propuesto). Si bien en el peor caso, el número de tetraedros es n^2 con respecto al número de vértices n, bajo el supuesto de que el número de tetraedros es lineal respecto a n, la complejidad del algoritmo es de $O(n \log n)$.

5.4.1. Estructura

La figura 5.4 muestra el diagrama de clases de la implementación de DELFIN++, que contiene las extensiones descritas previamente.

Se comienza realizando cambios en los nombres de la clases Facet y FacetDictionary a Tetrahedron y TetrahedronDictionary respectivamente. Este cambio se realiza porque lo que estas clases representan son tetraedros, y no un *facet* o faceta, que en 3 dimensiones corresponde a un polígono inscrito en un plano cuyas esquinas son vértices de un poliedro y no es una cara. Además, en la clase Tetrahedron se incluye la implementación del cálculo de Aspect Ratio Gamma.

Las métricas de densidad se implementan teniendo a la clase abstracta ElementDensity como base. De esta se derivan otras dos clases abstractas: PointDensity y EdgeDensity. Se

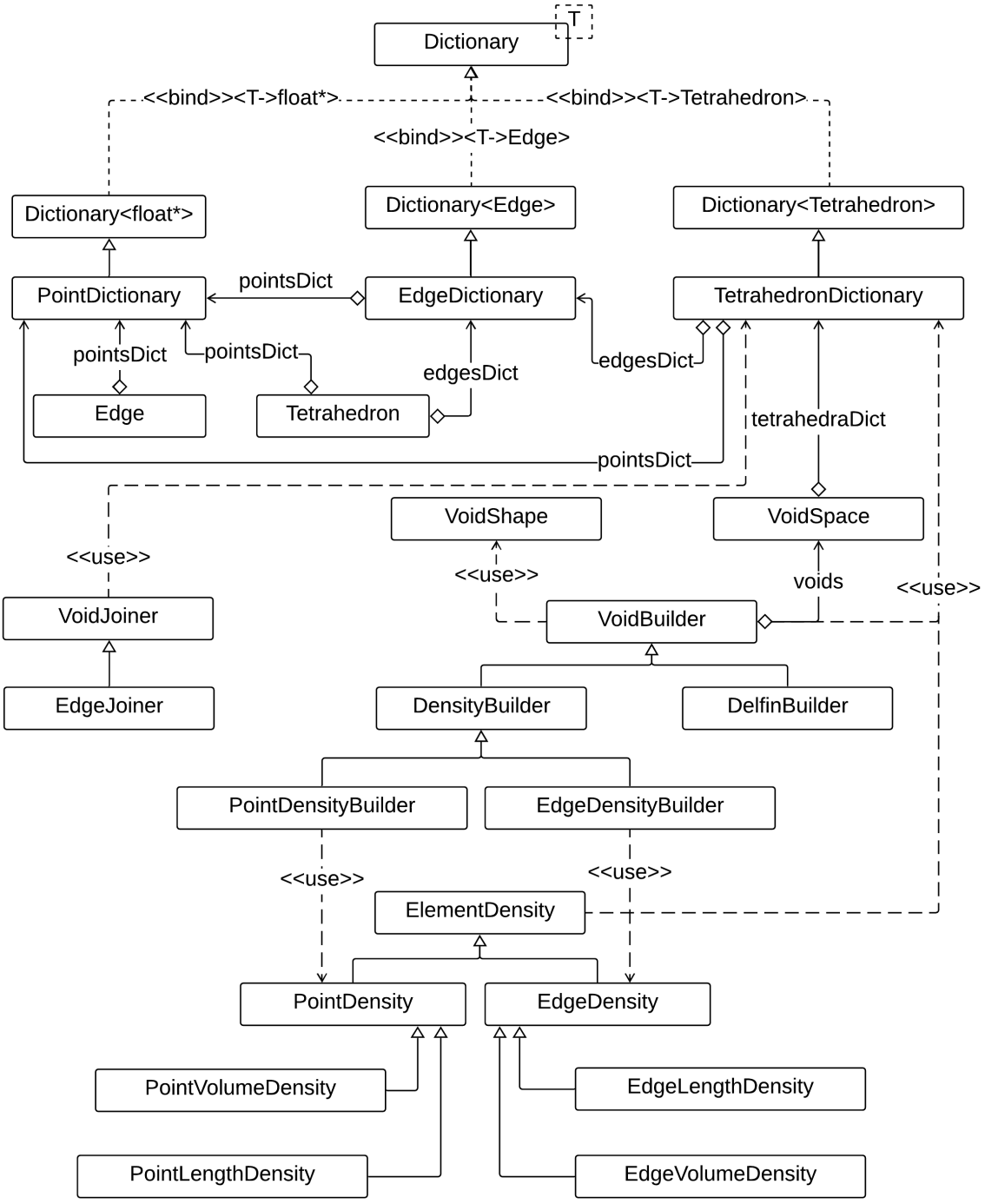


Figura 5.4: Diagrama de clases de la implementación de DELFIN++.

implementan dos clases no-abstractas de PointDensity, la clase PointVolumeDensity y la clase PointLengthDensity, que calculan la densidad de puntos por volumen de sus tetraedros y por largo de sus aristas respectivamente. De la misma forma, se implementan dos subclases de EdgeDensity, la clase EdgeVolumeDensity y la clase EdgeLengthDensity, en las que se calcula la densidad de aristas por volumen de tetraedros y por largo de arista relativo al volumen de los tetraedros respectivamente. Tanto en PointDensity como en EdgeDensity se implementan los métodos que ordenan los respectivos elementos por densidad ascendente.

Las clases encargadas de la búsqueda de vacíos son aquellas que heredan de la clase abstracta VoidBuilder. La clase DelfinBuilder es la implementada previamente, ordena los tetraedros por longitud descendente de su arista más larga y construye candidatos a vacíos, deteniendo la búsqueda en una longitud umbral de arista y filtrándolos por volumen. Luego, en un objeto de una clase no-abstracta de VoidJoiner se reciben los candidatos y se unen por algún criterio. El único método de unión de vacíos implementado es por longitud de arista compartida, el cual se encuentra contenido en EdgeJoiner.

La implementación de la búsqueda de vacíos por densidad se encuentra en las subclases de DensityBuilder, PointDensityBuilder y EdgeDensityBuilder, las cuales utilizan las métricas de PointDensity y EdgeDensity para obtener los elementos ordenados por densidad y agrupar tetraedros en vacíos. El filtro de vacíos, que es el mismo para ambas clases, se implementa en DensityBuilder.

En las clases derivadas de DensityBuilder, el manejo de vacíos se hace mediante objetos de la clase VoidSpace, la cual contiene la implementación del cálculo de propiedades como el volumen, densidad interna y porcentaje de superficie con tetraedros de mala calidad según ARG.

Para el cálculo de forma de los vacíos se implementa la clase VoidShape, en la cual se computa el tensor de inercia y se calculan sus autovectores y autovalores utilizando el método de Jacobi implementado en la biblioteca `jacobi_pd`[1]. Esta clase es utilizada por un constructor de vacíos luego de obtener los resultados y se comienzan a generar los reportes de información.

5.4.2. Flujo

El flujo seguido en el programa se muestra en la figura 5.5.

Los primeros pasos corresponden a cargar la teselación de Delaunay en objetos de la implementación, particularmente en diccionarios de puntos, aristas y tetraedros. Luego, si se quiere hacer una búsqueda de vacíos por densidad de elementos se utiliza un objeto de la clase DensityBuilder, en el caso contrario se utiliza un DelfinBuilder.

Al seguir el flujo por la búsqueda de vacíos ordenando los tetraedros según su arista más larga, en la unión de candidatos se utiliza un objeto de la clase VoidJoiner, a diferencia de la construcción de vacíos por densidad de elementos donde los vacíos no se presentan fragmentados, por lo que no hay un post-proceso de unión de candidatos.

[1] Andrew Jewett (2020) jacobi_pd [Source Code]. `https://github.com/jewettaij/jacobi_pd`

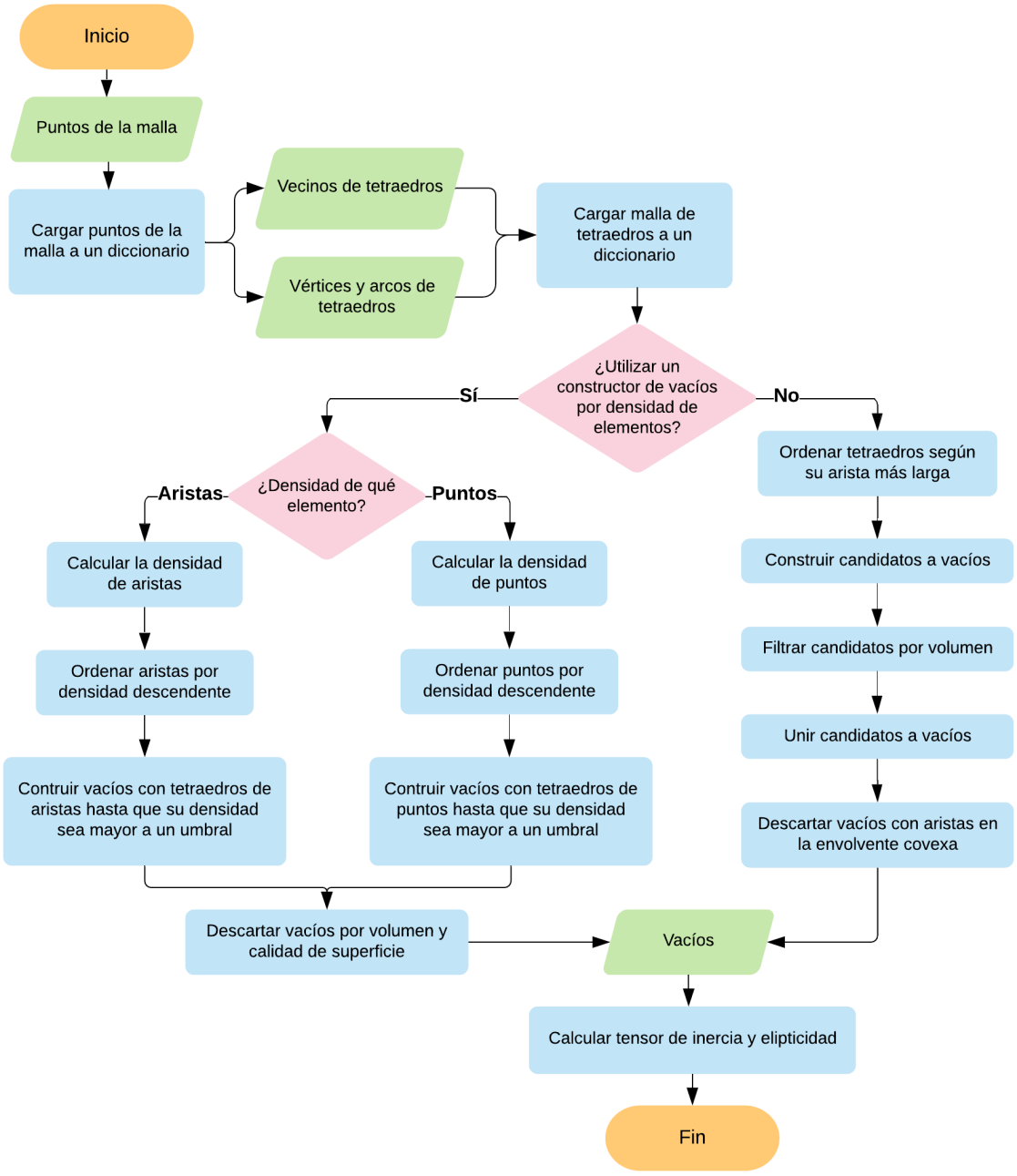


Figura 5.5: Flujo de DELFIN++ para generar vacíos.

Luego de reconocer los vacíos se calcula y reporta la información de los mismos, en particular el cálculo de forma es el último paso y como se mencionó anteriormente, esto se lleva a cabo en la clase VoidShape.

5.4.3. Modo de uso

Previo a la ejecución del programa debe completarse la instalación de *qhull* para generar los archivos que contienen los datos de la teselación, en específico se utilizan los archivos de vértices y vecinos de los tetraedros.

El programa se ejecuta de la siguiente forma:

```
$ delfin nombre_puntos [-h] [-b] [-q] [-vVOLUMEN] [-pPORCENTAJE] [-eLARGO]
    [-mMETODO] [-tCORTE] [-w]
```

donde los argumentos opcionales están encerrados en corchetes, y `nombre_puntos` es el nombre del archivo -sin la extensión del fichero- que contiene los puntos con los que se genera la triangulación. La implementación actualmente solo acepta archivos con la extensión '.dat'.

Los argumentos opcionales se describen en la tabla 5.1. Esta información también puede obtenerse al ejecutar el programa con el argumento `-h`.

Argumento	Descripción
`-h`	Ayuda: Muestra descripción y modo de uso del programa
`-b`	Si se utiliza, no se descartan los vacíos con arcos en el borde de la envolvente convexa
`-q`	Si se utiliza, se descartan los tetraedros de mala calidad en la superficie de vacíos con alto porcentaje de su área cubierta por estos tetraedros
`-vVOLUMEN`	Volumen mínimo por el cual filtrar los vacíos o candidatos a vacíos si se utiliza el constructor por largo de aristas
`-pPORCENTAJE`	Aplica un porcentaje de volumen mínimo que debe cumplir un tetraedro en relación al volumen del vacío para incluirlo, utilizado por el constructor por largo de aristas
`-eLARGO`	Aplica un largo de arista mínimo para generar nuevos vacíos en el constructor por largo de aristas
`-mMETODO`	La ID del método de búsqueda de vacíos. `METODO` debe reemplazarse con una de las siguientes IDs: -0: Constructor por densidad de puntos según el volumen de sus tetraedros -1: Constructor por densidad de puntos según el largo de sus aristas -2: Constructor por densidad de aristas según el volumen de sus tetraedros -3: Constructor por densidad de aristas según su largo y el volumen de sus tetraedros -4: Constructor por largo de aristas
`-tCORTE`	Porcentaje de datos evaluados para generar vacíos en los constructores por densidad de elementos
`-w`	Si se utiliza, el valor de corte para evaluar elementos es un porcentaje de la densidad promedio de los elementos.

Tabla 5.1: Argumentos para DELFIN++.

El archivo con los puntos tiene el formato utilizado por *qhull*. La primera línea contiene el número de dimensiones, que para esta aplicación debe ser 3. La segunda línea contiene n, el número de puntos en el archivo. Las siguientes n líneas representan las coordenadas de cada punto expresadas como una tupla (x, y, z) de números separados por un espacio.

En el mismo directorio que se encuentra el archivo de puntos se deben encontrar los archivos generados por *qhull* que indican los vértices y vecinos de cada tetraedro. Estos

archivos deben tener por nombre `nombre_puntos_vertex` y `nombre_puntos_neighbours` respectivamente, con la misma extensión que el archivo con los puntos. Para obtener estos archivos se ejecutan las siguientes líneas de comando:

```
$ qdelaunay TI nombre_puntos.dat Fv Qt TO nombre_puntos_vertex.dat
$ qdelaunay TI nombre_puntos.dat Fn Qt TO nombre_puntos_neighbours.dat
```

Luego de ejecutar el programa se retornan 5 archivos en el mismo directorio. Estos archivos corresponden a: propiedades y posición de vacíos; elipticidad, autovectores y autovalores del tensor de inercia; galaxias miembros de los vacíos; densidad de elementos; y un archivo OFF con los polígonos que forman los vacíos y su distribución en el espacio.

Capítulo 6

Validación

Se evalúa la nueva versión del algoritmo de búsqueda de vacíos cosmológicos sobre distintos datos, en concreto con distribuciones artificiales con vacíos esféricos y cúbicos, y una fracción de los datos observacionales publicados en SDSS Data Release 10.

6.1. Evaluación con datos artificiales

Para observar el comportamiento del algoritmo sobre vacíos con distintas densidades, se generan múltiples conjuntos de datos en 3 dimensiones, produciendo vacíos con puntos en su interior. Se utiliza un parámetro de contraste de densidad $\hat{\delta} = \bar{\rho}/\rho$ para caracterizar la muestra generada.

Primero se evalúa el algoritmo sobre una muestra con vacíos esféricos. Para esto se genera un volumen cúbico de $m \times m \times m$ con $m = 200h^{-1}$Mpc, que en su interior existen 20 vacíos esféricos en posiciones aleatorias, cuyo radio es de $r = 20h^{-1}$Mpc y tienen la restricción de no intersecarse entre sí, ni tener parte del vacío fuera del borde del volumen cúbico. Luego, el volumen fuera de los vacíos se ocupa de puntos aleatorios hasta lograr que tenga una densidad de $\bar{\rho} = 0{,}004$ puntos por h^{-1}Mpc cúbico. En los vacíos esféricos se sitúan puntos aleatorios hasta lograr que la densidad en conjunto de los vacíos sea ρ.

Con esta distribución los vacíos no tienen necesariamente la misma cantidad de puntos en su interior, y la posición de estos puntos tampoco sigue una distribución definida. Sin embargo, se ha observado que en los vacíos la densidad de galaxias tiende a aumentar a medida que uno se aleja de su centro [24]. Por este motivo se incluyen vacíos esféricos con las mismas características anteriores, pero la distancia d de los puntos interiores al centro del vacío sigue una distribución normal con parámetros $\mu = r$, y $\sigma = 3{,}0$, y con la restricción $0 \leq d < r$.

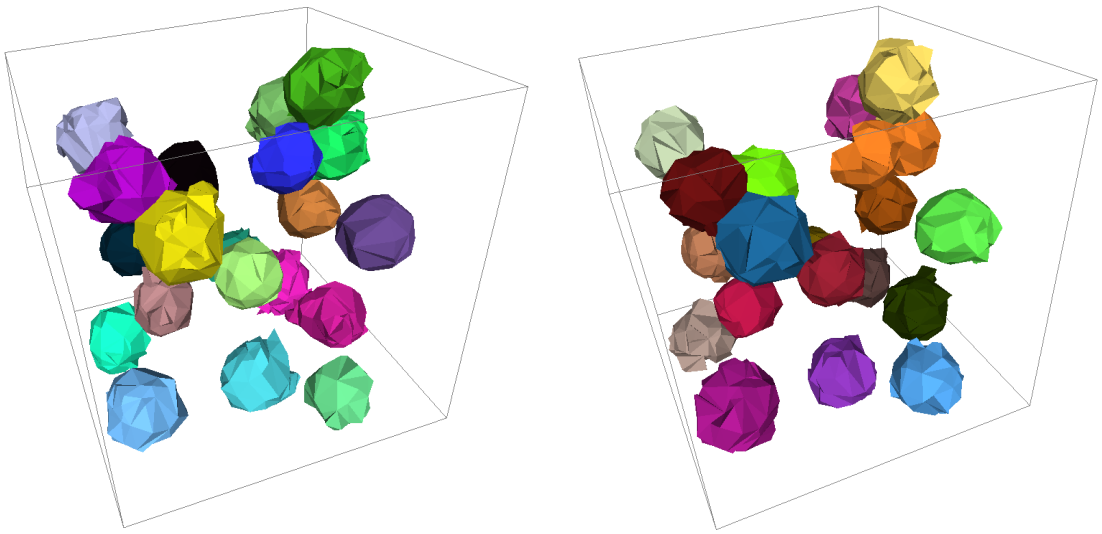
Debido a que los vacíos no son necesariamente esféricos, también se evalúa el algoritmo sobre vacíos cúbicos. Al igual que para las muestras descritas anteriormente, se genera un volumen cúbico de $m \times m \times m$ con $m = 200h^{-1}$Mpc, con 20 vacíos en su interior en posiciones aleatorias, cuya longitud de arista es $s = 30h^{-1}$Mpc. En esta muestra se vuelve a incluir la restricción de que no existan intersecciones entre vacíos, ni que tengan parte de ellos fuera del

borde del volumen cúbico. Para poblar la muestra se utilizan puntos aleatorios fuera de los vacíos hasta lograr una densidad $\bar{\rho} = 0{,}004$, y los puntos en el interior también se encuentran en posiciones aleatorias con una densidad ρ.

Para cada tipo de muestra se trabaja con diferentes valores de ρ, con la intención de evaluar el algoritmo sobre distintos valores de $\hat{\delta}$. La evaluación se hace calculando la tasa de recuperación r_v y la tasa de error e_v. La tasa recuperación de los vacíos se define como $r_v = \frac{V_{\cap v}}{V_v}$, donde $V_{\cap v}$ es el volumen en intersección de los vacíos generados en la muestra con los vacíos detectados, y V_v es el volumen de los vacíos generados. De manera similar, se define la tasa de sobre-detección o de error como $\mathrm{e}_v = 1 - \frac{V_{\cap v}}{V_{v*}}$, donde V_{v*} es el volumen de los vacíos detectados. Al utilizar estos índices se mide el efecto del contraste de densidad sobre la efectividad del algoritmo.

Resultados

En los resultados se observan poliedros de forma redondeada, no necesariamente convexos (ver figura 6.1). No se presentan vacíos fragmentados en aquellos obtenidos por la nueva versión del algoritmo, pero dependiendo del valor utilizado como umbral de densidad, en particular si este es muy alto, pueden ocurrir uniones entre vacíos cercanos (ver figura 6.1b). Los resultados obtenidos sobre los vacíos esféricos, esféricos con distribución normal y cúbicos se muestran en las figuras 6.2, 6.4 y 6.3 respectivamente.



(a) Algoritmo por densidad de puntos (b) Algoritmo por densidad de aristas

Figura 6.1: Vacíos encontrados por DELFIN++ en los datos de vacíos esféricos con contraste de densidad $\hat{\delta} = 80$, y con puntos interiores distribuidos aleatoriamente.

Con todos los métodos de búsqueda de vacíos se obtienen mejores resultados cuando el contraste de densidad $\hat{\delta}$ aumenta. Al trabajar con vacíos esféricos (figura 6.2) la tasa de recuperación con alto contraste de densidad se encuentra sobre el 80 %. Sobre estos mismos contrastes, la tasa de error de DELFIN++ es menor al 20 %, mientras que con la versión previa de DELFIN el error se encuentra sobre el 40 %, lo que indica una mejora en el reconocimiento de los límites o bordes del vacío. En la figura 6.5 se muestra la intersección entre las esferas generadas y los vacíos reconocidos por el algoritmo de densidad de puntos, aquí

se observa que el resultado obtenido no son exactamente las esferas, pero se reconocen las coordenadas y se respeta el tamaño de los vacíos, generando una fiel representación de los datos originales.

Al evaluar los vacíos cúbicos (figura 6.3), la tasa de recuperación con un $\hat{\delta}$ alto es menor que en los vacíos esféricos, con valores entre 65 % y 80 %. Al disminuir el contraste de densidad la tendencia es la misma, llegado a las peores tasas de recuperación con valores menores a 30 %, y tasas de error sobre el 50 %. En estos casos es natural que al disminuir $\hat{\delta}$ los algoritmos tengan mayor dificultad para reconocer los vacíos, debido a que el interior de los vacíos está poblado de manera similar al exterior.

Respecto a los vacíos esféricos con puntos interiores con una distribución normal (figura 6.4), se observan mejores resultados al recuperar los vacíos en comparación con los otros tipos de vacíos generados artificialmente. La mejora en los resultados obtenidos por DELFIN se debe a que como el algoritmo construye vacíos desde regiones menos densas hacia regiones estrictamente más densas, trabaja mejor en vacíos en que el centro se alcanza la menor densidad. A pesar de tener esta mejora, al llegar a $\hat{\delta} = 1{,}5$, el algoritmo de DELFIN presenta una baja en la tasa de recuperación y una alza en la tasa de error, llegando a 15 % y 60 % para cada una de estas. Los vacíos encontrados con esta distribución de puntos tienden a ser más pequeños a medida que disminuye el contraste de densidad, debido a que se mantiene el centro con una baja densidad y la mayoría de los puntos se encuentran concentrados en los bordes de las esferas, generando una pared de alta densidad.

Para todos las muestras, los mejores resultados se obtienen al utilizar DELFIN++ con métricas de densidad de elementos respecto al volumen de sus tetraedros. Con todas las versiones del nuevo algoritmo se mejoran los resultados obtenidos al trabajar con vacíos que en su interior tienen puntos en posiciones aleatorias: para DELFIN la baja en las tasas de recuperación ocurre alrededor de un $\hat{\delta} = 10$, llegando a valores menores al 60 %; para la nueva versión la misma baja ocurre con $2 < \hat{\delta} < 4$. Respecto a los vacíos con distribución normal de puntos, las curvas del nuevo algoritmo se encuentran constantemente sobre 60 % en recuperación y bajo 20 % en error, y los valores son mayormente mejores que aquellos de la versión anterior. En particular, con la nueva versión nunca ocurre la caída en recuperación (y alza en error) al disminuir el contraste de densidad.

En la figura 6.6 se muestra el campo de densidad computado con la métrica de densidad de puntos según el volumen de sus tetraedros. Como es esperado, aquellas zonas con menor densidad (más oscuras) coinciden con las esferas que representan a los vacíos. La diferencia de densidad generada entre estas zonas es la que permite localizar los vacíos con la precisión reportada. Los puntos más cercanos a las caras del volumen cúbico contenedor también son zonas de baja densidad, pero si se llegasen a reconocer como vacíos o parte de un vacío, se descartan por la mala calidad de los tetraedros en los bordes (alargados y volumen relativamente bajo).

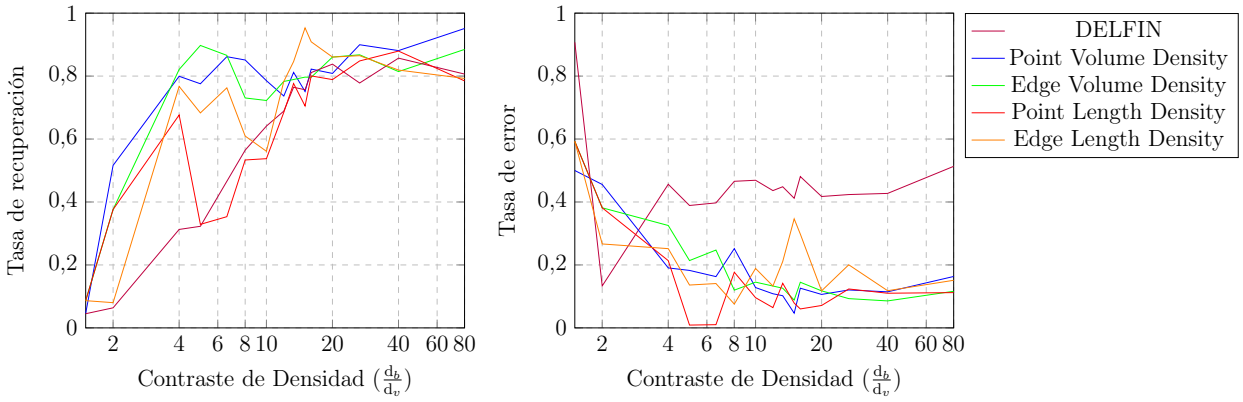


Figura 6.2: Tasa de recuperación de volumen y tasa de error de volumen en la identificación de vacíos esféricos con distintos contrastes de densidad.

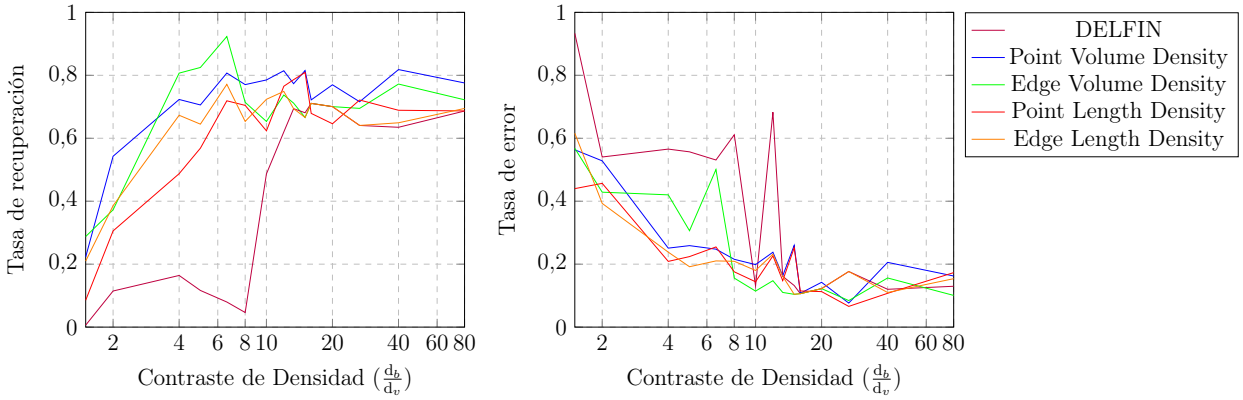


Figura 6.3: Tasa de recuperación de volumen y tasa de error de volumen en la identificación de vacíos cúbicos con distintos contrastes de densidad.

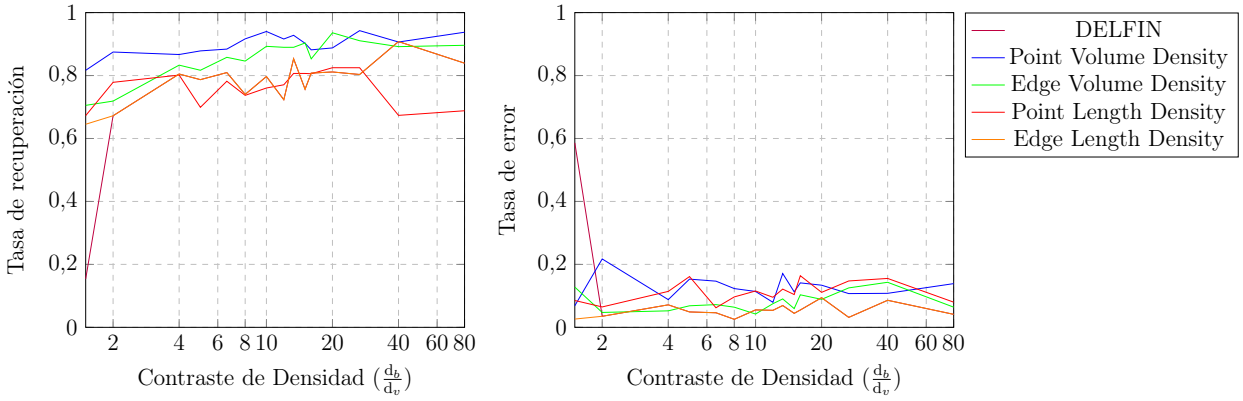


Figura 6.4: Tasa de recuperación de volumen y tasa de error de volumen en la identificación de vacíos esféricos con distintos contrastes de densidad. La distribución de los puntos al interior de las esferas es una distribución normal respecto al centro.

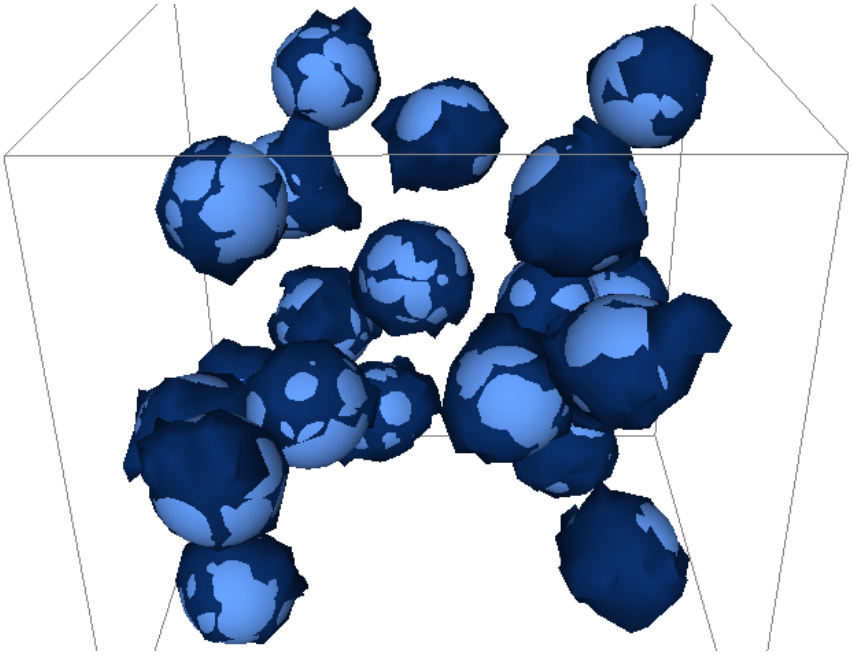


Figura 6.5: Intersección de los vacíos esféricos generados (azul claro) con los vacíos encontrados por DELFIN++ (azul oscuro).

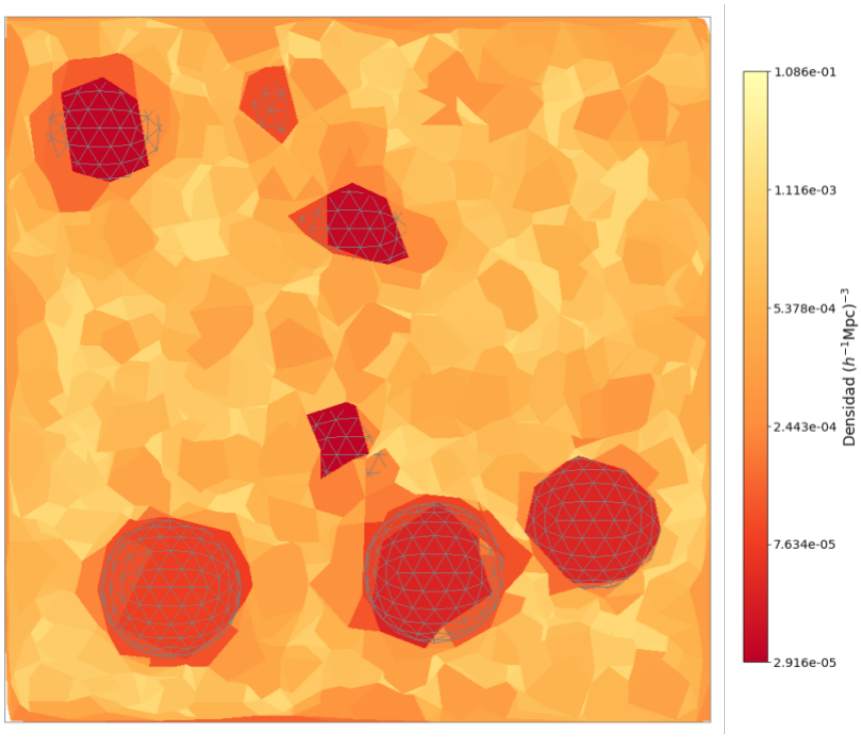


Figura 6.6: Corte en el plano $y = -55h^{-1}$Mpc del campo de densidad calculado con la métrica de densidad de puntos sobre el volumen cúbico con vacíos esféricos con contraste de densidad $\hat{\delta} = 80$. Se muestra la estructura de arcos de las esferas generadas.

6.1.1. Vacíos en el borde de los datos

Los datos de vacíos artificiales son utilizados para evaluar los distintos filtros o post-procesos de vacíos propuestos. Sobre el mismo conjunto de datos se mide el porcentaje de recuperación y de error de vacíos en 7 pruebas con distintos métodos:

A: Sin post-proceso.

B: Descarta vacíos con volumen $V < V_{min}$.

C: Descarta vacíos con volumen $V < V_{min}$.
Descarta vacíos con una arista en el borde de los datos.

D: Descarta vacíos con volumen $V < V_{min}$.
Descarta vacíos con 90 % de su superficie cubierta por tetraedros de mala calidad.

E: Descarta vacíos con volumen $V < V_{min}$.
Descarta vacíos con una arista en el borde de los datos.
Descarta vacíos con 90 % de su superficie cubierta por tetraedros de mala calidad.

F: Descarta vacíos con volumen $V < V_{min}$.
Descarta vacíos con 90 % de su superficie cubierta por tetraedros de mala calidad.
Post-proceso de borrar tetraedros de mala calidad de los bordes de vacíos con más de 55 % de superficie de mala calidad.

G: Descarta vacíos con volumen $V < V_{min}$.
Descarta vacíos con una arista en el borde de los datos.
Descarta vacíos con 90 % de su superficie cubierta por tetraedros de mala calidad.
Post-proceso de borrar tetraedros de mala calidad de los bordes de vacíos con más de 55 % de superficie de mala calidad.

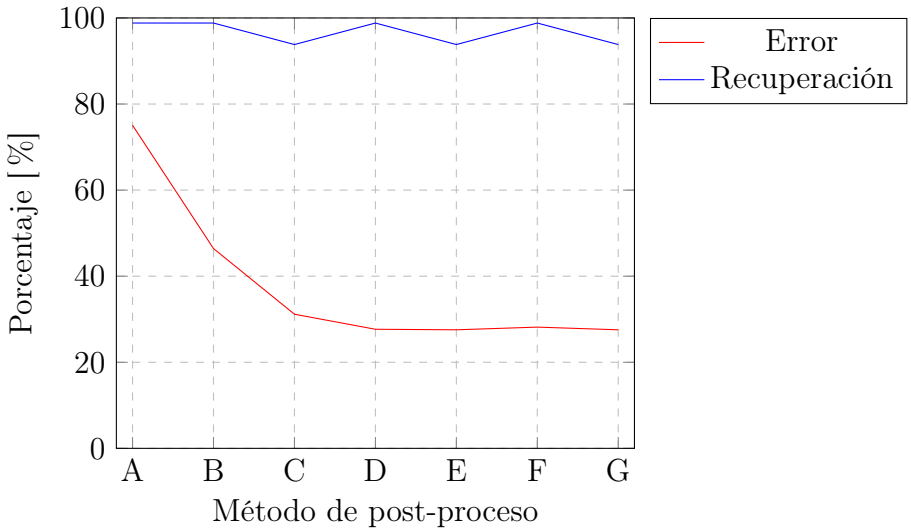


Figura 6.7: Porcentaje de error y recuperación de vacíos en datos artificiales con distintos métodos de post procesos, utilizando DELFIN++ en su versión de identificador de vacíos por densidad de volumen de puntos.

En la figura 6.7 se muestran los resultados obtenidos con los distintos métodos. Si bien todos los métodos tienen buenos resultados en cuanto al porcentaje de recuperación del volumen de los vacíos, aquellos en los que se descartan los vacíos con aristas en el borde de los datos (C, E y G) son los que presentan una pequeña baja en el porcentaje de recuperación. Guiándonos por el porcentaje de error, los mejores resultados ocurren al incluir el proceso de eliminar los vacíos con 90 % de su superficie cubierta por tetraedros de mala calidad, independiente de los otros métodos utilizados.

Agregar el post-proceso de borrar los tetraedros de mala calidad en la superficie de los vacíos no hace una mejora significativa en el resultado al evaluarlo mediante la recuperación y error en el volumen. Considerando estos resultados, que el agregar este último post-proceso significa un aumento en el tiempo de ejecución, que hay un aumento de eficiencia al incluir el proceso de descartar los vacíos con aristas en el borde de los datos, y que mediante los argumentos ofrecidos por el programa se puede elegir qué post-proceso incluir o descartar, se mantiene el método E como el filtro por defecto.

6.2. Evaluación con datos de catálogo

Como se presentó previamente, existe una variedad de algoritmos para ayudar a la búsqueda de vacíos cosmológicos, los que además utilizan distintos tipos de datos de entrada para realizar esta tarea. Debido a que los trabajos previos se evalúan en vacíos encontrados por algoritmos previos, este trabajo también pretende comparar sus resultados por aquellos obtenidos por un buscador de vacíos con resultados publicados. En el trabajo de Alonso[2] la comparación se hace con los resultados de Foster y Nelson[12], esto porque su búsqueda de vacíos a partir de esferas máximas comparte propiedades con las teselaciones de Delaunay, además de utilizar el método del tercer vecino más cercano para realizar uniones entre potenciales vacíos.

Debido a que el algoritmo propuesto en este trabajo de título se separa de la búsqueda de poliedros máximos a partir de tetraedros que comparten su arista más larga, se decide hacer la comparación con VIDE, un algoritmo que utiliza la posición de galaxias como datos de entrada y trabaja con campos de densidades a partir de diagramas de Voronoi. Se presta atención a este algoritmo debido a que es comentado y comparado con DELFIN en el trabajo de J. Ahumada[1], además de ser Open Source.

Se ejecuta el nuevo algoritmo con las distintas métricas de densidad sobre un par de muestras de galaxias limitadas por volumen del Data Release 10 de SDSS. Estos datos son obtenidos a través del portal de catálogos públicos "Public Cosmic Void Catalog"[1], en donde también se publica el algoritmo identificador de vacíos VIDE.

El catálogo de SDSS que se trabaja es Baryon Oscillation Spectroscopic Survey (BOSS), el cual mapea la distribución espacial de Galaxias Rojas Luminosas (LRGs, por sus siglas en inglés) y cuásares para detectar la escala característica impresa por las oscilaciones acústicas bariónicas en el Universo temprano. BOSS tiene dos muestras principales de galaxias, CMASS y LOWZ, además de las muestras más pequeñas, CMASS Sparse y CMASS Commissioning.

[1] `http://www.cosmicvoids.net/`

Los datos obtenidos desde el portal de catálogos son poblaciones de galaxias limitadas por volumen haciendo cortes en corrimiento de rojo y en magnitud de los catálogos LOWZ y CMASS. Se divide el catálogo LOWZ en 4 muestras comenzando en $z = 0{,}1$, y el catálogo CMASS en 2 muestras comenzando en $z = 0{,}45$. La muestra con el *redshift* más bajo en LOWZ la ignoran debido a que se superpone con datos de otro Data Release. Los detalles de corrimiento de rojo y cantidad de datos de cada muestra luego de aplicar el filtro por volumen se encuentran en la tabla 6.1.

Nombre de muestra	z_{min}	z_{max}	$\bar{b}^{-1/3}$ (h^{-1} Mpc)	$N_{galaxias}$	N_{vacios}
dr10lowz2	0.2	0.3	13.8	67030	137
dr10lowz3	0.3	0.4	15.1	100727	199
dr10lowz4	0.4	0.45	16.4	34281	91
dr10cmass1	0.45	0.5	14.0	108859	227
dr10cmass2	0.5	0.6	15.2	249506	694

Tabla 6.1: Muestras de galaxias en los datos limitados por volumen de SDSS DR10, sus límites de corrimiento de rojo, la densidad media del número de galaxias y el número total de vacíos identificados por VIDE. Datos obtenidos de *Sutter* (2014)[27].

Para trabajar con los datos deben transformarse de coordenadas esféricas a coordenadas cartesianas. Esto se resuelve con *Astropy*[5, 6], calculando la distancia por comovimiento d_c utilizando el corrimiento de rojo z y un objeto de cosmología FlatLambdaCDM con los argumentos $H_0 = 71$(km/s)/Mpc y $\Omega_0 = 0{,}3$. Luego con la ascensión recta ra (longitud), declinación dec (latitud) y distancia por comovimiento d_c de los puntos, se transforma su representación a coordenadas cartesianas.

Debido a que la forma de las muestras limitadas por volumen es un corte radial del espacio, su envolvente convexa contiene tetraedros fuera del dominio. Para evitar estos tetraedros, en vez de trabajar con la teselación de Delaunay producida por *qhull*, se trabaja con una teselación de Delaunay restringida, en donde la superficie que delimita el dominio de la tetraedrización se obtiene calculando el α-complex del conjunto de puntos, con $\alpha\,\% = 3{,}0$. El α-complex se computa utilizando *MeshLab*[9], y la teselación restringida se calcula utilizando *TetGen*[26], cuyo resultado se modifica para que se ajuste al formato de *qhull*.

Evaluar los resultados no es una tarea trivial. Los vacíos de DELFIN++ son poliedros generados de la unión de tetraedros, mientras que los vacíos generados por VIDE son agregaciones de celdas de Voronoi. Si bien en los resultados publicados por VIDE se tiene el centroide de los vacíos y el radio de la esfera de volumen equivalente, comparar la intersección de los resultados de DELFIN++ con estas esferas requeriría obtener la superficie de los vacíos a comparar, calcular la curva de intersección e integrar el volumen. Este método se escapa del alcance de este trabajo, e implica perder información sobre la forma y las galaxias pertenecientes a los vacíos.

Considerando esto, se decide utilizar la intersección de galaxias pertenecientes a los vacíos obtenidos por los distintos métodos, además de comparar el volumen de las regiones obtenidas. Nuevamente se define una tasa de recuperación r_g y una tasa de error e_g. La tasa de recuperación de galaxias se define como $r_g = \frac{N_\cap}{N_{gV}}$, donde $N_\cap$ es la cantidad de galaxias que

pertenecen a los vacíos de VIDE y a los detectados por DELFIN++, y N_{gV} es el número de galaxias que pertenecen a vacíos de VIDE. Similar a lo expuesto anteriormente, la tasa de error se define como $e_g = 1 - \frac{N_\cap}{N_{g*}}$, donde N_{g*} es la cantidad de galaxias que forman parte de los vacíos detectados.

Resultados

La cantidad de vacíos encontrados al utilizar distintas métricas de densidad en las muestras dr10cmass1 y dr10cmass2 se muestran en la tabla 6.2.

Muestra	PointVolume N_{vacios}	PointLength N_{vacios}	EdgeVolume N_{vacios}	EdgeLength N_{vacios}	VIDE N_{vacios}
dr10cmass1	163	164	412	412	227
dr10cmass2	445	675	1036	904	694

Tabla 6.2: Cantidad de vacíos encontrados por las distintas métricas de densidad que se utilizan en DELFIN++, y la cantidad de vacíos encontrados por VIDE.

Como resultado general, se tiene que al trabajar con la versión de DELFIN++ que utiliza densidad de aristas se generan más vacíos que los obtenidos con VIDE, y también más vacíos que al trabajar con la densidad de puntos. El volumen de tales vacíos tiende a ser mucho menor que aquellos encontrados por VIDE: con EdgeLengthDensity los vacíos encontrados (en ambas muestras) tienen volumen menor a $1 \times 10^5 (\mathrm{Mpc}/h)^3$, y con EdgeVolumeDensity son más pequeños que $1{,}5 \times 10^5 (\mathrm{Mpc}/h)^3$ (ver figuras 6.8 y 6.9). Si bien con este método se generan vacíos esféricos, la gran cantidad de vacíos de bajo volumen genera espacios como el presentado en las figuras 6.12 y 6.13, en donde hay una alta presencia de pequeños poliedros con formas aplanadas, los cuales se asemejan a astillas.

Para los métodos de densidad de puntos, la cantidad de vacíos encontrados es menor que la cantidad reportada por VIDE. Estos vacíos tienden a ser más grandes que aquellos generados con los métodos de aristas, y la variación en el volumen también es mayor que con EdgeVolumeDensity y EdgeLengthDensity. En ambas muestras ocurre que al trabajar con PointVolumeDensity se llegan a generar vacíos de mayor tamaño que con PointLengthDensity, sin embargo, la mediana del volumen se mantiene cercana para ambos métodos: $9{,}3 \times 10^4$ y $8{,}4 \times 10^4 (\mathrm{Mpc}/h)^3$ en dr10cmass1, y $1{,}0 \times 10^5$ y $9{,}1 \times 10^4 (\mathrm{Mpc}/h)^3$ en dr10cmass2.

Los vacíos obtenidos con métodos de densidad de puntos tienen formas esféricas (ver figuras 6.10 y 6.11), lo cual es esperable debido a que los poliedros se construyen a partir de tetraedros que rodean un punto, evitando las formas más aplanadas. Estos cuerpos tienen semejanza a los poliedros descritos por VIDE (ver sección 2.2.2), los cuales adquieren una forma esférica. En la figura 6.14 se destacan las galaxias que forman parte de vacíos en la muestra dr10cmass2, dando una idea de la forma de los vacíos y su distribución en el espacio.

Los resultados de recuperación de galaxias en vacíos se encuentran en las tablas 6.3 y 6.4. En todos los casos se presentan porcentajes de recuperación menores a 14 %. El alto número de vacíos pequeños deriva en malos resultados para los métodos de aristas, pues en estos es donde se obtiene la peor tasa de recuperación y de error. En particular al utilizar

EdgeLengthDensity se obtiene la menor recuperación de cada muestra, con $r_g = 2{,}97\,\%$ en dr10cmass1 y $r_g = 2{,}04\,\%$ en dr10cmass2.

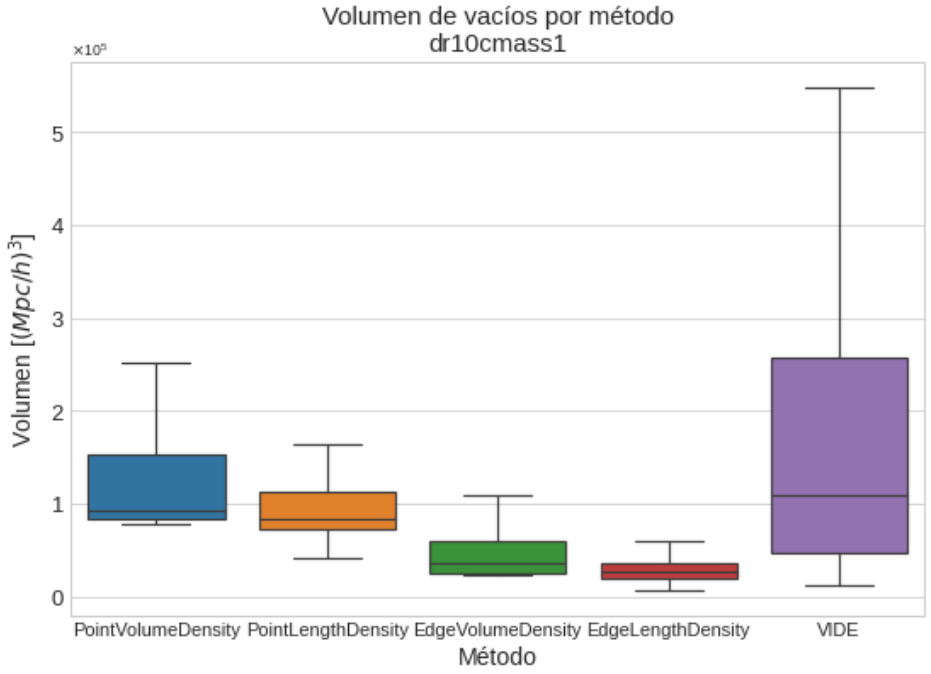


Figura 6.8: Volumen de los vacíos encontrados por DELFIN++ sobre la muestra dr10cmass1, junto con el volumen de los vacíos reportados por VIDE como referencia.

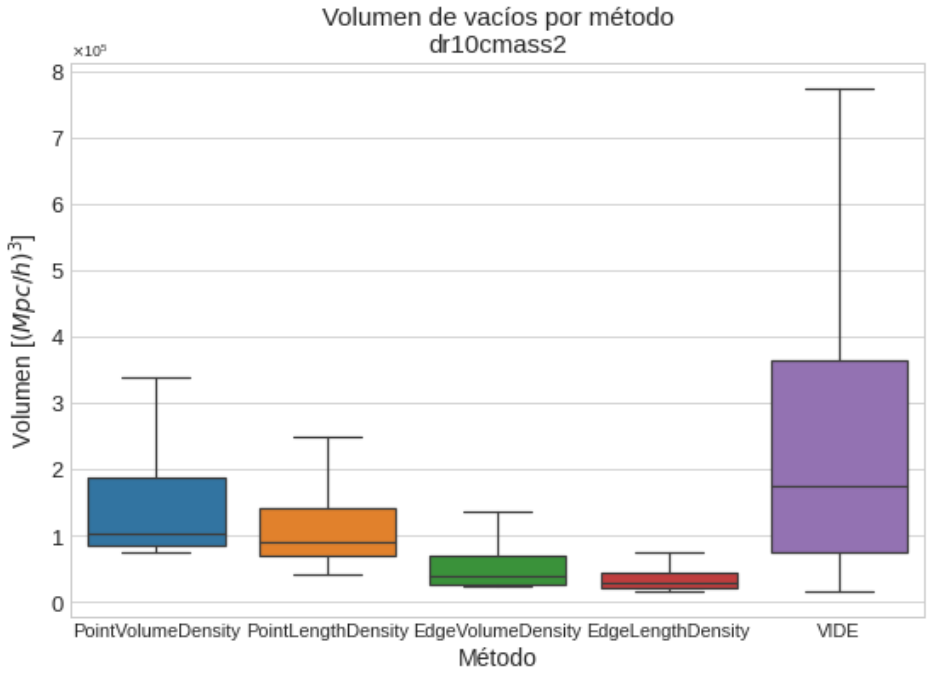


Figura 6.9: Volumen de los vacíos encontrados por DELFIN++ sobre la muestra dr10cmass2, junto con el volumen de los vacíos reportados por VIDE como referencia.

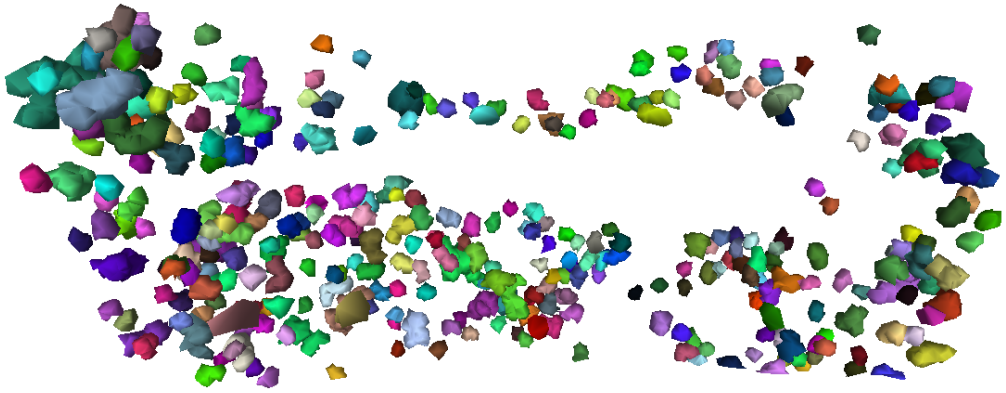


Figura 6.10: Visualización de los vacíos encontrados en la muestra CMASS $0{,}5 \leq z \leq 0{,}6$ en coordenadas con $x < 0$ utilizando el algoritmo DELFIN++ con el método de densidad de puntos por volumen de tetraedros.

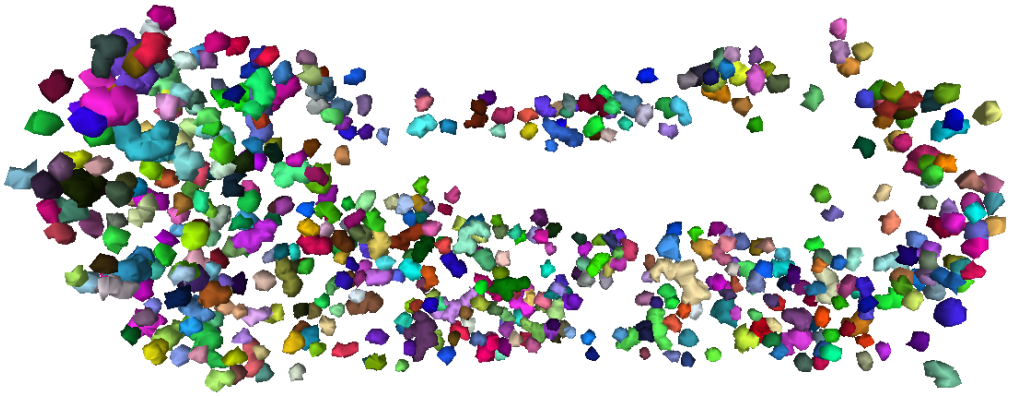


Figura 6.11: Visualización de los vacíos encontrados en la muestra CMASS $0{,}5 \leq z \leq 0{,}6$ en coordenadas con $x < 0$ utilizando el algoritmo DELFIN++ con el método de densidad de puntos por longitud de aristas.

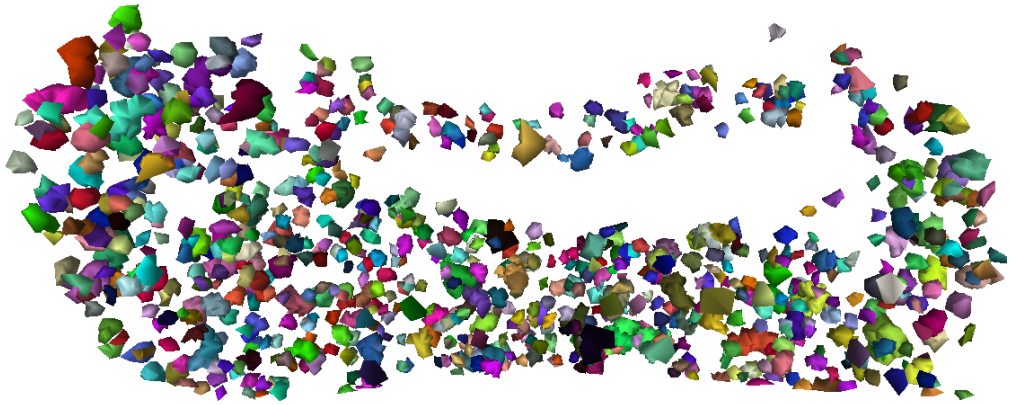


Figura 6.12: Visualización de los vacíos encontrados en la muestra CMASS $0{,}5 \leq z \leq 0{,}6$ en coordenadas con $x < 0$ utilizando el algoritmo DELFIN++ con el método de densidad de aristas por volumen de tetraedros.

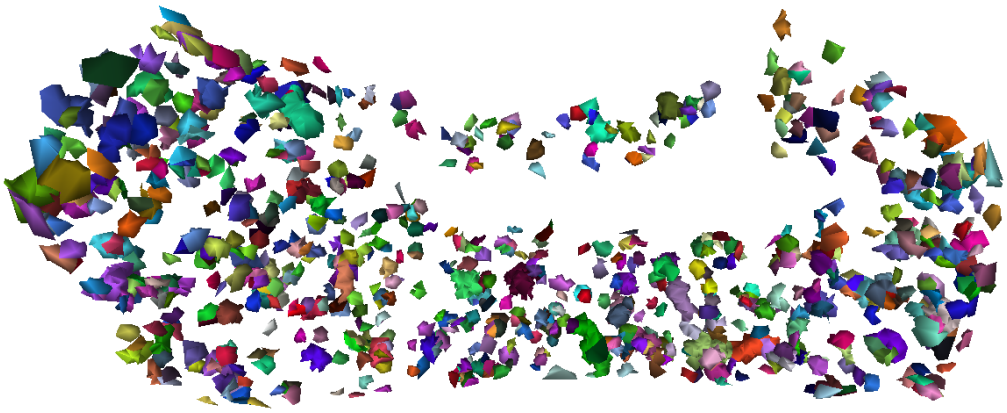


Figura 6.13: Visualización de los vacíos encontrados en la muestra CMASS $0{,}5 \leq z \leq 0{,}6$ en coordenadas con $x < 0$ utilizando el algoritmo DELFIN++ con el método de densidad de aristas por su largo y el volumen de sus tetraedros.

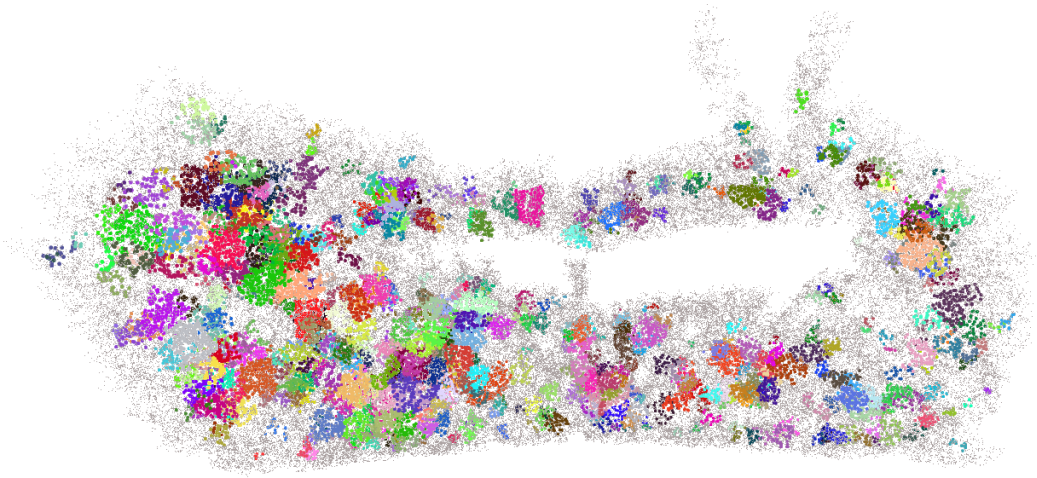


Figura 6.14: Visualización de las galaxias en la muestra CMASS $0{,}5 \leq z \leq 0{,}6$ en coordenadas con $x < 0$. En colores -diferentes a gris- se presentan las galaxias que son parte de un vacío reportado por VIDE, en gris las galaxias que no pertenecen a un vacío.

	N_{g*}	$N_{\cap}$	r_g	e_g
PointVolumeDensity	6046	362	4,51 %	94,01 %
PointLengthDensity	5956	549	6,84 %	90,78 %
EdgeVolumeDensity	5884	317	3,95 %	94,61 %
EdgeLengthDensity	5532	238	2,97 %	95,70 %

Tabla 6.3: Recuperación de galaxias en los vacíos encontrados por DELFIN++ sobre la muestra dr10cmass1, comparando los resultados con las galaxias de los vacíos reportados por VIDE (8024 galaxias sobre 227 vacíos).

	N_{g*}	$N_{\cap}$	r_g	e_g
PointVolumeDensity	21222	3157	8,11 %	85,12 %
PointLengthDensity	28075	5300	13,62 %	81,12 %
EdgeVolumeDensity	16785	1407	3,62 %	91,62 %
EdgeLengthDensity	13265	793	2,04 %	94,02 %

Tabla 6.4: Recuperación de galaxias en los vacíos encontrados por DELFIN++ sobre la muestra dr10cmass2, comparando los resultados con las galaxias de los vacíos reportados por VIDE (38920 galaxias sobre 694 vacíos).

Los métodos de densidad de puntos tienen relativamente mejores resultados, destacando la recuperación de galaxias con PointLengthDensity sobre la muestra dr10cmass2, donde $r_g = 13{,}62\,\%$ y $e_g = 81{,}12\,\%$. En este caso, el 18,9 % de las galaxias que se reconocieron como miembro de un vacío, también eran parte de un vacío reportado por VIDE.

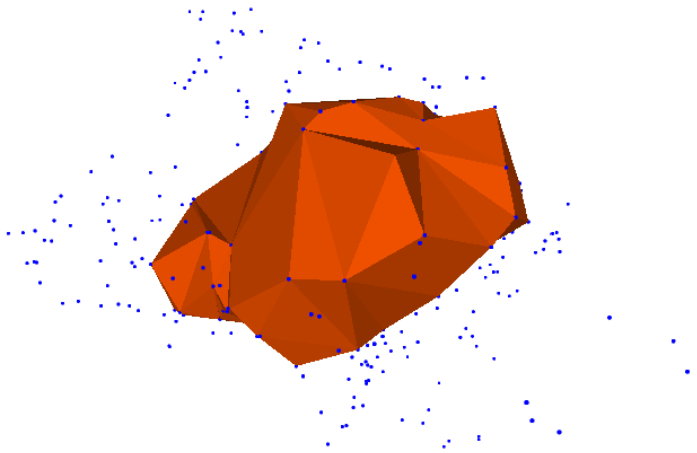


Figura 6.15: Vacío de DELFIN++ (poliedro naranja) encontrado con el método de densidad de puntos por la longitud de sus aristas, y las 280 galaxias del vacío de VIDE (azul) en su misma posición.

En la figura 6.15 se muestra un vacío con 66 galaxias y volumen 232069 $(\mathrm{Mpc}/h)^3$, encontrado por DELFIN++ con el método de densidad de puntos según la longitud de sus aristas. Las galaxias del vacío de VIDE que se encuentra en la misma posición -al calcular su teselación de Delaunay- ocupan un volumen de 883523 $(\mathrm{Mpc}/h)^3$, lo cual es aproximadamente 3,8 veces el volumen del poliedro expuesto en la figura.

En la figura 6.16 se muestra una coincidencia de vacíos más ajustada. El vacío encontrado por DELFIN++ formado por 36 galaxias, comparte 24 de estas con el vacío de VIDE que se encuentra en la misma posición, el cual a su vez tiene 39 galaxias. En términos de volumen, el vacío de VIDE cubre un espacio de 60659 $(\mathrm{Mpc}/h)^3$, mientras que el de DELFIN++ ocupa 56805 $(\mathrm{Mpc}/h)^3$, lo que es aproximadamente el 93,6 % del volumen del primero.

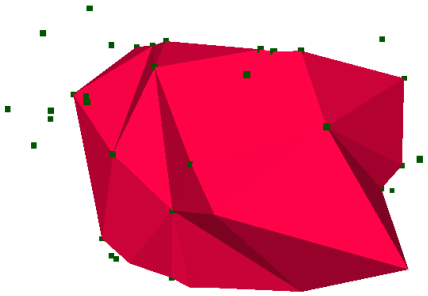


Figura 6.16: Vacío de DELFIN++ (poliedro rosa) encontrado con el método de densidad de puntos por la longitud de sus aristas, y las 39 galaxias del vacío de VIDE (verde) en su misma posición.

6.3. Análisis y Discusión

El algoritmo DELFIN++ se separa de la construcción del poliedro más grande posible a partir de las aristas más largas en una teselación de Delaunay, y en cambio el trabajo se asocia a la densidad de los elementos geométricos de la malla. Si antes era similar al algoritmo de Foster & Nelson, ahora se acerca a las ideas propuestas en los identificadores de vacíos basados en la transformación divisoria, en donde se hacen crecer los poliedros según la densidad de las partículas que definen la zona, y se utilizan límites de densidad para definir las fronteras de los vacíos.

Aunque los resultados respecto a las tasas de recuperación y de error definidas para la evaluación con datos de catálogo entregan valores bajos en la coincidencia de galaxias, realmente no se espera una relación uno a uno de los vacíos, sino que se tiene como objetivo localizar regiones de baja densidad en una muestra observacional.

Comparativamente, los mejores resultados fueron obtenidos al trabajar con densidad de puntos, encontrando vacíos con formas más acordes a lo esperado en una región del universo. Para los datos de catálogo, el rango de volumen de los vacíos encontrados con estas densidades tiende a ser más amplio que el de los métodos de densidad de aristas, y sus valores (sin datos atípicos) se asemejan a los del rango intercuartil del volumen de los vacíos de VIDE. En los datos de vacíos artificiales, se obtienen buenos resultados con todas las métricas de densidad de DELFIN++, destacando el método de densidad de puntos por el volumen de sus tetraedros, el cual entrega la mayor recuperación de volumen en todas las muestras.

A partir de las visualizaciones de los vacíos encontrados en los datos de SDSS DR10, y de las coincidencias presentadas en las figuras 6.15 y 6.16, se reconoce la detección de regiones de baja densidad. En particular, en la figura 6.15 se muestra un caso que se repite en el espacio trabajado, en el cual se encuentra un vacío en coordenadas donde VIDE también reporta uno, pero la cantidad de galaxias y el volumen del vacío del algoritmo basado en diagramas de Voronoi es mucho mayor. Esto ocurre porque en DELFIN++ se exige que una galaxia que es parte de un vacío tenga una densidad menor al umbral definido, o que sea vecina de una galaxia con baja densidad. Este comportamiento explica la baja coincidencia de galaxias y la tendencia a reconocer vacíos más pequeños que los de VIDE.

El algoritmo puede seguir siendo beneficiado por mejoras en la implementación o extensiones que apoyen el trabajo. Por ejemplo, respecto a la implementación, se puede paralelizar el proceso posterior a la identificación de los vacíos, en los cuales se descartan aquellos cuyo volumen es muy pequeño o que tienen gran parte de su volumen en el borde de los datos.

El caso de los vacíos con tetraedros cerca de los límites del volumen contenedor de la teselación ha sido un tema de discusión desde la concepción del algoritmo. En 3 dimensiones es común la existencia de tetraedros con bajo volumen relativo al largo de sus aristas, y si bien pueden estar presentes en cualquier región de la teselación, son más comunes en los bordes de esta, como parte o cercanos a la envolvente convexa. Esto genera problemas como tener vacíos cercanos a los bordes que incluyen puntos que son parte de uno de estos tetraedros alargados, por lo que incluyen puntos lejanos al vacío, degenerando su forma. Otro problema es que se pueden generar poliedros formados solo por estos tetraedros, los que no se deben clasificar como vacío debido a que su baja densidad es producto de que no está la información completa del espacio. En este trabajo es de gran importancia las coordenadas de cada partícula o galaxia en el espacio, por lo que no se puede hacer un suavizamiento de la malla basado en cambiar la posición de las partículas para tratar los tetraedros de mala calidad. En cambio se puede incluir un refinamiento de la malla, en el cual se realice la distinción entre los puntos que representan galaxias y puntos auxiliares destinados a mejorar la definición de las regiones fronterizas. Una opción más sencilla para evitar generar vacíos desde el borde es solo considerar la densidad de los puntos que no son parte de tetraedros en la frontera, lo que implicaría que las galaxias cuyo *redshift* está en el límite del volumen se utilicen solo para delimitar el espacio, y no para ser miembros de vacíos (lo que de todas formas ocurre al descartar vacíos con aristas en el borde).

Debido a que las muestras de datos con las que se trabajan son limitadas por volumen, cuya forma es de un corte radial del espacio, hacer la teselación de Delaunay con *qhull* genera muchos tetraedros fuera del dominio, ya que se calcula a partir de la envoltura convexa de los datos. En este trabajo, para evitar los tetraedros externos se trabajó con una teselación de Delaunay restringida. Este proceso exigió realizar muchas modificaciones al flujo de trabajo de preparación de datos, pues con *qhull* no se pueden computar teselaciones restringidas. La dificultad agregada a este proceso puede ser determinante en la decisión de utilizar o no el algoritmo, por lo que se propone extender la modificación para aceptar teselaciones no solo obtenidas desde *qhull*, sino también de librerías como *CGAL* y *TetGen*. Además, se debe evaluar un método para generar la teselación de manera más sencilla, en lo posible automatizando o evitando el paso de generar la superficie que delimita el dominio de la malla.

6.4. Escalabilidad de *software*

Se analiza la implementación de DELFIN++ sobre datos con n puntos en un cubo de $m \times m \times m$ ($m = 200h^{-1}$Mpc), usando un conjunto de datos de vacíos esféricos generados artificialmente. Los experimentos fueron repetidos para valores de $n = 1000 \times 2^0, 1000 \times 2^1, ..., 1000 \times 2^{11}$ usando un contraste de densidad $\hat{\delta} = 10$. En cuanto al tiempo de ejecución, solo se considera aquel del algoritmo, lo que comprende desde la lectura de los datos generados por *qhull* hasta que se obtienen, evalúan y filtran los vacíos.

El sistema utilizado consta de un procesador Intel Core i7-4710HQ de 4 núcleos, 16(2x8)

GB de memoria DDR3 de 1600 MHz y Windows 10 Home versión 2004 como sistema operativo, ejecutando las pruebas en WSL2 con Ubuntu 20.04. Para medir el consumo de memoria se utiliza la herramienta *massif* de *Valgrind*[22]. El análisis se detiene con 2048000 puntos debido a que la memoria requerida para la siguiente cantidad de puntos sobrepasa la memoria disponible.

El tiempo total de ejecución se ajusta a una curva linearítmica $O(n \log n)$, satisfaciendo el análisis de complejidad del algoritmo realizado previamente, en el cual se trabajó con la propiedad de que en la práctica el número de tetraedros en una teselación de Delaunay es lineal en el número de vértices. Se presenta una mejora en el tiempo de ejecución de DELFIN++ respecto a DELFIN. Si bien es una mejora en el tiempo total (figura 6.17), la diferencia se marca en el proceso de búsqueda de vacíos (figura 6.18). La diferencia es de aproximadamente un orden de magnitud, lo que al trabajar con 1000×2^4 puntos son 10 segundos, pero al trabajar con 1000×2^{11} puntos la diferencia aumenta a 38 minutos.

Al dividir los tiempos de ejecución por $\log n$ y ajustarlos a una recta, se observa la diferencia de tiempos entre DELFIN y DELFIN++ (ver figura 6.19). Las pendientes de DELFIN++ son mucho menores a la de DELFIN. Mientras que DELFIN tiene una pendiente $m = 9{,}58\text{e-}05$, las pendientes de DELFIN++ son más cercanas a $m = 1{,}00\text{e-}05$ ($\pm 0{,}18\text{e-}05$). Para valores y gráficos más precisos de las rectas ajustadas, revisar la tabla 6.5 y las figuras en el apéndice A. En los cocientes de las pendientes entre los dos software (m_{DELFIN}/m) se refleja la mejora en el desempeño de DELFIN++ respecto a DELFIN, llegando a ser 9 veces más rápido.

Método	m	b	R^2	m_{DELFIN}/m
DELFIN	9,58e-05	−0,170	99,90 %	1,00
DELFIN++ PointVolumeDensity	9,86e-06	−0,013	99,97 %	9,72
DELFIN++ PointLengthDensity	1,18e-05	0,043	99,99 %	8,11
DELFIN++ EdgeVolumeDensity	1,09e-05	−0,002	99,98 %	8,75
DELFIN++ EdgeLengthDensity	1,01e-05	0,034	99,99 %	9,53

Tabla 6.5: Valores de pendiente m, ordenada en el origen b y coeficiente de correlación R^2 para la regresión lineal sobre los pares $(n, t/\log n)$ de los tiempos de ejecución de la búsqueda de vacíos.

Para disminuir los tiempos de ejecución, el algoritmo puede ser paralelizado. En particular se puede paralelizar la etapa en la que se descartan los vacíos encontrados, ya que las semillas de los mismos ya fueron encontradas y no existiría *data race* sobre los tetraedros, pues con los métodos utilizados actualmente los vacíos no comparten sus tetraedros, ni aumentan en tamaño. De todas formas, antes de implementar la paralelización se debe llevar a cabo un análisis y diseño más completo de la modificación.

El aumento del consumo máximo de memoria (figura 6.20) en DELFIN++ también es lineal respecto a la cantidad de puntos de la muestra, y se mantiene cercano a los valores de memoria utilizada por DELFIN. El mayor consumo de memoria ocurre durante la ejecución de la operación de búsqueda de vacíos, donde mayormente la memoria se encuentra asignada a la identificación de los tetraedros, y la asignación de memoria para la densidad de los elementos es la que genera el aumento de consumo que lleva a los valores máximos reportados. La excepción a esta regla se genera al trabajar con PointLengthDensity, en donde la memoria

asignada para almacenar información asociada a la densidad de puntos es mayor al resto.

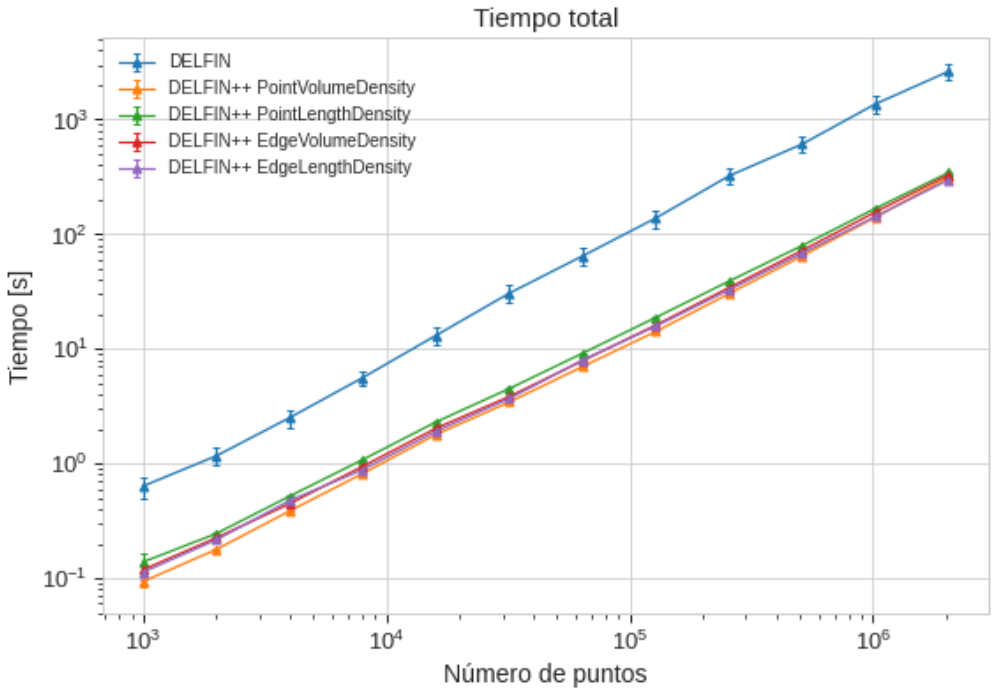


Figura 6.17: Tiempo total de ejecución de DELFIN y DELFIN++ sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000, 2048000$ puntos.

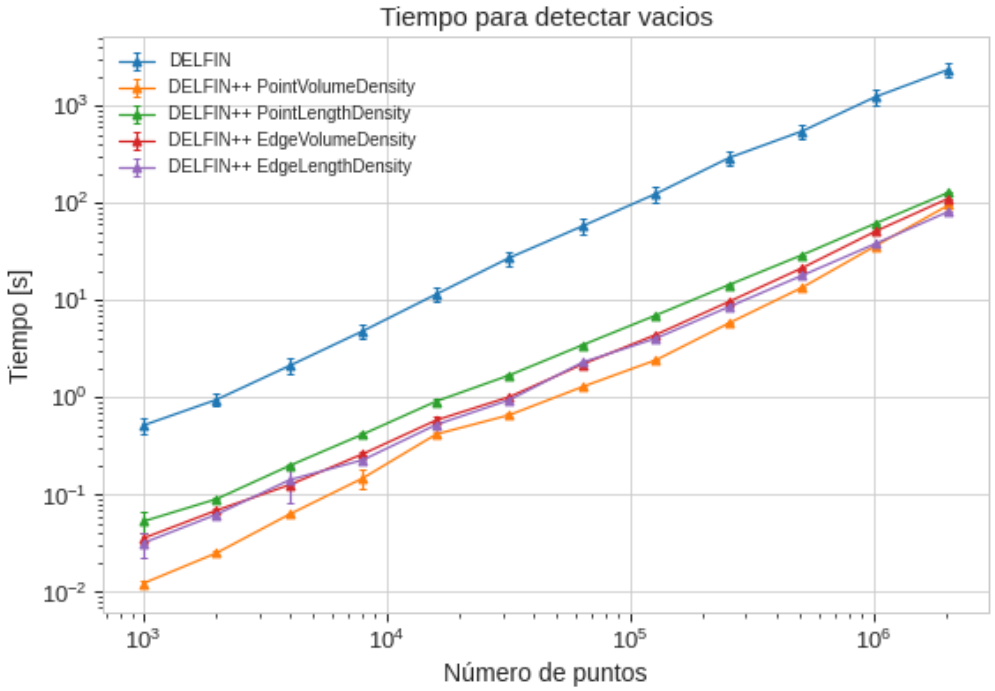


Figura 6.18: Tiempo de búsqueda de vacíos de DELFIN y DELFIN++ sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000, 2048000$ puntos.

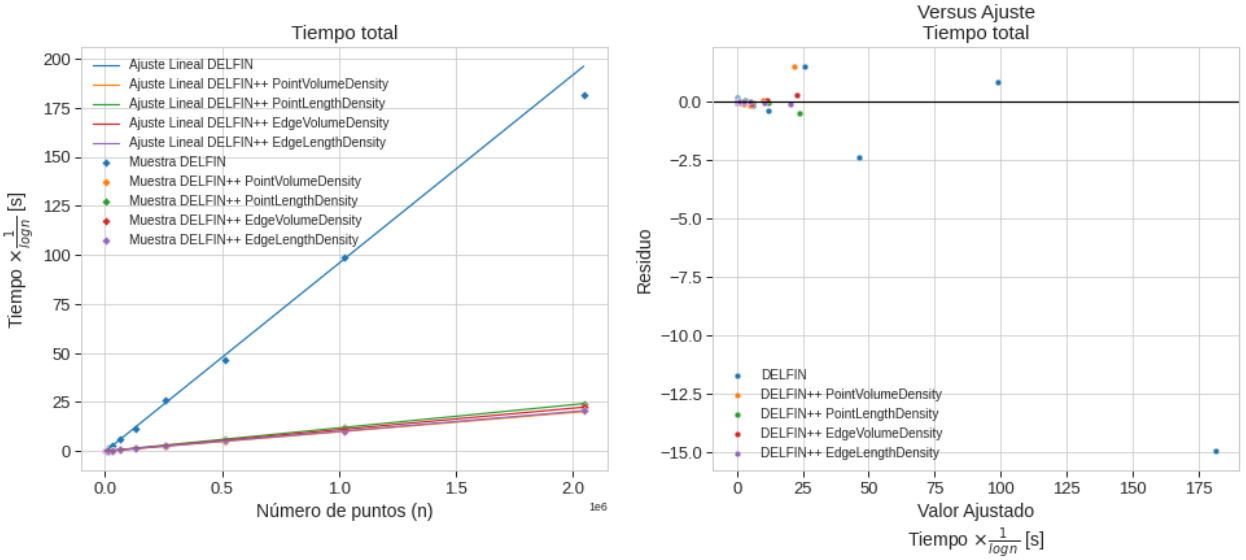


Figura 6.19: En la izquierda, el ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN y DELFIN++. En la derecha, el residuo entre los valores observados (puntos) y los valores ajustados (recta en 0).

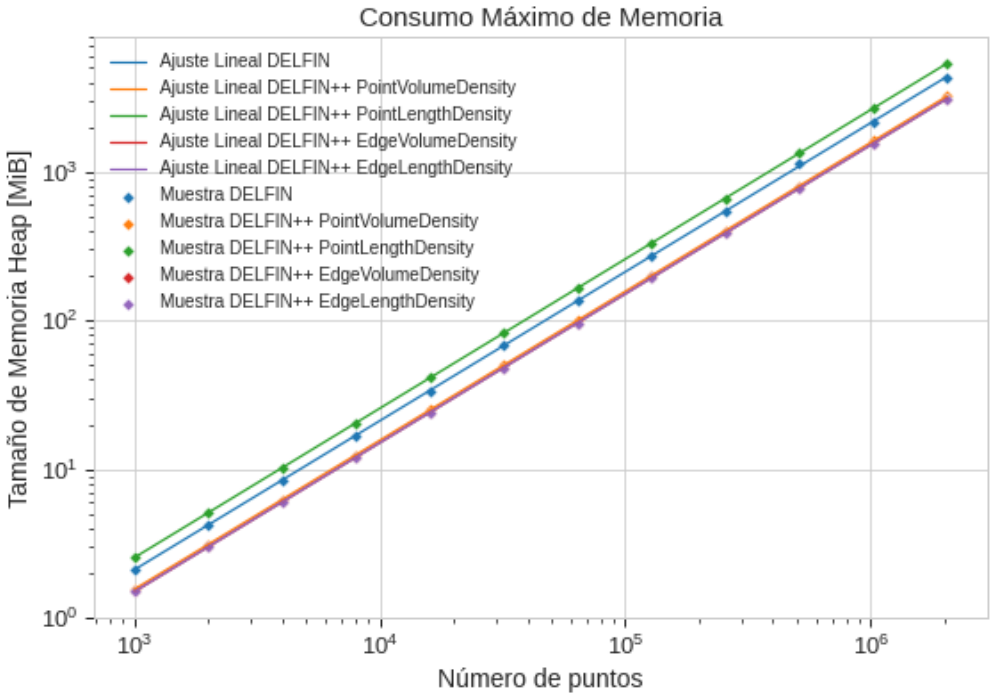


Figura 6.20: Consumo máximo de memoria dinámica al ejecutar DELFIN y DELFIN++ sobre un conjunto de datos con $n = 1000, 2000, ..., 1024000, 2048000$ puntos.

6.5. Calidad de *software*

Sobre el software extendido se realiza un análisis estático de código con *SourceMonitor* y *Sourcetrail*, haciendo un enfoque en las métricas de acoplamiento, cohesión y complejidad de código. Además se mide la cobertura de código por las pruebas de software utilizando *gcov*, y de esta manera determinar qué partes no han sido comprobadas. Finalmente, se analiza en tiempo de ejecución la posible existencia de fugas de memorias.

6.5.1. Acoplamiento y cohesión

Los valores de las métricas resultantes del análisis estático se presentan en la tabla 6.6.

	Inheritance				**Cohesion**		**Coupling**		
Name	**NOC**	**DIT**	**NM**	**NF**	**LCOM**	**ABC**	**Ca**	**Ce**	**In**
Dictionary	3	0	7	1	0,86	7	3	0	0,00
PointDictionary	0	1	6	3	0,28	29	10	1	0,09
Edge	0	0	15	6	0,81	11	8	1	0,11
EdgeDictionary	0	1	7	6	0,67	28	8	3	0,27
Tetrahedron	0	0	21	11	0,84	24	14	3	0,18
TetrahedronDictionary	0	1	11	7	0,60	48	16	4	0,20
ElementDensity	6	0	6	1	0,83	3	4	1	0,20
EdgeDensity	2	1	8	2	0,75	34	3	4	0,57
EdgeLengthDensity	0	2	1	2	0,00	19	1	5	0,83
EdgeVolumeDensity	0	2	1	2	0,00	18	1	5	0,83
PointDensity	2	1	9	3	0,78	34	3	4	0,57
PointLengthDensity	0	2	1	3	0,00	28	1	5	0,83
PointVolumeDensity	0	2	1	3	0,00	16	1	4	0,80
VoidBuilder	4	0	4	2	0,38	46	4	5	0,56
DelfinBuilder	0	1	6	5	0,50	38	0	5	1,00
DensityBuilder	2	1	14	6	0,76	39	2	4	0,67
EdgeDensityBuilder	0	2	15	6	0,73	34	0	10	1,00
PointDensityBuilder	0	2	15	6	0,73	34	0	8	1,00
VoidJoiner	1	0	7	2	0,64	39	1	4	0,80
EdgeJoiner	0	1	10	6	0,75	24	0	5	1,00
VoidSpace	0	0	19	9	0,69	40	6	3	0,33
VoidShape	0	0	8	5	0,55	18	1	1	0,50

Tabla 6.6: Métricas de acoplamiento, cohesión y herencia de la implementación de DELFIN++.

Las clases concretas de la implementación de métricas de densidad presentan los mejores resultados de cohesión (LCOM=0.0). Recordemos que se recomienda evitar las clases problemáticas en las cuales LCOM >0.8, número de variables (NF) >10 y número de métodos (NM) >10. Dentro de esta categoría solo se tiene a la clase Tetrahedron, en la cual se aumenta la cantidad de variables para determinar la calidad de los tetraedros según su Aspect Ratio Gamma. Esta clase es importante para el sistema: su responsabilidad es de caracterizar a los

tetraedros de la teselación, por lo que la extensión realizada sigue estando dentro del alcance de sus obligaciones.

Respecto al acoplamiento por herencia, dada por las métricas de número de hijos (NOC) y profundidad del árbol de herencia (DIT), se mantienen los buenos resultados. Las clases introducidas generan un aumento en el número de hijos de la clase VoidBuilder, subiendo de 1 a 4 hijos, y aquellas en las que se implementa las métricas de densidad de la teselación presentan el mayor número de hijos (ElementDensity con NOC=6), que puede seguir aumentando si se definen nuevas métricas. Sin embargo, la dificultad para el mantenimiento de código comienza cuando un alto NOC se combina con un DIT mayor a 6, lo que no es el caso, ya que el máximo DIT del proyecto es igual a 2. Por lo tanto, estos resultados indican que existe poco acoplamiento por métodos y variables de instancia heredadas.

Se repiten los resultados que indican que las clases que se encuentran al final del flujo de búsqueda de vacíos (EdgeDensityBuilder, PointDensityBuilder DelfinBuilder y EdgeJoiner) presentan alta inestabilidad (In=1.0) por su nulo acoplamiento aferente (Ca=0), lo cual es un comportamiento normal y esperable. No existen otras clases inestables además de las ya mencionadas.

Previamente se tenía que las clases con mejor estabilidad eran Dictionary, PointDictionary y Edge. Estos resultados se mantienen, pero además se incluye a Tetrahedron, TetrahedronDictionary, ElementDensity y EdgeDictionary dentro de las clases estables. Es importante que las clases con alta responsabilidad en el sistema, que trabajan con múltiples otras clases para obtener los resultados y por lo tanto tienen alto acoplamiento total tengan un bajo índice de inestabilidad, lo que según las métricas se cumple en la implementación de DELFIN++.

6.5.2. Complejidad de código

Los resultados de complejidad de código obtenidos del análisis realizado con *SourceMonitor* se encuentran en la tabla 6.7.

Se tienen buenos resultados en las métricas de complejidad por profundidad de bloque de código (Max Depth, Avg Depth), en donde la profundidad promedio es siempre menor a 3.0, y la profundidad máxima es menor o igual a 7. Si bien el valor máximo aceptable de profundidad es 6, este solo se supera en TetrahedronDictionary, en el método encargado de la lectura de los archivos que contienen la teselación.

La complejidad calculada para las clases de la implementación de métricas de densidad se encuentra en valores dentro del rango aceptable (Avg Complexity <4.0). Sin embargo, las clases encargadas de la búsqueda de vacíos superan el valor máximo aceptable de complejidad. Esta complejidad se debe a que son las clases en las que se encuentra la implementación del algoritmo y el reporte de la información de los vacíos, por lo que se puede revisar la necesidad de disminuir la complejidad, pero no es prioritario.

Considerando la complejidad por herencia, los métricas DIT y NOC se mantienen en valores bajos, por lo que se evita aumentar la complejidad, y se mantiene el control sobre la predicción de su comportamiento.

Name	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Dictionary	2.8	3	2	0.96	1.50
PointDictionary	7.8	5	5	1.53	2.17
Edge	1.7	2	2	0.74	1.07
EdgeDictionary	4.7	3	2	1.15	1.29
Tetrahedron	6.3	5	4	1.22	1.76
TetrahedronDictionary	9.5	15	7	2.00	2.82
ElementDensity	6.3	10	3	1.77	3.25
EdgeDensity	8.4	6	4	1.91	1.71
EdgeLengthDensity	18.0	3	3	1.30	3.00
EdgeVolumeDensity	18.0	3	3	1.30	3.00
PointDensity	7.0	6	4	1.82	1.63
PointLengthDensity	23.0	4	3	1.28	4.00
PointVolumeDensity	16.0	4	3	1.24	4.00
VoidBuilder	21.3	9	5	2.37	5.50
DelfinBuilder	11.3	13	6	1.89	4.17
DensityBuilder	5.9	10	5	1.69	3.20
EdgeDensityBuilder	16.5	9	4	1.49	5.00
PointDensityBuilder	17.5	9	4	1.53	5.00
VoidJoiner	19.0	10	5	2.52	4.50
EdgeJoiner	10.8	15	5	1.67	5.20
VoidSpace	6.0	12	6	1.91	2.95
VoidShape	8.8	11	4	1.30	2.50

Tabla 6.7: Métricas de complejidad de código de la implementación de DELFIN++

6.5.3. Cobertura de código

Los resultados de cobertura de código por los test unitarios para DELFIN++ se presentan en detalle en la tabla 6.8, y en resumen en la tabla 6.9.

Un 79.2 % de las funciones son ejecutadas en los test unitarios, lo que corresponde al 71.3 % de las líneas. Las clases que -según las métricas de calidad presentadas previamente- necesitan pruebas de código más completas (Edge, EdgeDictionary, TetrahedronDictionary y PointDictionary) tienen a lo menos el 89 % de sus líneas ejecutadas en los tests, cumpliendo el porcentaje mínimo de cobertura propuesto (75 %).

Las clases abstractas o plantillas como Dictionary, ElementDensity, VoidBuilder y VoidJoiner presentan bajo porcentaje de cobertura, pero de todas formas incluyen ejecuciones de funciones definidas en ellas, siendo 33.3 % (en VoidBuilder y VoidJoiner) el menor porcentaje de cobertura de funciones en un archivo. Sus clases concretas, en cambio, tienen un mayor porcentaje de cobertura. Por ejemplo, las cuatro clases en las que se implementan las métricas de densidad se testean en un 100 %.

Los archivos con utilidades auxiliares para cálculos geométricos (geom3d.cpp y meshUtils.cpp) son completamente testeados, tanto en funciones como en líneas ejecutadas. Debido a esto se encuentra y soluciona un error en la implementación del cálculo del tercer vecino

Archivo	Líneas		Funciones	
DensityBuilder.cpp	86.3 %	63 / 73	81.8 %	9 / 11
Dictionary.hpp	92.9 %	13 / 14	57.1 %	8 / 14
Edge.cpp	100.0 %	49 / 49	100.0 %	15 / 15
EdgeDensity.cpp	18.9 %	14 / 74	62.5 %	5 / 8
EdgeDictionary.cpp	97.6 %	40 / 41	75.0 %	6 / 8
EdgeDictionary.hpp	100.0 %	1 / 1	100.0 %	1 / 1
EdgeJoiner.cpp	94.8 %	55 / 58	66.7 %	4 / 6
EdgeLengthDensity.cpp	100.0 %	18 / 18	100.0 %	1 / 1
EdgeVolumeDensity.cpp	100.0 %	18 / 18	100.0 %	1 / 1
ElementDensity.hpp	7.4 %	2 / 27	40.0 %	2 / 5
PointDensity.cpp	22.2 %	16 / 72	66.7 %	6 / 9
PointDictionary.cpp	89.1 %	49 / 55	71.4 %	5 / 7
PointLengthDensity.cpp	100.0 %	24 / 24	100.0 %	1 / 1
PointVolumeDensity.cpp	100.0 %	18 / 18	100.0 %	1 / 1
Tetrahedron.cpp	99.3 %	152 / 153	100.0 %	21 / 21
TetrahedronDictionary.cpp	89.3 %	108 / 121	83.3 %	10 / 12
VoidBuilder.hpp	50.0 %	1 / 2	33.3 %	1 / 3
VoidJoiner.cpp	16.1 %	14 / 87	50.0 %	2 / 4
VoidJoiner.hpp	50.0 %	1 / 2	33.3 %	1 / 3
VoidShape.cpp	87.8 %	65 / 74	87.5 %	7 / 8
VoidSpace.cpp	97.1 %	133 / 137	100.0 %	19 / 19
geom3d.cpp	100.0 %	41 / 41	100.0 %	6 / 6
meshUtils.cpp	100.0 %	24 / 24	100.0 %	1 / 1

Tabla 6.8: Cobertura de código por los test de DELFIN++.

	Alcance	Total	Cobertura
Líneas	945	1326	71.3 %
Funciones	141	178	79.2 %

Tabla 6.9: Resumen de cobertura de código por los tests de DELFIN++.

más cercano r_3 (ec. 2.3).

En general, se cumple con los requisitos de cobertura establecidos en el capítulo 3, y se comprueba la correcta implementación de los cálculos de métricas de densidad, calidad y forma.

6.5.4. Manejo de memoria

En el capítulo 4 se mostró la existencia de fugas de memoria en la implementación de DELFIN. Luego de esta evaluación se detectó el origen de estas fallas y se eliminaron de la implementación, lo cual se logró liberando la memoria utilizada por variables de instancia en el caso de los errores asociados a TetrahedronDictionary (previamente llamado FacetDictionary), y mejorando el manejo de variables locales con memoria asignada dinámicamente en el error de DelfinBuilder.

Luego de extender el programa se realiza un nuevo análisis con *Memcheck*. Esta vez se evalúan las dos versiones del algoritmo, DELFIN para verificar que no queden fugas en la implementación, y DELFIN++ para verificar que no se introdujeron nuevas fugas. Para estos se utiliza el mismo conjunto de 52236 puntos con el cual se evaluó previamente a DELFIN. Los resultados se encuentran en las figuras 6.21 y 6.22.

```
==35==
==35== HEAP SUMMARY:
==35==     in use at exit: 0 bytes in 0 blocks
==35==   total heap usage: 11,534,590 allocs, 11,534,590 frees, 662,945,995 bytes allocated
==35==
==35== All heap blocks were freed -- no leaks are possible
==35==
==35== For lists of detected and suppressed errors, rerun with: -s
==35== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 6.21: Resultados de *Memcheck* sobre DELFIN++.

```
==589==
==589== HEAP SUMMARY:
==589==     in use at exit: 0 bytes in 0 blocks
==589==   total heap usage: 103,917,840 allocs, 103,917,840 frees, 2,938,059,802 bytes allocated
==589==
==589== All heap blocks were freed -- no leaks are possible
==589==
==589== For lists of detected and suppressed errors, rerun with: -s
==589== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Figura 6.22: Resultados de *Memcheck* sobre DELFIN, luego de eliminar las fugas.

No se encuentran errores relacionados al manejo de memoria. En todas las versiones del algoritmo se utiliza la clase TetrahedronDictionary para almacenar y manejar la teselación, y a partir de los análisis de *Memcheck* se puede concluir que la fuga fue solucionada. Además, no existen fugas asociadas a los archivos con nuevas utilidades y clases de la implementación.

Conclusión

En este trabajo se realiza una extensión al algoritmo de búsqueda de vacíos cosmológicos basado en teselaciones de Delaunay. Para esto se evalúa la calidad de la implementación y se identifican errores en la misma. Se proponen dos métricas de densidad de puntos, y dos de densidad de aristas, las cuales utilizan propiedades ya presentes en la tetraedrización, como lo es el volumen de los tetraedros y la longitud de aristas. Estas métricas generan un campo de densidad en la teselación, el cual se utiliza para generar poliedros en las zonas de baja densidad, los cuales se clasifican como vacíos.

En la implementación de esta extensión se solucionan errores de cálculo y de manejo de memoria presentes en DELFIN. Además, se aumenta la cantidad de pruebas de código realizadas para las funciones previamente existentes, y se generan nuevas para las funciones agregadas en el desarrollo de la extensión. La implementación se realizó siguiendo patrones de diseño orientado a objetos, facilitando la introducción de nuevas métricas de densidad y de nuevos algoritmos basados en tales métricas.

Considerando lo presentado a lo largo de este informe, es posible concluir que se cumplió con el objetivo principal de este trabajo. El algoritmo DELFIN++ (nombre con el que se reconoce la nueva versión) se valida sobre datos artificiales y sobre un catálogo de datos observacionales con muestras limitadas por volumen.

Los objetivos específicos se cumplen en su mayoría. Para comparar DELFIN++ con la versión previa se utilizan los datos artificiales, sobre los cuales se logra una mejora en la detección de los vacíos, no solo en la recuperación del volumen de los mismos, sino que también se presentan menos errores asociados a fragmentación y detección de zonas que no corresponden a vacíos. Relacionado a esto último, la identificación y almacenamiento de vacíos en el borde de la muestra de datos se modifica, introduciendo el cálculo de la relación de aspecto de los tetraedros para detectar cuando la superficie se degenera por la mala calidad de aquellos símplices cercanos a la frontera.

Sobre la información entregada para identificar a los vacíos, de las métricas de densidad de puntos se obtienen los datos necesarios para generar reportes que contengan esta magnitud. Además, se trabaja con el momento de inercia de la nube de galaxias que componen un vacío para generar reportes de la forma, aumentando los parámetros con los cuales se describen.

De la validación con datos de catálogo, se concluye que se detectan las zonas de baja densidad, las cuales suelen coincidir con las zonas encontradas por el algoritmo de Sutter, pero tienden a ocupar menos volumen que estas últimas. Para estas muestras, DELFIN++ obtuvo

mejores resultados al utilizar la versión que trabaja con densidad de puntos. Siguiendo con las validaciones, en el análisis de escalabilidad el algoritmo DELFIN++ prueba ser eficiente, ejecutándose en tiempo $O(n \log n)$.

El objetivo específico que no se logra cumplir es el generar acceso al algoritmo y a sus resultados. Si bien se puede tener un repositorio público con el código, para publicarlo se pretende, al menos, realizar ajustes al proceso de generación de la teselación, debido a que las muestras utilizadas en trabajos de identificación de vacíos son limitadas por volumen, y la teselación generada con el proceso actual genera múltiples tetraedros fuera del dominio de los datos, alterando los datos.

Trabajo Futuro

Algunas propuestas para mejorar la solución se han mencionado en el capítulo 6. En la sección 6.4 se propone paralelizar el proceso de descartar y modificar los vacíos, pues en este punto del algoritmo ya se terminaron de procesar los puntos (o aristas) de baja densidad para generar las semillas de los vacíos, y no existirían condiciones de carrera sobre los tetraedros y sus elementos. En la sección 6.3 se propone que al calcular la densidad promedio de la muestra y al evaluar los puntos para que formen parte de un vacío se ignoren aquellos que pertenecen a tetraedros que son parte del borde, debido a que los tetraedros de mala calidad tienden a alterar las densidades de los puntos y la forma de los vacíos.

Modificar la generación de la teselación del algoritmo para tratar con las muestras limitadas por volumen es un trabajo futuro que debe ser realizado, esto para que el flujo de trabajo con DELFIN++ no sea tan complejo ni dependa de parámetros como el α, que define el α-complex con el cual se calcula la superficie que delimita el volumen de la teselación de Delaunay restringida.

Los datos artificiales con los que se trabajó estaban contenidos dentro de un volumen cúbico. Sin embargo, durante el desarrollo de la validación se reconoció que las muestras con las que se suele trabajar son cortes radiales del universo observado. Por este motivo se propone que se genere una variedad de datos artificiales en que las galaxias y vacíos estén contenidos en un volumen con la forma recién mencionada, así la validación puede guiar las modificaciones y alertar de errores desde las pruebas iniciales, que suelen ser con datos creados a partir de simulaciones.

Finalmente, se reconoce la necesidad de ampliar el alcance de las validaciones. Por ejemplo, se puede realizar una comparación con la intersección de las superficies de los poliedros generados por distintos algoritmos. También se pueden realizar estudios sobre la elipticidad de los vacíos, y comparar resultados con algoritmos que utilizan distintas geometrías, como mallas cúbicas o esferas maximales.

Bibliografía

[1] J. Ahumada. Validación astrofísica de algoritmo DELFIN. Trabajo dirigido, Universidad de Chile, 2019.

[2] Rodrigo Alonso. *A Delaunay Tessellation Based Void Finder Algorithm*. Tesis de magíster en ciencias, mención computación, Universidad de Chile, Santiago de Chile, 2016.

[3] Rodrigo Alonso, José Ojeda, Nancy Hitschfeld, Carlos Hervías, and Luis E. Campusano. Delaunay based algorithm for finding polygonal voids in planar point sets. *Astronomy and Computing*, 22:48–62, 2018.

[4] M. A. Aragon-Calvo and A. S. Szalay. The hierarchical structure and dynamics of voids. *Monthly Notices of the Royal Astronomical Society*, 428(4):3409–3424, December 2012.

[5] Astropy Collaboration. Astropy: A community Python package for astronomy. *Astronomy and Astrophysics*, 558:A33, October 2013.

[6] Astropy Collaboration and Astropy Contributors. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *Astronomical Journal*, 156(3):123, September 2018.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[8] I. Chowdhury and M. Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294 – 313, 2011. Special Issue on Security and Dependability Assurance of Software Architectures.

[9] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[10] Jörg M. Colberg, Frazer Pearce, Caroline Foster, Erwin Platen, Riccardo Brunino, Mark Neyrinck, Spyros Basilakos, Anthony Fairall, Hume Feldman, Stefan Gottlöber, and et al. The Aspen–Amsterdam void finder comparison project. *Monthly Notices of the Royal Astronomical Society*, 387(2):933–944, June 2008.

[11] H. El-Ad and T. Piran. Voids in the large-scale structure. *The Astrophysical Journal*,

491(2):421–435, December 1997.

[12] C. Foster and L. A. Nelson. The Size, Shape and Orientation of Cosmological Voids in the Sloan Digital Sky Survey. *The Astrophysical Journal*, 699(2):1252–1260, June 2009.

[13] S. Gottlober, E. L. Lokas, A. Klypin, and Y. Hoffman. The structure of voids. *Monthly Notices of the Royal Astronomical Society*, 344(3):715–724, September 2003.

[14] S. A. Gregory and L. A. Thompson. The coma/A1367 supercluster and its environs. *The Astrophysical Journal*, 222:784, June 1978.

[15] C. Hervías, N. Hitschfeld-Kahler, L. E. Campusano, and G. Font. On finding large polygonal voids using delaunay triangulation: The case of planar point sets. In *Proceedings of the 22nd International Meshing Roundtable*, pages 275–292, 2014.

[16] F. Hoyle and M. S. Vogeley. Voids in the Point Source Catalogue Survey and the Updated Zwicky Catalog. *The Astrophysical Journal*, 566(2):641–651, February 2002.

[17] M. Joeveer, J. Einasto, and E. Tago. Spatial distribution of galaxies and of clusters of galaxies in the southern galactic hemisphere. *Monthly Notices of the Royal Astronomical Society*, 185(2):357–370, November 1978.

[18] C. Johnson, R. Patel, Deyanira Radcliffe, P. Lee, and J. Nguyen. Establishing quantitative software metrics in department of the navy programs. 2015.

[19] G. Kauffmann and A. P. Fairall. Voids in the distribution of galaxies: an assessment of their significance and derivation of a void spectrum. *Monthly Notices of the Royal Astronomical Society*, 248(2):313–324, January 1991.

[20] R. P. Kirshner, Jr. Oemler A., P. L. Schechter, and S. A. Shectman. A million cubic megaparsec void in bootes. *The Astrophysical Journal*, 248:L57, September 1981.

[21] C. Lobos. Towards a unified measurement of quality for mixed–elements. Technical Report 2015/01, Departamento de Informática, UTFSM, April 2015.

[22] Nicholas Nethercote and Julian Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, 42(6):89–100, June 2007.

[23] M. C. Neyrinck. ZOBOV: a parameter-free void-finding algorithm. *Monthly Notices of the Royal Astronomical Society*, 386(4):2101–2109, June 2008.

[24] S. G. Patiri, F. Prada, J. Holtzman, A. Klypin, and J. Betancort-Rijo. The properties of galaxies in voids. *Monthly Notices of the Royal Astronomical Society*, 372(4):1710–1720, November 2006.

[25] E. Platen, R. van de Weygaert, and B. J. T. Jones. A cosmic watershed: the WVF void detection technique. *Monthly Notices of the Royal Astronomical Society*, 380(2):551–570, September 2007.

[26] Hang Si. TetGen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software*, 41(2):1–36, February 2015.

[27] P. M. Sutter, Alice Pisani, Benjamin D. Wandelt, and David H. Weinberg. A measurement of the alcock–paczyński effect using cosmic voids in the SDSS. *Monthly Notices of the Royal Astronomical Society*, 443(4):2983–2990, August 2014.

[28] P.M. Sutter, G. Lavaux, N. Hamaus, A. Pisani, B.D. Wandelt, M. Warren, F. Villaescusa-Navarro, P. Zivick, Q. Mao, and B.B. Thompson. VIDE: The Void IDentification and Examination toolkit. *Astronomy and Computing*, 9:1 – 9, 2015.

[29] R. van de Weygaert and E. Platen. Cosmic voids: structure, dynamics and galaxies. *International Journal of Modern Physics: Conference Series*, 01:41–66, January 2011.

[30] T. W. Williams, M. R. Mercer, J. P. Mucha, and R. Kapur. Code coverage, what does it mean in terms of quality? In *Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity (Cat. No.01CH37179)*, pages 420–424, 2001.

Apéndices

A. Ajuste lineal de tiempos de ejecución

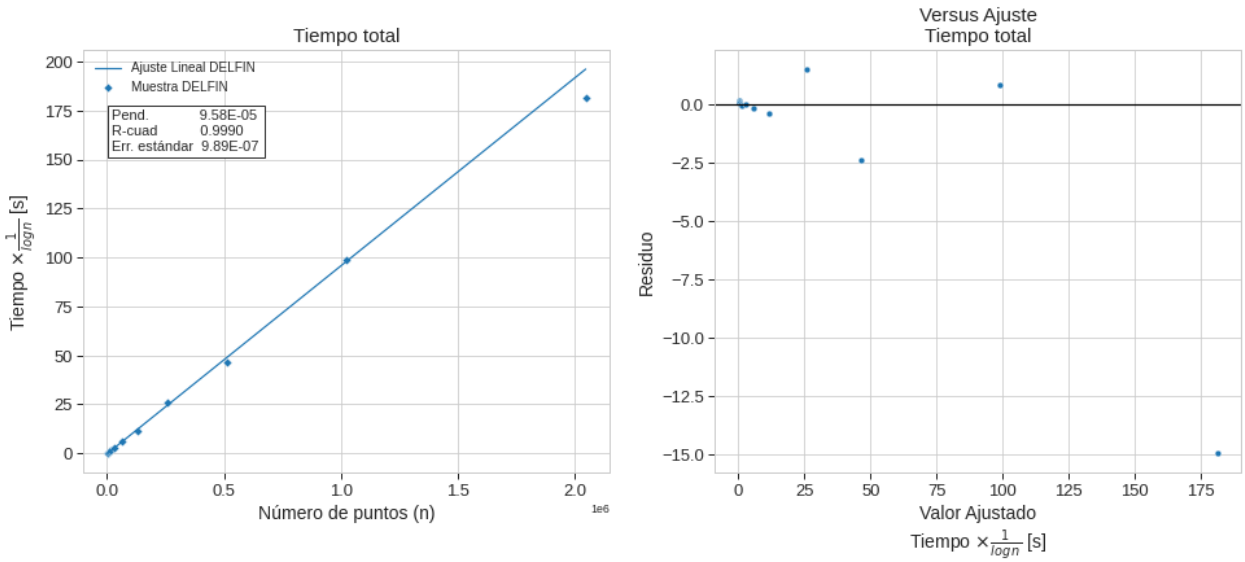


Figura A.1: Ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN. La pendiente de la recta es $m = 9{,}58\text{e}{-05}$, la ordenada en el origen es $b = -0{,}170$ y el coeficiente de correlación es $R^2 = 0{,}9990$.

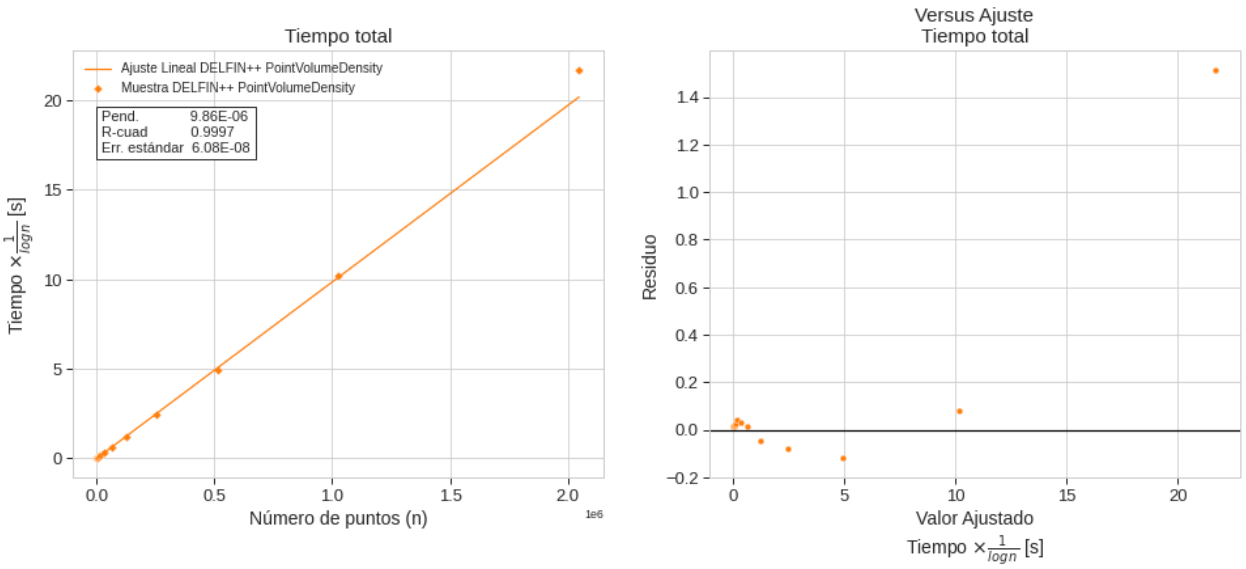


Figura A.2: Ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN++ en su versión de densidad de puntos según el volumen de sus tetraedros. La pendiente de la recta es $m = 9{,}86\text{e}{-06}$, la ordenada en el origen es $b = -0{,}013$ y el coeficiente de correlación es $R^2 = 0{,}9997$.

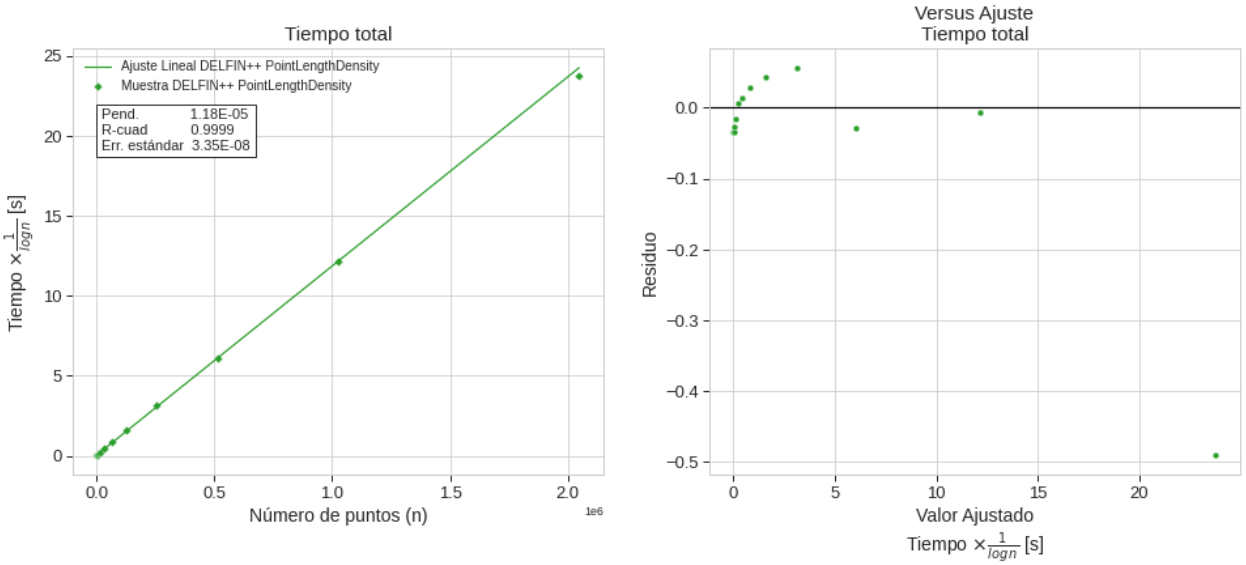


Figura A.3: Ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN++ en su versión de densidad de puntos según el largo de sus aristas. La pendiente de la recta es $m = 1{,}18\text{e}{-05}$, la ordenada en el origen es $b = 0{,}043$ y el coeficiente de correlación es $R^2 = 0{,}9999$.

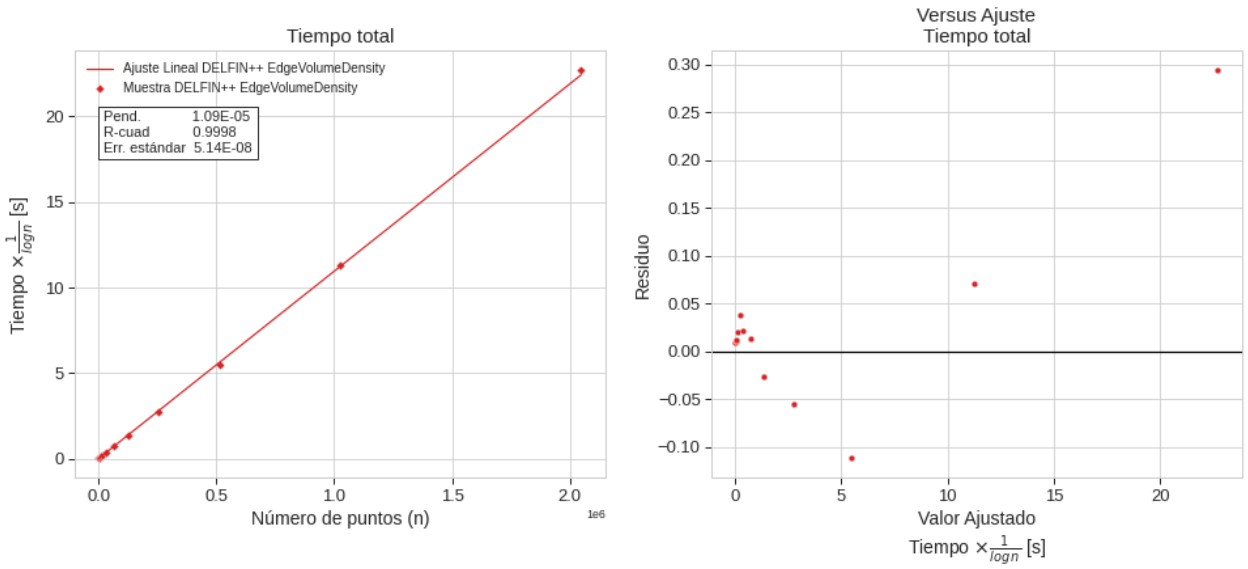


Figura A.4: Ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN++ en su versión de densidad de aristas según el volumen de sus tetraedros. La pendiente de la recta es $m = 1{,}09\text{e}{-}05$, la ordenada en el origen es $b = -0{,}002$ y el coeficiente de correlación es $R^2 = 0{,}9998$.

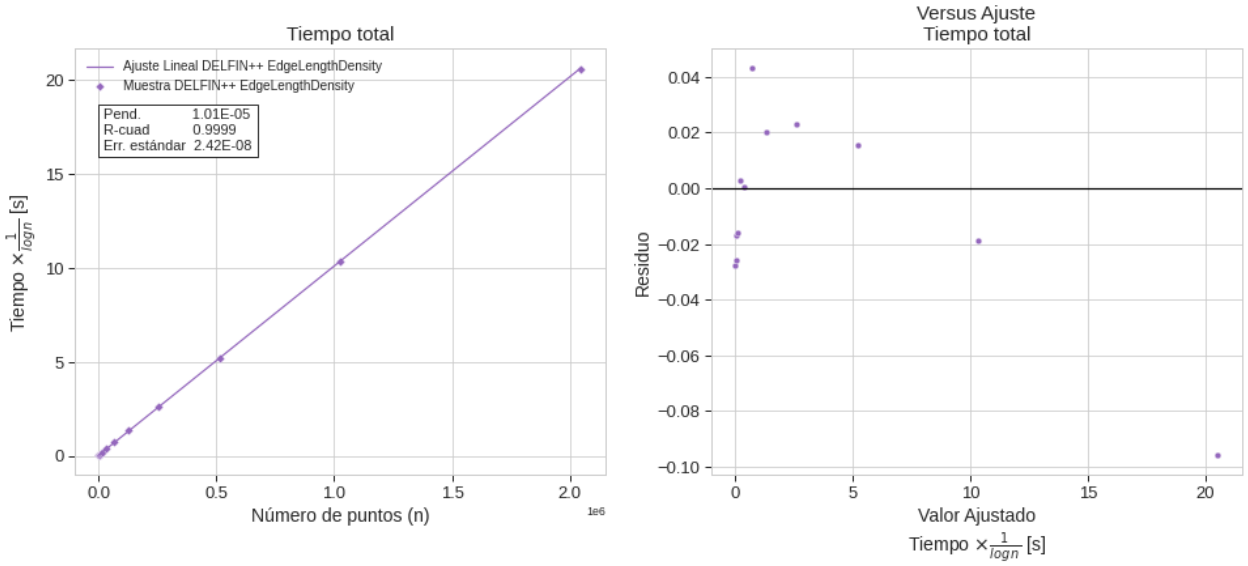


Figura A.5: Ajuste lineal sobre los pares $(n, t/\log n)$, donde n es el número de puntos y t es el tiempo de ejecución de DELFIN++ en su versión de densidad de aristas según su longitud y el volumen de sus tetraedros. La pendiente de la recta es $m = 1{,}01\text{e}{-}05$, la ordenada en el origen es $b = 0{,}034$ y el coeficiente de correlación es $R^2 = 0{,}9999$.