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

A DELAUNAY TESSELLATION BASED VOID FINDER ALGORITHM

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

RODRIGO IGNACIO ALONSO ORTEGA

PROFESOR GUÍA:
NANCY HITSCHFELD-KAHLER

PROFESOR COGUÍA:
LUIS CAMPUSANO BROWN

MIEMBROS DE LA COMISIÓN:
JÉRÉMY BARBAY
BENJAMÍN BUSTOS CÁRDENAS
PABLO PÉREZ LANTERO

Este trabajo ha sido parcialmente financiado por Proyecto ENL009/15, VID 2015 y Proyecto Anillo ACT1122

SANTIAGO DE CHILE
2016

Resumen

En el campo de la cosmología, los vacíos son regiones del espacio cuya densidad es notablemente menor que la densidad de fondo, abarcando distancias en el rango de 20–50 Mpc/h (1 Mpc/h $\sim 2{,}10 \times 10^{19}$ km). Los primeros vacíos fueron descubiertos en los primeros catálogos sistemáticos de galaxias lejanas a fines de la década de 1970. Sus propiedades han sido reconocidas como críticas para la comprensión de la estructura a gran escala del universo. Múltiples esfuerzos han sido destinados al estudio de los vacíos cósmicos para una mejor comprensión de las etapas tempranas y posterior evolución del universo a gran escala, mediante el refinamiento y validación de los modelos cósmicos actuales. La tarea de detectar vacíos, sin embargo, no es trivial y es subjetiva pues la definición de vacío es algo ambigua. Hasta el día de hoy diversos autores continúan investigando este campo, por ejemplo mediante el diseño de mejores algoritmos para detectarlos dentro de catálogos o *surveys* de galaxias.

Considerando lo anterior, hemos desarrollado un algoritmo de detección de vacíos basado en teselaciones de Delaunay: el algoritmo DELFIN (**DEL**aunay Edge Void **FIN**der) que se caracteriza por ser robusto, eficiente y extensible (tanto en 2-d y 3-d), aplicable en grandes catálogos de galaxias. Hemos alcanzado estas características mediante modificaciones y extensiones sobre el algoritmo de Hervías et al. publicado en 2014. Nuestro algoritmo comparte algunas similitudes elementales con otros trabajos, pero las teselaciones de Delaunay proveen mayor maleabilidad y mejor rendimiento. Además, hemos validado nuestro algoritmo tanto con datos artificiales como reales, evaluándonos frente a un algoritmo buscador de vacíos ya existente (Foster y Nelson, 2008), con resultados alentadores. Tanto la implementación 2-d como la implementación 3-d (bajo ciertos supuestos) corren en tiempo $O(n \log n)$, donde n es el número de galaxias.

Finalmente, proponemos algunas áreas para futura investigación, a partir de las cuales este algoritmo se vería beneficiado, así como también algunas sugerencias sobre cómo abordar y resolver algunos problemas asociados.

Abstract

In the field of cosmology, voids are regions in space whose density is noticeably lower than the average, with usual dimensions in the range of 20–50 Mpc/h (1 Mpc/h $\sim 2.10 \times 10^{19}$ km). They were discovered after the first systematic surveys of distant galaxies in the late 1970's. Their properties have been identified as critical for the understanding of the large scale structure of the universe. Several efforts have been directed towards the study of cosmic voids for a better comprehension of the early stages and evolution of the large scale universe, through the refinement and validation of the current cosmic models. The task of identifying voids, however, is not trivial and is also subjective, as the definition of a void is somewhat ambiguous. Even today, multiple authors continue research on this field, for example, designing better void finding algorithms for application over both simulated and observed sets of galaxies.

With this in mind, we have developed an algorithm based on Delaunay tessellations: the DELFIN (**DEL**aunay Edge Void **FIN**der) algorithm, which is robust, efficient and extensible (both in 2-d and 3-d), ready to be applied over large galaxy surveys. We have achieved these features through adjustment and extension of several aspects of the 2014 algorithm by Hervías et al. Our algorithm shares some essential properties with existing algorithms, but Delaunay tessellations provide a more malleable and better performing environment. We have completed the validation of the original algorithm by evaluating its output over both artificial and real data sets, comparing our results with Foster and Nelson's algorithm, showing promising results. Both 2-d and 3-d (under certain assumptions) implementations run analytically and empirically in time $O(n \log n)$, where n is the number of galaxies.

Lastly, we propose some areas for future research from which this algorithm would benefit, as well as some suggestions on how to approach and solve some of the issues associated with the algorithm.

A mis padres, que me dieron la vida y mucho más.

Agradecimientos

Quisiera agradecer en primer lugar a mis padres, Ana Ortega y Rodrigo Alonso, por todo el apoyo que me han entregado y por su disposición a aconsejarme cuando lo necesito. Esta tesis no habría sido posible de no haber sido por ellos, en muchos sentidos.

También me gustaría agradecer a mis profesores, en particular a la profesora Nancy Hitschfeld por su amabilidad e infinita disposición y paciencia durante el desarrollo de mi trabajo de tesis. Aprovecharé la ocasión para dar las gracias a Benjamín Bustos, Pablo Barceló y Daniel Perovich, cuyos cursos de primer semestre me terminaron convenciendo de entrar al departamento, y a Jorge Pérez, que revivió mi motivación por programar (la cual se había escondido), gracias Jorge.

A Angélica Aguirre y Sandra Gáez, que siempre se preocupan por nosotros y tienen la mejor disposición. Con su trabajo hacen que el DCC funcione día a día, no logro concebir el departamento de computación sin ustedes.

A mi hermano, Diego Alonso, que me ha enseñado muchas lecciones de vida que indirectamente contribuyeron al desarrollo de esta tesis. A mis amigos: Javier Mallea, pese a las dificultades que te han tocado, siempre estás ahí para apoyar (¡incluso si no entiendes nada de lo que digo!). Nicolás Lehmann, por las interesantes discusiones respecto a tantos temas de nuestras áreas de investigación. Christian von Borries y Francisco Montoto, que me hicieron reflexionar y me apuntaron en el buen camino en momentos de incertidumbre.

Al equipo de Niclabs del 2013-2014, que me entregó tantas herramientas y experiencia, y en particular a Francisco Cifuentes, que ha sido un modelo a seguir (quizá debería hacerlo más seguido).

Y a tantos otros que me ayudaron a completar esta larga pero fructífera etapa y que seguramente olvidé mencionar por descuido (como es habitual en mí).

Estoy y seguiré estando eternamente agradecido de todos ustedes.

Rodrigo Alonso

Contents

List of Figures

Introduction

Over the past few decades, several efforts have been directed to systematically map the visible universe with the aim of providing public, high quality catalogs for research. Projects such as the Sloan Digital Sky Survey (SDSS)[2] have identified both close and far objects from roughly one third of the sky. These large databases are suitable for studies over the large scale structure (LSS) of the universe, and in particular, serves as empirical evidence to contrast with predictions about the evolution of the large scale universe from the Λ Cold Dark Matter (or ΛCDM), the currently accepted cosmological model. One of its both observed and predicted structural elements, cosmic voids, provide us with valuable information about the nature of dark energy, and for this reason, several scholars have studied its properties. This chapter will provide some background about cosmological voids, and introduce the goals and scope of our work. It will also introduce some theoretical tools from the field of geometry: the Delaunay and Voronoi tessellations.

Cosmological Voids

The first time I was asked about the topic of my thesis work, I found it rather difficult to explain what a cosmological void was. In Spanish, my native language, and just as in English, the word *vacío* (void) is usually understood as *vacuum*, and thus my interlocutors' preconceived idea about voids was about absolute emptiness, while this is no longer the consensed characterization. Probably a vestigial noun from the first definitions, it is still in use, perhaps since *void* is more concise than *underdense region* or *underdensity*.

Cosmological voids are vast regions of space nearly devoid of galaxies surrounded by walls, filaments and clusters of galaxies. Together, they comprise a bubbly, foam-like distribution of galaxies, known as the large scale structure of the universe. Although roundish and initially characterized as spheres, voids have more recently been described as potato-shaped and more general polyhedron shapes[4, 12]. They also have a rich substructure as a consequence of void merges and collapses during their formation. While diameters of voids defined by typical L^* galaxies or more luminous are in the order of $10h^{-1}$ – $20h^{-1}$ Mpc, voids defined by rare luminous galaxies can extend to diameters of $20h^{-1}$ – $50h^{-1}$ Mpc, and no upper limit has been found. At lower scales, other studies[17] have found about 30 so called mini-voids, with diameters of $0.7h^{-1}$ – $3.5h^{-1}$ Mpc. Albeit a more detailed definition for a cosmological void would be highly relevant to this introduction to our work, there is no consensus on a standard definition since there are many proposed variants for its boundaries, disjointness,

and as described before, for its shape and size, among other features; without any particular set of features being widely adopted[4, 7, 12].

Nevertheless, the study of cosmological voids is critical for the comprehension and refinement of the current cosmological model, named Λ *Cold Dark Matter* (ΛCDM). According to several authors[4, 14, 16], there are three main motives for the study of voids. Firstly, their study itself helps us understand one of the main components of the LSS universe; these studies serve as a probe to compare theoretical results with observations. Secondly, studies over their shape, size and outflow provide constraints to ΛCDM, in particular over cosmological parameters such as the Hubble constant h, and provide information about the nature of dark energy. And finally, cosmological voids are also a cleaner, more isolated environment to study the evolution of galaxies.

Motivation

An increasing amount of galaxy redshift survey data is becoming available with surveys such as the 2dF Galaxy Redshift Survey (2dFGRS)[5] and the Sloan Digital Sky Survey (SDSS)[2]. These larger surveys allow for systematic study of cosmological voids, something which was not possible before as previous survey coverage was not large enough to contain a significant number of voids. In particular, the SDSS covers about one third of the celestial sphere and it is composed of multiple smaller surveys with different goals and scope, with the current generation (SDSS-IV) having started in 2014 and which will continue until 2020. A different source of void data are those from simulations of the evolution of galaxies in the universe, such as the Millennium Run[15], which are used to contrast cosmological models with observations (see figure 1). Although their general features are consistent with observations, some discrepancies between simulations and observations have been detected, such as an excessive number of void particles generated by ΛCDM compared to observations[13].

For these reasons, multiple void finder algorithms have been developed to help with systematic studies of large scale galaxy surveys, building their own void catalogs as a result. These algorithms receive a sample of a galaxy survey such as SDSS as input, and produces a set of voids present in the sample region. From these catalogs, other researchers may retrieve interesting properties about voids, and they can also compare different algorithms either from the catalogs itself or by using the void finders over the same data set. A major issue when comparing these catalogs is that these algorithms have different shape restrictions and border conditions for voids, some of them do not allow void galaxies in their interior, and even the methods used to represent voids are different: some algorithms use unions of cubes, others use polyhedra, spheres, etc. Some algorithms do not allow any intersection between voids, while others produce hierarchies of voids where larger voids contain smaller ones. They might also differ on the data used to identify voids; those which depend on dark matter density may only be used over simulation data, for example. However, they all seem to agree on voids having a very underdense center, and having very steep edges[4]. As a consequence, it is not trivial to compare different void algorithms and special care must be taken with inherent differences between their results.

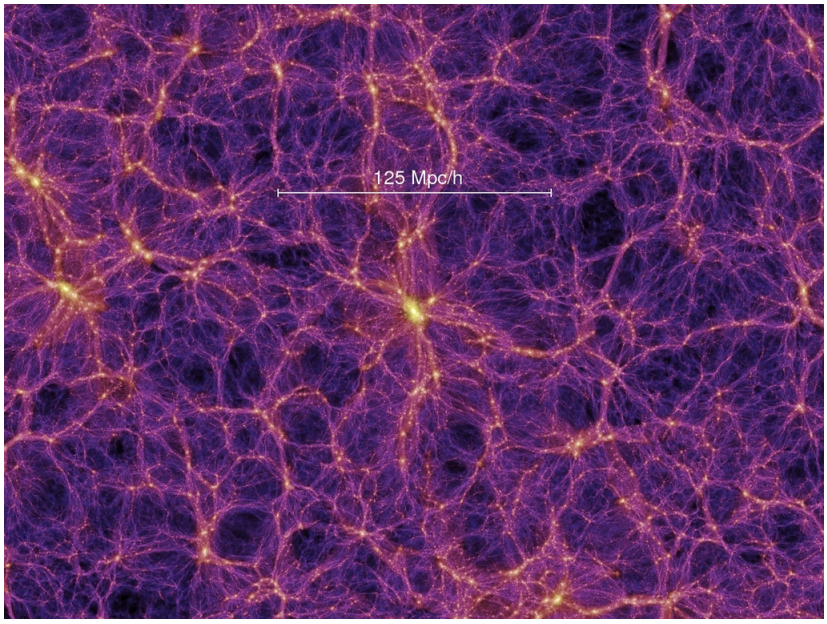


Figure 1: A 15 h^{-1}Mpc thick slice of the Millennium Simulation[15] for $z = 0$.

One of the more recent void finder algorithms is DELFIN[8]. This algorithm is novel in that it is the first of its kind to detect voids by direct use of Delaunay tessellations[1], a triangle (tetrahedron in 3-d) mesh formed by connecting each point (in the context of LSS, a point would be a galaxy or a dark matter particle) to its natural neighbors. The algorithm produces voids by grouping tetrahedra sharing an edge as long as this edge is the longest of one of them. As with other algorithms based on Delaunay or Voronoi tessellations, its simplicity and properties make it a very efficient solution while placing almost no constraints over the shape and size of detected voids. However, a preliminary run of this algorithm over a real data set shows several shortcomings: this algorithm fragments regions which other algorithms would detect as a void, and also tends to produce spurious voids near to the boundaries of the data set.

Objectives

Throughout this work, we intend to develop and validate a robust and efficient 3-d void detecting algorithm, valid for application over large galaxy surveys, and compare it with well known algorithms, based on the original version of DELFIN[8].

Specific Objectives

- Extend the original detection process in the DELFIN void finder algorithm by designing and developing several criteria for subvoid joining.

[1]An earlier work did indeed use Delaunay tessellations to generate a density field estimator, but the tessellation is lost after sampling the field in a grid.

- Optimize the current implementation of DELFIN both by using better data structures and by refactoring the current implementation.
- Design a correct and efficient criteria for identification of border voids, which are likely to be part of a larger void, and thus should not be regarded as complete voids.
- Validate the resulting algorithm. The 3-d version of DELFIN has not been validated before and the 2-d version, while already validated in a previous work[8], must be revalidated after including the extensions mentioned above. In fact, one of the most important objectives is the validation of the algorithm, given the extensions described above.

Methodology

Our work has extended the DELFIN algorithm. Delaunay tessellations are a central element of the DELFIN algorithm, and consequently, considering its features, this work is based on them as well.

The main phases of this work will be:

1. A thorough literature review of existing void finders.
2. Setting an adequate void definition for the scope of this work.
3. Designing criteria to join voids and implementing them. Testing and selecting the most appropriate criteria according to the void definition chosen earlier.
4. Correctness and performance evaluation with artificial data and real galaxy survey data.
5. Validation using artificial data and through comparison with available algorithm implementations.
6. Data structure/algorithm optimizations to reduce computation time. This includes application of design methodologies on each implementation instance with the goal of simplifying further extensions.

For our preliminary tests, artificial data will be used to evaluate edge cases and density contrast tolerance. Data from SDSS will be used to compare our implementation with other algorithms.

Thesis Contents

This thesis is organized as follows: Chapter 1 covers previous work in the context of void finders, along with some common essential geometrical tessellations. Chapter 2 describes the 2-d DELFIN void finder algorithm, together with its validation, results and discussion. Chapter 3 features a description the 3-d DELFIN void finder algorithm and its validation, including a comparison of its results with Foster & Nelson[7] and a discussion of these results. The final chapter summarizes our work and describes areas of this work left open for future research.

Chapter 1

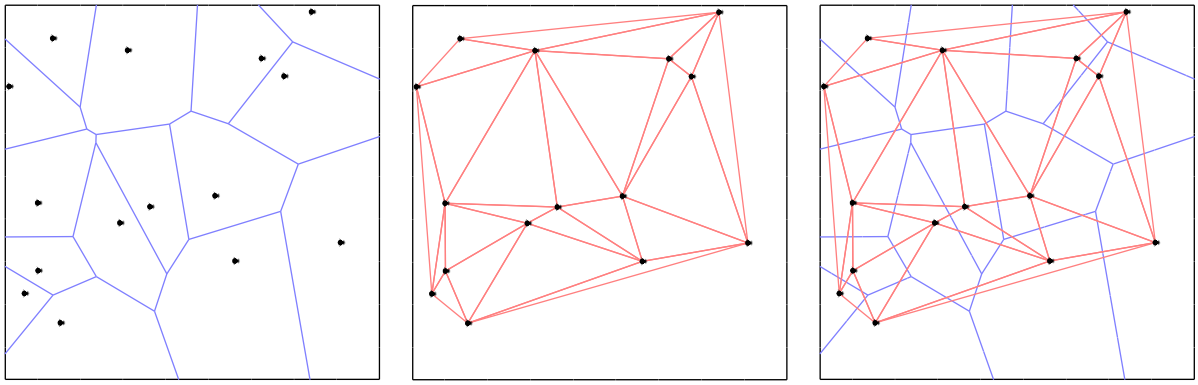
Previous Work

As we saw in the previous chapter, there are several existing void finding algorithms. During this chapter, we will provide a briefing on the void finders' state of the art along with other authors' approaches to the void finding problem. We will highlight how each algorithm influenced others and how they contributed to the void finding scene. But first, we will introduce two geometrical tools used in several of these void finders: the Voronoi diagram and the Delaunay tessellation.

1.1. Voronoi Diagram and Delaunay Tessellation

Given an d-dimensional space H and a finite set S of *sites* (points in H), the *Dirichlet tessellation* or *Voronoi diagram* (VD) of S corresponds to the partition of H into regions, one for each point in S, where each point p in the space belongs to the region associated with the point in S closest to p. In other words, p belongs to the region associated with $s \in S$ iff there is no site $\hat{s} \in S$ closer to p than s. We also define the set of *natural neighbors of* s as those sites in S whose corresponding regions are adjacent to that of s.

The *Delaunay tessellation* (DT) is the dual of the Voronoi diagram (see figure 1.1). Given an d-dimensional space H and a set S of points in H, the DT is a mesh formed by simplices (triangles in 2-d space, tetrahedra in 3-d space) whose edges join pairs of natural neighbors. Note that the simplices do not tessellate the entire space, but only the convex hull of the set S. Additionally, the simplices from this tessellation satisfy the following restriction: given any simplex t from a DT, no point $s \in S$ lies in the interior of the d-dimensional ball circumscribing t. Points, however, may lay over the surface of these simplices, and in fact, each d-ball will indeed have at least $d + 1$ points lying on its surface: the $d + 1$ vertices of the circumscribed simplex. The definition allows for additional points on the surface of the circumballs, which is possible if there is an extra cospherical point; in this rare case, we can still divide the resulting polyhedron into two or more simplices without breaking the Delaunay restriction. From now on, we will restrict to $d = 2$ and $d = 3$, whose respective simplices are triangles and tetrahedra.



(a) Voronoi diagram (b) Delaunay tessellation (c) Both tessellations

Figure 1.1: Voronoi diagram and Delaunay tessellation for a random set of 15 points. Note there is a triangle edge for every pair of adjacent Voronoi cells. Circumcenters for the corresponding Delaunay triangles are located in the three cell intersection for the corresponding Voronoi cells.

It is worth noting that the length of the edges in a Delaunay tessellation provide us with rich information about local density. Every edge e in the tessellation is a chord for some ball B devoid of any points in its interior. So as a chord, e must be of equal or lower length than the diameter d_B of B, implying the existence of an empty ball of diameter greater than or equal to $|e|$.

Furthermore, for every Delaunay tetrahedron circumball B of radius r_B, if its center is inside the convex hull of the point set, then there is an associated edge in the DT of length at least $l = \frac{2\sqrt{6}}{3} r_B$, i.e., the edge length of a regular tetrahedron with B as its circumcircle. The proof consists in fixing any three points p_1, p_2, p_3 on an r_B-radius ball forming a triangle with edge lengths lower than l, and then noting that both the points p_1, p_2, p_3 and the locus of the points on the surface of the ball which are at a distance at most l from all three vertices belong to the same semiball defined by a plane parallel to the triangle $p_1p_2p_3$. Since, by hypothesis, at least one of the DT tetrahedra must contain the center of the ball, such tetrahedron cannot have all of its vertices on the same semiball (for any defining plane), we may conclude that one of its edges must be of length at least l.

Although VDs are frequently used in the field of cosmology to identify high density regions given a set of galaxies, we believe DTs are better suited to detect low density regions without any loss of significant information since, as the dual of VDs, they are just a different representation of the same information. VDs are composed of point-centered regions and thus are better suited to detect high density regions, as opposed to DTs, whose simplices are emptiness-centered regions, while still capturing density differences effectively and thus capture underdensities more naturally than VDs; in fact, from the definition we have given for DTs, it corresponds *exactly* to the adjacency graph for Voronoi regions. Another advantage of DTs is that, although more numerous, its base elements are simplices, whereas VDs are formed by convex polyhedra which are usually harder and more costly to evaluate.

Still, there are some deficiencies of DTs compared to VDs. First of all, under small perturbations, the tessellation might change considerably, specifically in the case where natural

neighbors change. While this is not a problem under VDs, where a change of natural neighbors is reflected by a smooth transition between old and new adjacent cells with continuous edge length changes, in DTs it means that edges and faces are respectively flipped in 2-d and 3-d, a change which is not continuous and thus small variations in point locations are more notorious in DTs than in their dual. However, this situation is very uncommon, as (in a 3-d scenario) it would require at least 5 (almost) cospherical sites without any galaxy on its circumball interior. Another problem with DTs in our opinion, is that they might be less intuitive than VDs, specially in more than two dimensions, as its definition is not as straightforward as with VDs.

1.2. Void Finders

General knowledge about voids' presence has been around since the discovery of the Boöotes void by Kirshner et al.[11]. Their importance quickly became apparent, and currently, as we mentioned earlier in page 2, study of voids is critical to fully comprehend the large scale structure of the universe. Although the first voids were identified by visual inspection, the need for consistent void identification methods pushed towards more objective alternatives, and several void finders were developed over the years. While more exhaustive and deeper redshift surveys have been compiled, the information about the average shape and size of these voids has been refined but is still open to improvement, and thus, void finding tools are still being developed and improved.

The definition of the shape of a void is non trivial: since the gaps between galaxies are considerable even in dense regions (in the order of Mpc), there must be some constraint regulating a minimum width at any given point, to the point there is still no agreement on which these constraints must be.

The first void finder algorithms were designed to find regions completely devoid of galaxies. As modern surveys were obtained by more precise and powerful instruments, alternative definitions were adopted: both theory predictions and observational surveys showed that the interior of these voids are not empty but instead have a lower density of galaxies.

In fact, almost all recent works on voids use an overdensity parameter δ as:

$$\delta = \frac{\rho - \bar{\rho}}{\bar{\rho}} \tag{1.1}$$

where $\bar{\rho}$ is the average density of the full sample and ρ is the average density of the described void. For example, empty voids (such as those described in subsection 2.4.1) have an overdensity parameter equal to -1. Average overdensity values depend on the void definition used, of course, but in general, it follows $-1 \leq \delta \leq -0.5$.

How do void finders compare to each other? In 2008, the Aspen-Amsterdam void finder comparison project[4] addressed this question by analyzing and comparing 13 different void finding algorithms. One of the concerns they mentioned early in their publication was the difficulty of comparing these heterogeneous void finders; having some of them work over

galaxy information and others over the dark matter distribution (which cannot be measured directly, and in fact has not been proven to exist), these methods do not even share the same input data, and thus the latter group can only be used over data from ΛCDM simulations. For that purpose, all algorithms were executed over a sample of the Millennium Run, an n-body simulation representing the evolution of the universe from its early state to the current age. While the spread of most results per void finder was significant, an expected consequence of grouping together density based and geometry based algorithms, the resulting voids did share some very general features; for example, the largest void was found by all 13 algorithms and the galaxy overdensity showed a less extreme spread than anticipated.

While we will not compare them thoroughly here, we will describe the workings of some of these algorithms, highlighting their contributions to the field.

1.2.1. Geometry Based Void Finders

Voidsearch Algorithm

This algorithm by Kauffmann and Fairall (1991)[10] first constructs a cubic grid of cell size half the average distance between galaxies. A base void is formed by forming the largest possible cube (with size at least double the grid size) from the empty and unused cubes in the grid. From this cube, rectangular layers adjacent to a face are appended to the void, with the added restriction of its area covering at least 2/3 of the void face, until no more layers can be added. This is repeated until no more cubes can be built. This is one of the earliest void finders, allowing no intersection between voids and no galaxies in their interior.

Void Finder Algorithm

This algorithm by El-Ad and Piran (1997)[6] is the base of other algorithms such as Hoyle & Vogeley's and Foster & Nelson's. It consists on two phases, namely, the *wall builder* and *void finder* phases.

In the wall builder phase, they define l_m, the distance to a galaxy's m^{th} nearest galaxy. From the average and standard deviation for l_m, they define the wall separation distance $l = \bar{l_m} + \beta\sigma_m$, with $\beta = 1.5$. If a galaxy has at least m other wall galaxies in a radius l, then it is classified as a wall galaxy, otherwise, it is classified as a field galaxy. The value $m = 3$ is used as they claim it is the minimal value allowing filtering of thin chains of galaxies. Although unused, they also suggest drawing the edges between galaxies closer than l as visual aid for analysis.

The void finder phase takes the wall galaxies (i.e., field galaxies are not considered) and finds maximal empty balls starting from the center of each cube of a grid, with no wall galaxies in its interior. These balls are found by expanding an empty ball until a galaxy is on its surface. Then it is expanded in the direction defined by that galaxy towards the center of the ball, until a second galaxy is on its surface. Once more, it is expanded in the

direction defined by the midpoint of the two galaxies towards the center of the ball, until a third galaxy is on its surface. Finally, it is expanded in the direction of the normal to the plane containing the three galaxies towards the center of the ball, until a fourth galaxy is on the surface of the ball.

These maximal balls are joined in several iterations, using decreasing values for d_i and $\xi = 0.85$ (although the values for d_i are not clear in the article, a reasonable set of values for d_i would be setting d_0 to the largest diameter in the dataset and $d_{i+1} = \xi d_i$). On the i-th iteration, all previously unused balls of diameter greater than ξd_i are detected. For this, the previous method to detect maximal empty balls is used on a grid of cell size $l_{cell} = \xi d_i / \sqrt{3}$, starting from cell centers not contained in previously found voids. The cell size guarantees all maximal balls of diameter greater than ξd_i are found; smaller ones might be found, but such balls are discarded, at least for now. Then, balls are grouped together under the following conditions:

- At least one ball in a group must be of diameter greater than d_i; we will name the largest in the group d_{max}.
- Other balls may be added as long as the intersection of its surface with the surface of a ball of the group is a circumference of diameter greater than ξd_{max}.

Finally, any unused balls of diameter $d > d_i$ are joined to an older void.

By classifying certain galaxies as field galaxies, this algorithm allows detection of non empty voids without interference from isolated galaxies. Also, note that these maximal balls correspond to Delaunay circumballs, and their surface intersections are exactly the circumcircle of the shared face in the Delaunay triangulation, although it is unknown to us if the authors knew about this fact.

Hoyle & Vogeley

The algorithm by Hoyle and Vogeley (2002)[9] is based on the *void finder* algorithm described above. While the wall builder phase and the maximal ball retrieval process have been taken from El-Ad & Piran's work[6], the process is not iterative. Instead, there is a single pass to detect balls on a grid of cell size $l_{cell} = l_3$, growing maximal balls from the center of each empty cell.

Once all maximal balls have been detected, the void building process is quite different as well. Spheres are evaluated in order, from largest to smallest. There are three possible outcomes:

- If the ball does not overlap any voids by more than η_1, it becomes a new void (always the case for the largest ball).
- If it overlaps a single void by more than η_1, it becomes a member of that void.
- If it overlaps more than one void by η_1, then the ball is discarded.

They select $\eta_1 = 10\%$ claiming that voids obtained with it visually appear as distinct regions. To prevent identifying wall gaps as voids, they also add a threshold of $10h^{-1}$ Mpc for the minimum size of voids; only balls larger than this are evaluated as mentioned.

The last step of the algorithm is void volume enhancement. Balls smaller than $10h^{-1}$ Mpc are added to a void if they intersect one of its maximal balls by more than η_2, unless it overlaps with more than one void (in which case it is discarded). The value $\eta_2 = 50\%$ is used because it fills the void without changing the overall spherical shape of voids.

Foster & Nelson

The algorithm by Foster and Nelson (2008)[7] is based on the previously described Hoyle and Vogeley's algorithm. They have made a slight value change of $l = \bar{l_n} + \beta\sigma_n$ from the definition by El-Ad & Piran, using $\beta = 2.0$ instead of 1.5, and reduced the grid cell size by half, allowing smaller balls to be part of a void.

However, the main difference from Hoyle & Vogeley is a more precise distinction between possible outcomes during the void building process, evaluated in the following order:

1. If it overlaps more than one void by η_3 each, then the ball is discarded.
2. If it overlaps a single void by more than η_1, it becomes a member of that void.
3. If the ball does not overlap any voids, it becomes a new void (always the case for the largest ball).

The first condition also replaces the condition set during void volume enhancement (see subsection 1.2.1).

1.2.2. Watershed-based void finders

WVF

The Watershed Void Finder (WVF) algorithm, developed by Platen et al. (2007)[14] involves a minimum of assumption about the scale, structure and shape of voids. It is based upon the Watershed Transform, a well known technique used in geology and image processing. A very condensed description of the algorithm is as follows:

1. A Delaunay Tessellation Field Estimator (DTFE) is used to define a density field over the sample points (redshifts or N-body simulation particles). The density ρ at each sampling point x_i is defined as: $\rho(x_i) = \frac{1+D}{V(\mathcal{W}_i)}$ where $D = 3$ is the space dimension and $V(\mathcal{W}_i)$ is the sum of the volumes of the Delaunay tetrahedron having x_i as a vertex. The density in other points is defined by linear functions. If its vertices are $x_0, \ldots, x_3$, then the density at any point $x = \alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$ in the tetrahedron (i.e.,

with $\sum_{i\in\{0,\dots,3\}} \alpha_i = 1$) is:

$$\rho(x) = \alpha_0\rho(x_0) + \alpha_1\rho(x_1) + \alpha_2\rho(x_2) + \alpha_3\rho(x_3) \tag{1.2}$$

2. Then, a sampling grid, whose cell size is in the order of the interparticle separation, is constructed. For each cell, a number of random Montecarlo samples is chosen and the cell density is defined as the average density at those samples.

3. The grid is filtered using a natural neighbor rank-ordered filtering, in which the density for each sample is replaced by the median, minimum or maximum of its natural neighbors.

4. The image is discretized by uniformly partitioning the cumulative density distribution and replacing each density for the corresponding value in the partition.

5. Using a 2-pixel opening and closing operation (respectively equivalent to a Minkowski subtraction followed by a Minkowski addition, and a Minkowski addition followed by a subtraction) noise is reduced even further.

6. The flooding procedure, based on the watershed transform, is applied: starting from field minima (pixels surrounded exclusively by higher density pixels), the terrain is flooded at increasing levels, adding the flooded points to the basin of the corresponding minima. Each time a pixel is reached by two basins, it is identified as part of their segmentation boundary. When all pixels have been flooded, we are left with *void patches* which tessellate the whole region.

7. The patches are corrected by removing those segmentation boundaries whose density is lower than the typical void underdensity $\Delta = -0.8$.

One of the most interesting properties of this algorithm is how it detects voids relatively parameter free. This property makes it potentially capable of detecting voids of any size. The usage of a density field also allows it to ignore the particular position of particles and instead focus on their vicinity. These properties are shared with other algorithms based on the watershed transform.

ZOBOV

The ZOnes Bordering on Voidness (ZOBOV) algorithm by Neyrinck (2008)[12], uses a similar approach to WVF, but the Voronoi regions of the point set are used instead.

The algorithm begins by obtaining a density field over dark matter particles using a Voronoi Tessellation Field Estimator (VTFE). VTFE is similar to DTFE, but the density estimate for each particle p is $1/V(p)$, where $V(p)$ is the volume of the Voronoi cell associated with particle p. Then it applies a watershed transform-like process where, each Voronoi cell is joined to its lowest density neighbor, resulting in regions named *zones*.

However, because of discreteness noise, these zones are fragments of what the eye would

pick up as voids. For this reason, zones are joined by flooding each zone z, where the height corresponds to the density of the Voronoi cell of lowest density (e.g., a less dense zone would be deeper than a high density zone), until a zone deeper than z is about to be flooded. The process stops at this point, and the flooded zones correspond to a void. If the starting zone is the deepest, the process stops just before flooding the last zone, i.e., it contains all but the most shallow zone).

Note that this algorithm generates overlapping voids; in fact, for every overlap, one of the voids is a subset of the other. This property allows to form void hierarchies, which may hint the void formation processes. For example, a void formed by the merging of two smaller voids might be identified in this hierarchy as two different voids, with one of them as the parent of the other in this void hierarchy.

Chapter 2

The 2-d DELFIN Algorithm

In the first chapter, we have discussed some of the geometrical tools used by void finder algorithms, as well as some of these void finders and their contributions to the field. In this chapter will explain the original 2-d DELFIN algorithm[8], the enhancements we have made since its publication, their validation and discussion.

2.1. The Original Algorithm

The void finder algorithm by Hervías et al., which we have named DELFIN (**DEL**aunay Edge Void **FIN**der), is a fast and precise geometrical approach to the void finding problem. It detects voids by identifying large, non-convex and generally empty regions in space from the Delaunay tessellation of the dataset. Lower density regions in space will have a larger space for an empty sphere inside them than on the rest of space; the same assumption is made by Foster and Nelson[7].

From the previous assumption, the following question arises: Given a bounded volume and a set of galaxies, how do we find the largest empty spheres in this volume? This is a well known problem in computational geometry and its solution lies on the Delaunay tessellation of the set S of points (galaxies).

Given a set of points S, for every circle C devoid of points in S, there is either a circumcircle from a Delaunay triangle which is equal or larger in radius, or C is not completely inside the convex hull of the set of points. This can be easily proven as the center of C must be either outside the convex hull of S, or inside one of the Delaunay triangles t. In this last case, C must be of equal or smaller radius than the circumcircle C_t of the containing triangle t because the circumcircle maximizes the minimum distance to the vertices of t (among the set of points inside t). Therefore, when considering the convex hull of S, the largest empty spheres on its interior are exactly the circumcircles of the Delaunay triangles.

Now that we have fixed a space partition as our building blocks, the following steps would be to choose:

1. Which triangles should be kept as part of a void.

2. When should we consider two adjacent regions as part of the same void, and when should we keep these regions separated.

In the context of void finding algorithms, the general approach has been detecting local minimum densities, followed by expanding these minima towards less underdense regions until a specific criterion is met. To our knowledge, all existing void finders follow this approach in some way, an exception being ZOBOV, described in section 1.2.2, which organizes such adjacent regions in a hierarchy. Nevertheless, ZOBOV still uses a watershed algorithm to select those regions with the lowest local density, joining them towards more dense regions[12]. Platen et al.[14] also follows this pattern over grid cells.

In a similar style, the DELFIN algorithm starts building voids from our own definition of local minima, which considers density over Delaunay edges only (edges connecting natural neighbors). This makes sense since these longest edges guarantee the existence of an empty circle with this edge as a chord[1]. Thus, its local minima are those edges which are the longest for all triangles containing it.

Hereafter, we will use the term *frontier* of a void v as the set of edges shared between a triangle in v and another one not in v.

Then, from each local minimum, it joins triangles adjacent to the void iff their longest edge is one of the edges of the void frontier. The invariant here is that at any given time, all of the longest edges for the triangles in the void at this step are inner (i.e. non frontier) edges. This way, it will not allow any decreases in the local density as it moves away from the starting least dense (but locally longest) edge since the invariant implies that every new void triangle has a longest edge not longer than the longest edge of the void triangle it was joined to. This process can be repeated until no neighboring triangle is connected by its longest edge to a void, but it can be stopped earlier if some criterion is met.

However, if the final process is not stopped earlier by some criterion, these regions will cover the whole convex hull of the point set, and thus some distinction must be made between relevant voids and small, irrelevant variations in local density. For this reason, the algorithm discards every region whose area is less than an user defined threshold value a_v, returning the remaining regions as the output.

Special care must be taken at the boundaries of the data set: since we are working with finite samples, the sampling at the boundaries is incomplete, providing incorrect local density values. As a result, Delaunay triangulations tend to produce skewed triangles (with a large circumcircle radius) close to the data set boundaries on straight (planar) boundaries (see figure 2.2). Even if some voids detected adjacent to the boundary may be real voids, we cannot ensure their completeness as their frontier is not well defined at the sample boundary. We would like to distinguish or discard these voids, and for this reason, the original algorithm evaluates triangles contained in the α-shape of the set of points only, for some value α defined by the user. Discarding the α-shape is equivalent to discarding those triangles whose

[1]But not necessarily as its diameter; in the case of obtuse triangles, the circle having the longest edge as its diameter always encloses the obtuse vertex.

circumcircle C possesses a radius of length greater or equal than $-1/\alpha$.

Algorithm 1: Original DELFIN Algorithm

```
read the Delaunay triangulation inside the alpha shape;
read threshold_value;
void_list = ∅;
order triangles by longest-edge;
repeat
    get the two unused triangles t1, t2 that share the longest-edge;
    triangle_set = [t1, t2];
    label t1, t2 as used;
    foreach t neighbor of triangle_set do
        if The longest edge of t is in the frontier of triangle_set then
            triangle_set = triangle_set ∪ t;
            label t as used;
        end
    end
    if area of triangles in triangle_set ≥ threshold_value then
        void_list = void_list ∪ polygon(triangle_set);
    end
until there is no triangle left to process;
```

The algorithm, as described in Hervías et al.[8], is included as algorithm 1.

The previous steps would often yield adjacent voids, and a natural question about those is whether those regions are effectively different voids or rather belong to a single void. The previous question has no correct answer, since there is no strict definition of a void, particularly in the case of void merging, where the point from which two voids become one has no generally accepted definition; but our observations on runs of the algorithm over artificial data inclined us to evaluate some criteria to choose whether to join them or not. We further discuss this matter in sections 2.2.3 and 3.3.2.

Figure 2.1 represents the sequence of steps followed by the algorithm on a sample (168 points are visible). Figure 2.1a shows the initial set of points. Figure 2.1b shows the Delaunay tessellation of the point set. Figure 2.1c shows the detection of a locally longest edge. Figure 2.1d adds the triangles sharing the locally longest edge to the void. Figures 2.1e–2.1j show several triangles added to the void as their parent triangle (the one with which it shares its longest edge) belongs to the void. Figure 2.1k shows the void in its final state, as the edges constituting its perimeter are not a longest edge for any of the triangles adjacent to the void. Finally, figure 2.1l shows the rest of the regions detected by the same method, with each contiguous region of the same color being a void; note that they tessellate the space. The voids are then discarded if their area is lower than an user defined threshold.

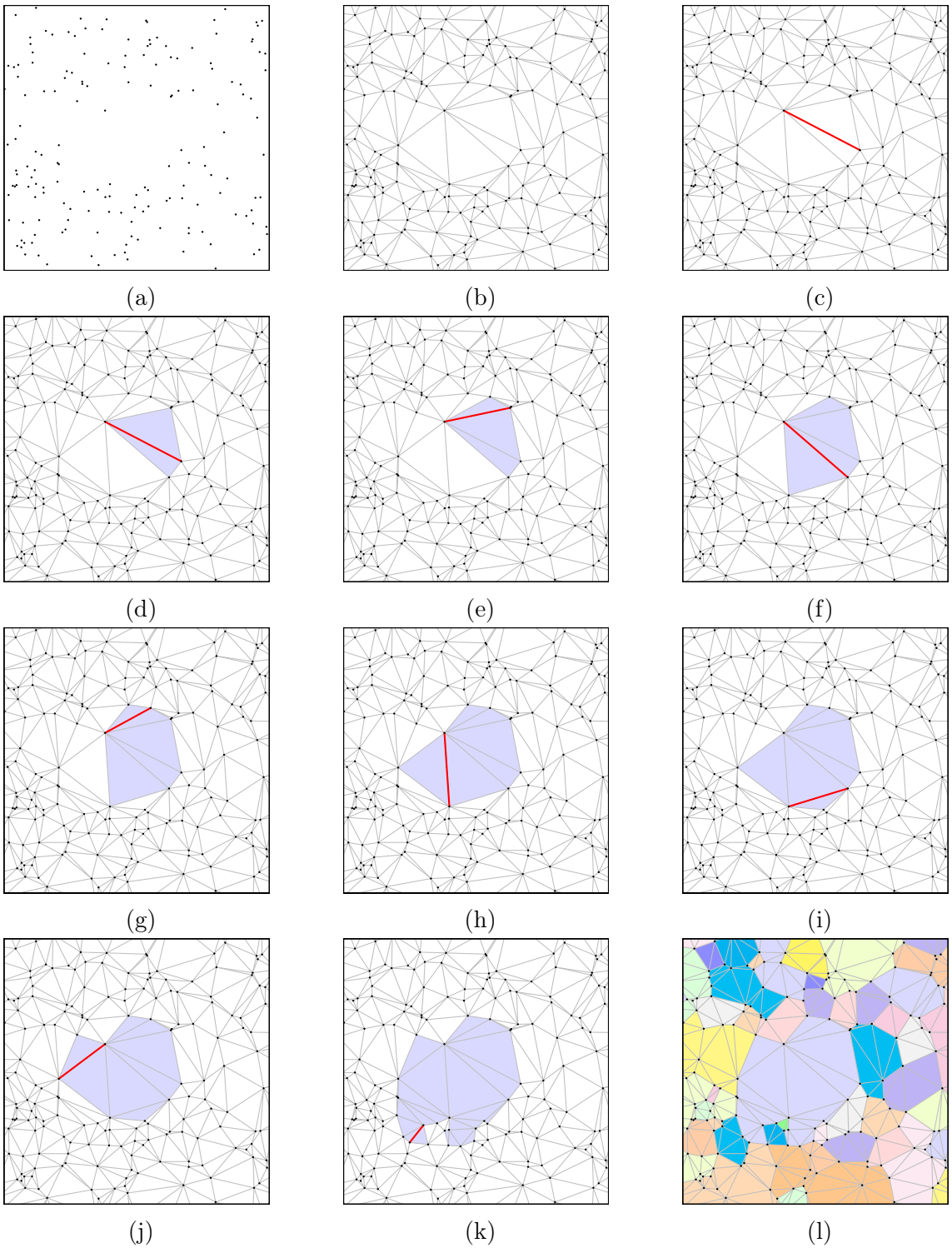


Figure 2.1: A step-by-step figure of DELFIN void retrieval, showcasing the critical parts of the algorithm. The longest edge for the most recently added triangle in the void is shown marked with a thick red line.

2.2. The Extended Algorithm

The current version follows the same spirit as the original algorithm: begin from the Delaunay tessellation of the data set and join each of the resulting triangles to the one with which it shares its longest edge. However, we have enhanced several of its sections, including:

- A different treatment for boundary voids
- A faster approach for void construction
- A new post processing pass

We have continued working with Delaunay tessellations for several reasons. First, the Delaunay tessellation preserves points from the data set while allowing quick access to properties such as density (in terms of natural neighbors) and circumcircles; we can access raw data from the final structures at any time. With the proper data structures, adjacency can be determined in expected constant time, and operations can be done efficiently. Secondly, as we mentioned in section 1.1, the Delaunay tessellation reflects empty regions more naturally than a Voronoi diagram, as each of its triangles is associated with an empty circumcircle; while for Voronoi regions, each of them associates itself with a point. Lastly, Delaunay tessellations have been left relatively unexploited in the context of void finding algorithms. We are aware of two groups, Platen et al.[14] and Way et al.[18] which have directly used Delaunay Tessellations; the former group makes use of a DTFE but later discards the Delaunay tessellation structure in favor of a cubic grid; the latter group has worked with Delaunay tessellations, and our work and theirs indeed share the same building blocks.

A step by step description of the extended algorithm is available as algorithm 2.

To avoid confusion, we redefine the original voids as *prevoids* and leave the term *void* for the final structures obtained by our new algorithm. The following subsections explain our modifications in detail.

2.2.1. Boundary Voids

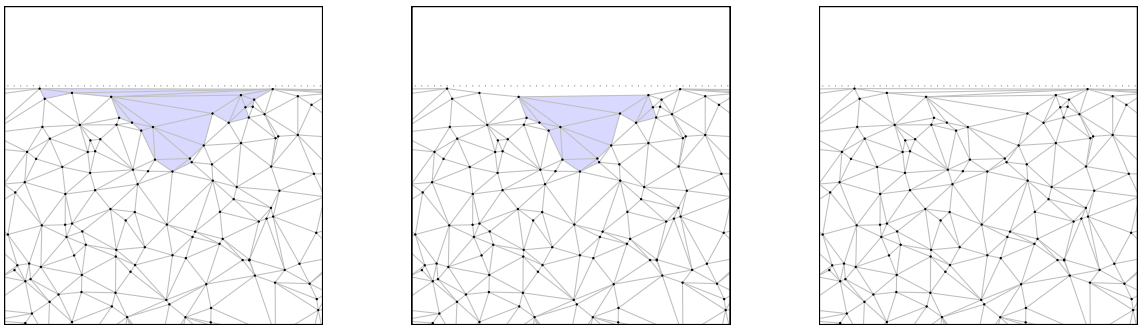
As explained earlier in section 2.1, the original algorithm discards triangles outside the α-shape of the dataset (i.e., whose circumcircle radius is larger than $-1/\alpha$) to handle boundary voids (which might be false positive or incomplete). However, this step does not solve the problem of false positive voids completely and will generally allow incomplete voids in the boundary (figure 2.2b).

For this reason, we have removed the α-shape filter and instead attempted a different approach: we discard any void having one or more edges in the surface of the convex hull of the dataset (figure 2.2c). Alternatively, these voids could be kept for the final user and tagged as incomplete/potentially false instead of discarding them. This approach still allows for a certain distribution of points beyond the boundaries to modify the shape of detected voids and consequently, the possibility of a false positive. However, empirically, situations

Algorithm 2: Extended DELFIN Algorithm

```
read the Delaunay triangulation;
read pre_threshold_value;
read threshold_value;
prevoid_list = ∅;
order triangles by longest-edge;
repeat
    get the two unused triangles t1, t2 that share the longest-edge;
    triangle_set = [t1, t2];
    label t1, t2 as used;
    foreach t neighbor of triangle_set do
        if t shares its longest-edge with a triangle of triangle_set then
            triangle_set = triangle_set ∪ t;
            label t as used;
        end
    end
    if area of triangles in triangle_set ≥ pre_threshold_value then
        prevoid_list = prevoid_list ∪ polygon(triangle_set);
    end
until there is no triangle left to process;
repeat
    label prevoid_list as unchanged;
    void_list = ∅;
    foreach unused prevoid v in prevoid_list do
        void = v;
        label v as used;
        foreach unused prevoid u in prevoid_list do
            if void and u meet the joining criterion then
                void = void ∪ u;
                label u as used;
                label prevoid_list as changed;
            end
        end
        void_list = void_list ∪ {void };
    end
    prevoid_list = void_list;
until prevoid_list is not changed;
remove voids from void_list whose area is less than threshold_value or have a
 boundary edge;
```

where a boundary void is not identified as such are extremely uncommon over convex-shaped samples.



(a) (b) (c)

Figure 2.2: Boundary void handling. 2.2a displays a boundary void, 2.2b displays how the α-shape handles this void by removing edge triangles, and 2.2c displays the result with our new filter. The dotted line indicates the boundary of the sample.

2.2.2. Prevoid Construction and Pruning

The prevoids constructed by the original algorithm can be characterized by a tree defined by the edge-adjacency graph of the prevoid. Since a triangle can only be joined to its parent or children in this representation, we can get each triangle's parent by determining its longest edge, and later ask whether this parent is part of the current prevoid or not. It is worth noting that this step can be done in any order, so if no pruning is done over the void forest, a union-find (disjoint-set) data structure may be used to build voids in quasi-linear time ($O(n\alpha(n))$, where α is the inverse of the Ackermann function and n is the number of triangles). However, usage of a tree pruning depending on the order in which triangles are joined (for example, depending on an accumulated property over the ancestors of a node) will increase the order of the algorithm back to (i.e., $O(n \log n)$).

We experimented with a different pruning criterion from the original (threshold area): threshold longest edge. This criterion discards those prevoids whose longest (inner) edge is shorter than a specified distance $e_{\min}$.

Both criteria allow us to run the algorithm without sorting triangles by their longest edge, as the final deletion only requires a linear traversal over the representatives of prevoids, and in some way, resembles significance analysis from other algorithm.

2.2.3. Prevoid Joining Criteria

While results from the algorithm from Foster et al. and our (unmodified) own over SDSS DR 5[1] overlapped significantly, in many cases our data were represented with noticeable fragmentation compared to Foster's voids. This was studied in the work by Hervías et al.[8] and the conclusion was that the algorithm had issues detecting non-empty voids, usually affecting the Delaunay tessellation in such a way that a void would be divided into smaller fragments. To counter this limitation, we designed and developed a set of methods to decide whether a pair of prevoids should be joined or kept separated. Although their results in 2-d cannot be extrapolated directly to 3-d, we estimated they would provide us with a better

insight over the fragmentation present in our results. Our final goal was to obtain a simple, but adaptive and efficient method to join voids, trying to avoid any preconceived assumptions about their shape.

The following joined criteria were implemented and tested:

Null criterion Leave each prevoid as-is, i.e. each prevoid is output as a void.

This criterion is used as reference, and corresponds to the unmodified version of DELFIN.

Arc criterion For each prevoid, obtain its equivalent circle, i.e. a circle of the same area and centroid as the prevoid. Two prevoids are joined into a new prevoid if and only if the arc angle of the larger equivalent circle inside the smaller circle is greater than a constant $\theta = \frac{\pi}{3}$.

This criterion will only consider the size difference and center distance relative to their radii. As a result, it will tend to join those void pairs whose centers are close respect to their size and which have a small relative difference in area; the size difference is an important factor, as radii ratios over 2 will never be joined. However, it is not effective to join pieces of arbitrarily elongated voids.

Border criterion Two prevoids are joined together if and only if the ratio between their shared perimeter and the largest perimeter (between the two prevoids) is larger than a constant $\beta = \frac{1}{16}$.

This criterion also ignores the prevoids' absolute size. As a result, it will favor those void pairs sharing a large contact area as long as their relative difference in area is not large. This criterion is more strict regarding prevoid adjacency compared to the previous method. However, it is still ineffective on joining pieces of elongated voids, as it relies on features associated with full prevoids instead of its triangles.

Second-longest edge criterion Two prevoids are joined if and only if one of the edges they share is the second longest edge for two tetrahedra, one on each prevoid.

This simple but fast criterion considers neither the absolute nor relative void size difference. It also ignores global properties of the voids involved, depending on triangle properties instead, making it a good algorithm to detect voids of arbitrary elongation. However, it might join two pieces belonging to different voids, especially where density variations are less pronounced.

Threshold edge criterion Two prevoids are joined if and only if one of the edges they share is longer than a parameter r_3 defined as

$$r_3 = d_3 + \lambda\sigma_3 \tag{2.1}$$

where d_3 is the average distance from each point to its third nearest neighbor, λ is a parameter usually set equal to 2.0, and σ_3 is the standard deviation for the third nearest neighbor distance[7].

Note that all of these criteria are independent from the scale of prevoids, i.e., given the

same prevoids, the same void joins will be produced even if prevoids were scaled by a positive factor.

A second area filter a_p ($a_p \leq a_v$) was added to allow for more flexibility on the sizes of prevoids which could be found. This filter a_p is used to discard prevoids, while a_v is used after the joining process.

2.3. Implementation

By the beginning of this thesis work, a 2d implementation of the original algorithm was already in existence (which was used in Hervías et al.[8]), as well as a non validated 3d prototype. The original 2d implementation consists on a Python 2 code, using *qdelaunay*[3] externally to obtain the Delaunay triangulation of a set of points, as well as the libraries *numpy* and *shapely* for general numerical operations and point in polygon checks, respectively.

The code was revised extensively. Apart from the updates regarding the algorithm extensions described previously, the code was refactored and the documentation improved. Command line parameters can now set different variables which originally had to be either introduced during runtime or edited directly on the Python source. The joining criterion to be used is also selectable thanks to the introduction of a parameter. As a result, the writing of scripts which invoke the implementation is now easier and more concise. Various refactorings increased code legibility and eliminated unnecessary structures, significantly improving runtime performance (see section 2.4.4).

2.3.1. Structure

Figure 2.3 shows the class diagram for our 2-d implementation.

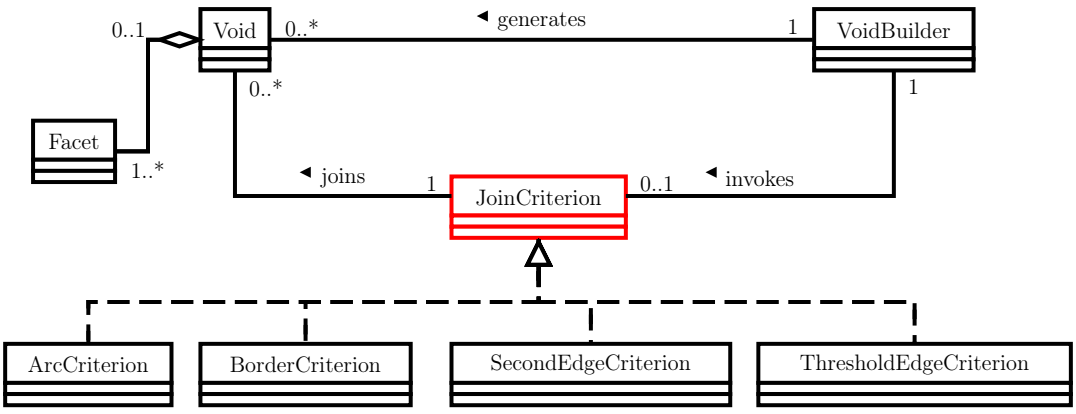


Figure 2.3: Class diagram for the 2-d DELFIN implementation.

A `VoidBuilder` class singleton loads the vertices and facet information into `Facet` objects. Then, it creates several `Void` objects which, by the end of the execution, contain facet objects

created earlier according to the definition of DELFIN. Finally, an instance of the selected joining criterion class is created, and is used by `VoidBuilder` to join `Void` objects.

Observe there is no actual need for a `JoinCriterion` interface as Python is a duck-typed language: it has been indicated here for convenience only.

2.3.2. Usage

Before execution, the installation of the *qhull* applications (specifically *qdelaunay*) is required. Some other libraries (*numpy*, *matplotlib*) are also required to run our code.

To execute the program from the main folder, run:

```
$ python2 src/delfin.py [-h] [-j JOIN] [-r]
                        [-e EDGEMIN] [-b AREAPRE] [-d AREAMIN]
                        rootname
```

where arguments enclosed by brackets are optional, and `rootname` corresponds to the point input file.

The point input file follows the format convention set by *qhull* and others. The first line must contain d, the number of dimensions (2 for this program). The second line must contain a single number n, the number of points in the file. Each of the following n lines represents the pair of coordinates of a point and must be expressed as a pair (x, y) of integers or floating point numbers separated by whitespace (scientific notation format, e.g. "6.02e23" is allowed).

Optional arguments are described in table 2.1. This same information can be accessed by running the program with the arguments `-h` or `--help`.

Table 2.1: Command line arguments for delfin.py.

Argument	Description
`-h, --help`	Shows the help message and ends the program
`-j JOIN, --join JOIN`	Joining method ID. `JOIN` must be replaced for one of these IDs: – 0: none – 1: Arc criterion – 2: Border criterion – 3: Second longest edge criterion – 4: Edge threshold criterion
`-r, --preprocess`	If set, removes points using d_3 criterion
`-e EDGEMIN, --edgeMin EDGEMIN`	Prevoids edge filter $e_{\min}$
`-b AREAPRE, --areaPre AREAPRE`	Prevoids area filter a_p
`-d AREAMIN, --areaMin AREAMIN`	Final voids area filter a_v

The program will run the algorithm over the point input file with the specified parameters, and will output its results in the 'results' folder. This output consists on 4 files, respectively

describing the Delaunay triangles, their corresponding neighbor data, the resulting voids' boundary, and their composing triangles. Additionally, a postscript plot is generated, depicting the point distribution, the void triangles, and the void centroids. The plot assumes a sample area between coordinates $-1\,000 \leq x, y \leq 1\,000$ (in distance units), so the plotting code must be adapted to correctly display samples with different boundaries.

2.4. Validation

The original DELFIN algorithm has already been validated. However, after several enhancements and new joining methods were implemented, a new validation was necessary to evaluate their contribution in void finding. The different joining methods we evaluated are explained in section 2.2.3. To compare the effectiveness of joining criteria, we ran our implementation over multiple artificially generated data sets.

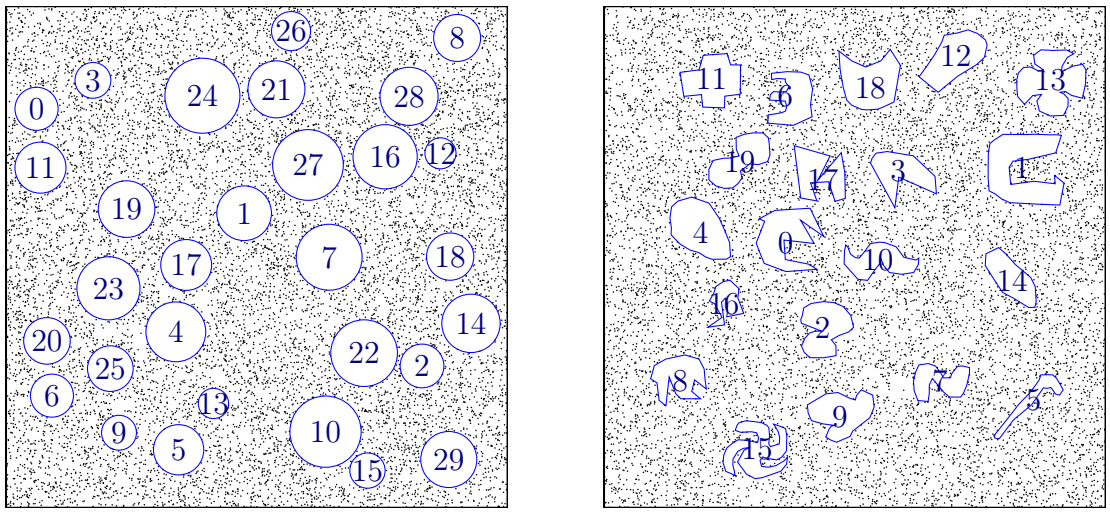
2.4.1. Datasets

We generated two void templates: a circular void template and an irregular void template. The circular void template contains 30 circles, with randomly generated centers according to an uniform distribution in an $N \times N$ square space (with $N = 2\,000$ Mpc$/h$), and their radii randomly generated by an uniform distribution over the range $[0.03N, 0.075N)$. The irregular void template contains 20 manually-drawn irregular polygons in a $N \times N$ square ($N = 2\,000$ Mpc$/h$), with their centers and areas also randomly generated in similar ranges.

From each template, a data set was generated by randomly choosing n points with an uniform distribution over the area comprehended by the template minus the generated voids. This process was repeated for each template, using the values of $n = 5\,000, 10\,000, 50\,000$, respectively generating background densities ρ_B equal to $1/800$, $1/400$ and $1/80$ particles per square Mpc$/h$. As a result, our generated voids are completely devoid of galaxies ($\delta = -1.0$), and their areas in the range of $[10\,000, 60\,000]$ (Mpc$/h)^2$. Figure 2.4 is a rendering of both templates with a void dataset ($\rho_B = 1/400$, $n = 10\,000$).

The values used to discard prevoids and voids too small in size (either by prevoid/void area with a_p/a_v, or by longest edge $e_{\min}$) were handpicked depending on the dataset size n in use. More precisely, these values were selected considering the average density, which depends on the area enclosed for analysis (constant in our experiments) and the number of points.

The specific values used are indicated in table 2.2 for each n.



(a) Circular voids

(b) Irregular voids

Figure 2.4: 2 000 Mpc/h × 2 000 Mpc/h artificial datasets with $n = 10\,000$ points. The void templates are displayed in blue.

n	a_p [(Mpc h^{-1})^2]	a_v [(Mpc h^{-1})^2]	$e_{\min}$ [Mpc h^{-1}]
5 000	6 000	12 000	80
10 000	3 000	8 000	65
50 000	600	4 000	35

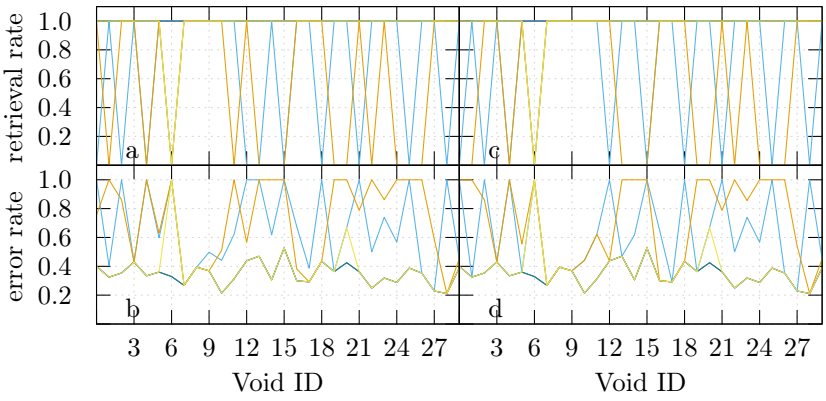
Table 2.2: Filters used for each dataset size.

2.4.2. Results

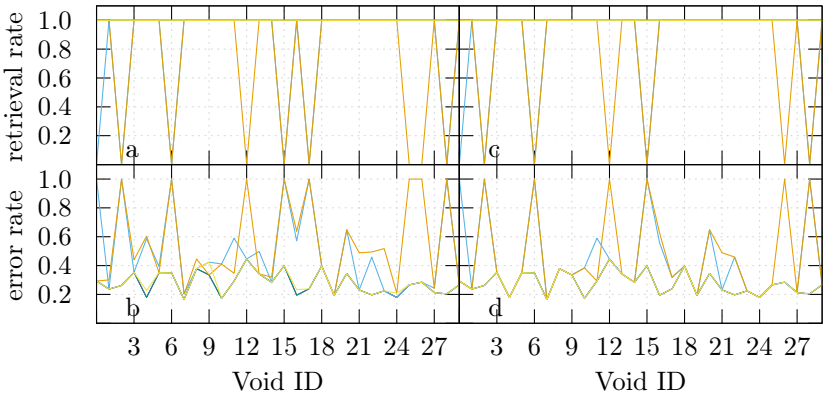
To measure the performance of each algorithm, we used two parameters r_v, the retrieval rate for void v, and e_v, the overdetection rate for void v. These quantities are respectively calculated as the detected fraction of the original void, and the non-void fraction of the detected void. More precisely, we define the retrieval rate of a void as $r_v = \frac{A_{\cap v}}{A_v}$ where $A_{\cap v}$ is the intersection between the generated void v and the retrieved void v^*, and A_v is the area of v. Similarly, we define the overdetection or error rate of a void as $e_v = 1 - \frac{A_{\cap v}}{A_{v^*}}$ where A_{v^*} is the area of the retrieved void v^*. Note that since we generate the datasets by generating points at random, the actual voids will be larger than the ones we generated. This area increase can be quantified approximately as $\Delta a = C\sqrt{\frac{a_0}{\rho_B}}$, where a_0 is the area of the generated void, ρ_B is the background density, and C is a positive constant which depends on the void shape.

For each generated void v_i, we assign to it the closest retrieved void from the output of a joining method, and calculate r_v and e_v for each generated void. The results are represented in figures 2.5 and 2.6.

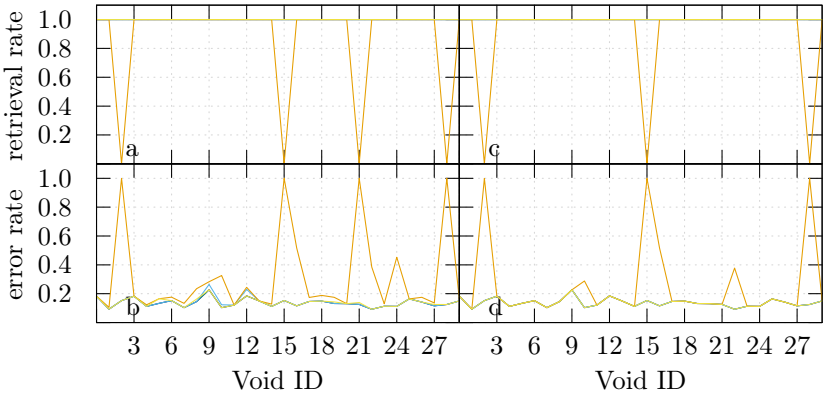
We also retrieve the average number of void fragments associated with each generated void as an additional metric of the effectivity of each joining criteria. We only consider detected



(a) Results for $n = 5\,000$ points



(b) Results for $n = 10\,000$ points



(c) Results for $n = 50\,000$ points

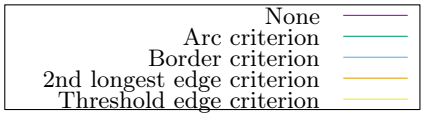
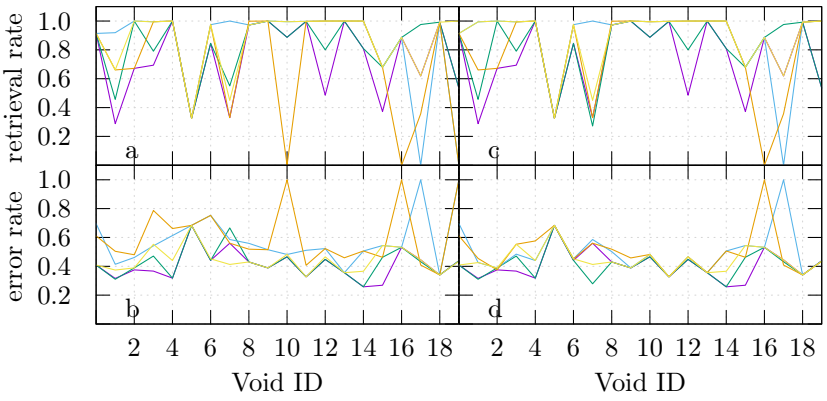
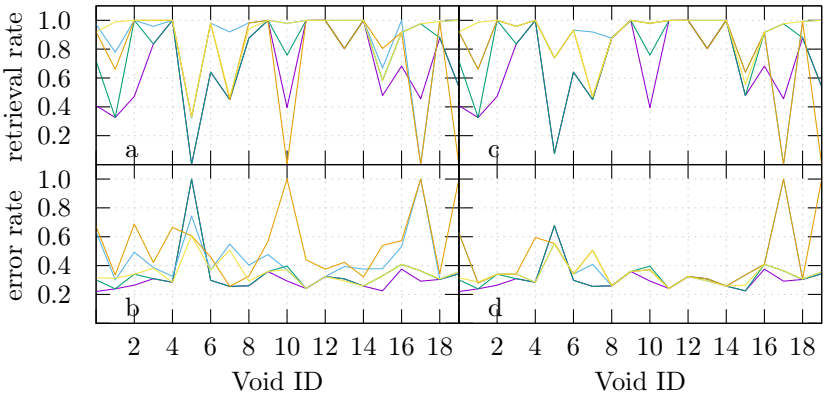


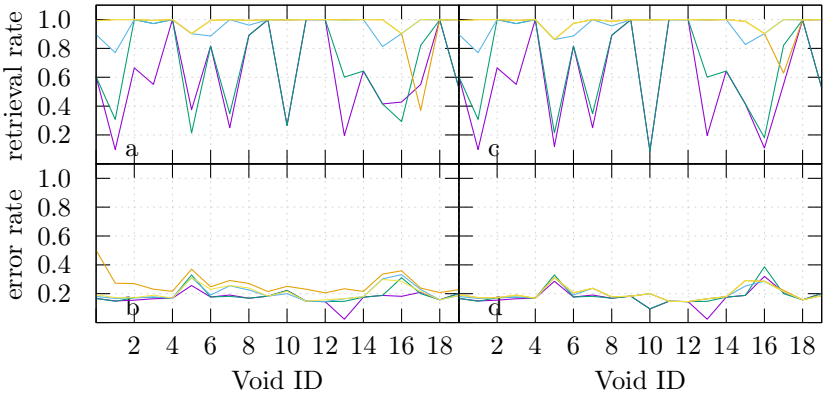
Figure 2.5: Retrieval and error rates for circular voids. Subfigures **a** and **b** indicate rates for the area filter, while subfigures **c** and **d** indicate results for the longest edge filter.



(a) Results for $n = 5\,000$ points



(b) Results for $n = 10\,000$ points



(c) Results for $n = 50\,000$ points

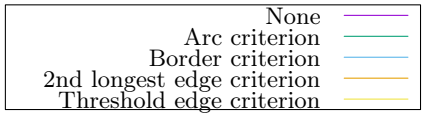


Figure 2.6: Retrieval and error rates for irregular voids. Subfigures **a** and **b** indicate rates for the area filter, while subfigures **c** and **d** indicate results for the longest edge filter.

voids to avoid underestimating the number of fragments. The average for irregular voids with $n = 10\,000$ and $n = 50\,000$ is displayed in table 2.3.

		Joining Criterion				
		None	Arc	Border	2nd longest edge	Threshold edge
Prevoid Filter	Area (a_p, a_v)	1.84	1.58	1.55	1.50	1.10
	Edge ($e_{\min}$)	2.50	1.90	1.10	1.20	1.15

(a) Fragmentation for $n = 10\,000$.

		Joining Criterion				
		None	Arc	Border	2nd longest edge	Threshold edge
Prevoid Filter	Area (a_p, a_v)	1.95	1.85	1.15	1.00	1.00
	Edge ($e_{\min}$)	3.90	3.00	1.45	1.05	1.00

(b) Fragmentation for $n = 50\,000$.

Table 2.3: Average number of void fragments detected by each criteria combination. A value equal to 1.0 indicates zero fragmentation.

2.4.3. Analysis

We would like to remark that actual voids are larger than the boundaries we defined for them. Not suprisingly, smaller values of n increase the error rate because of this effect and because at low densities, the voids' widths in some sections become comparable to the average distance between galaxies. In fact, smaller voids in a random distribution will appear following a Poisson distribution as a function of area[6] but they will be discarded by the area/longest-edge filter. However, they would pass unnoticed through it if one of them is next to the edge of our voids, as the algorithm might join them (even without using a void joiner criterion). In those cases, it would be very hard to retrieve any shapes as the density information has been lost as a result of discretization.

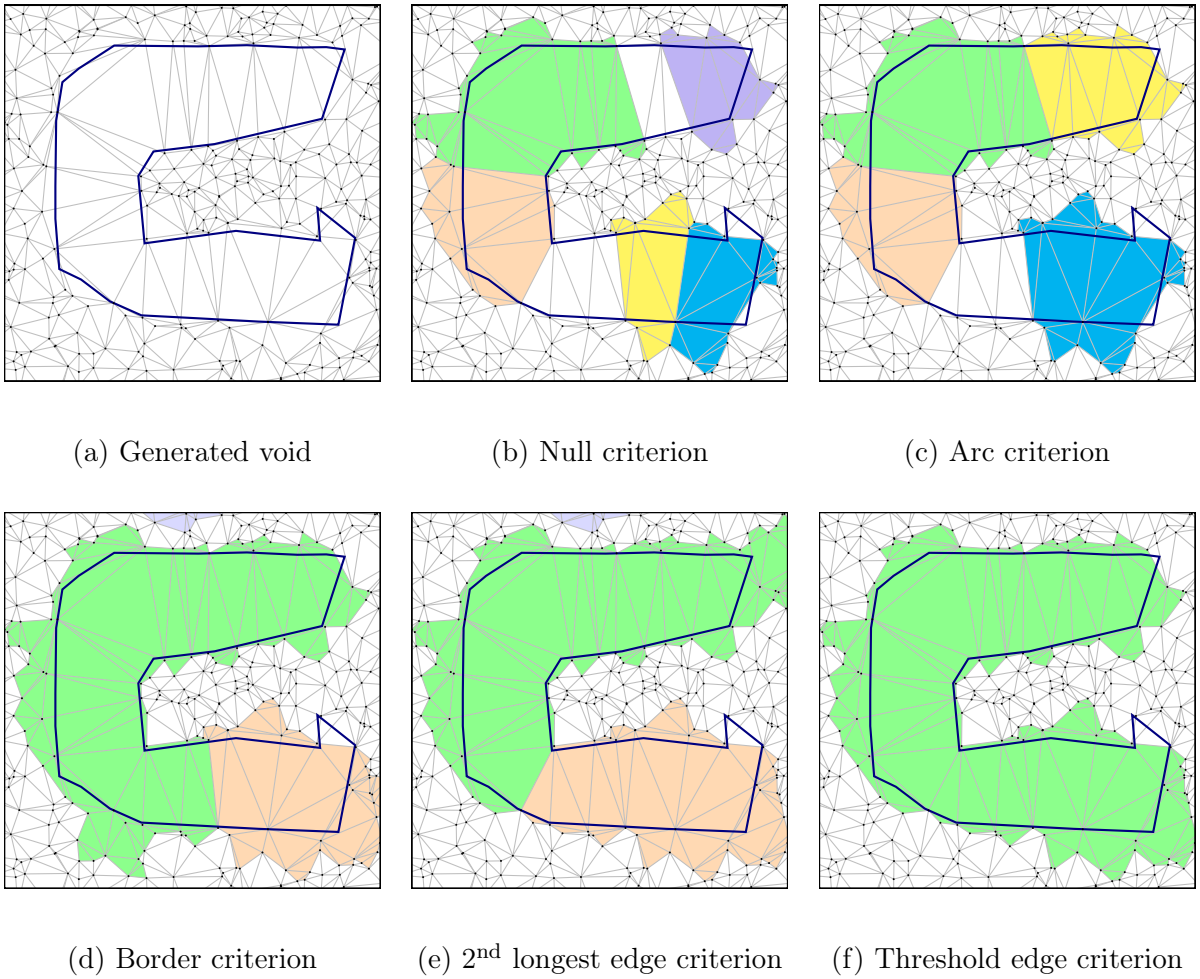
Barring the previous situation, circular voids were correctly detected. This is expected since DELFIN was originally capable of detecting circular voids effectively, and thus our joining algorithms were not required in this case. The picture, however, is different in the case of irregular voids. With these datasets, the peculiarities of each void joining criterion come to light.

The arc criterion, while effective with some cases where a single void is detected as two fragments (such as the one described in the original publication of the algorithm[8]), still fails to connect arbitrarily shaped figures, and the order in which fragments are joined heavily affect the results. The border criterion had the void shapes in consideration, and thus was more precise at joining void fragments. However, it shares some of the main weaknesses with the arc algorithm, such as its inability to join together voids with an elongated shape and its heavy dependence on the order in which fragments are joined. In the case of circular voids, it tended to join two into one if they were too close, something expected from the criterion's definition, but undesirable nonetheless.

Our other two criteria, the second longest edge criterion and the threshold edge criterion

had a higher retrieval rate, effectively identifying most details of our generated voids. While we had reservations on whether the second longest edge criterion would run erratically or not, it correctly joined a significant portion of fragmented voids. However, it is highly dependent on shape, and was unable to join fragments on curved *tube-shaped* voids, where more equilateral triangles are formed (see figure 2.7e), and bifurcations, where one of the three possible paths would necessarily be left out. Additionally, when discarding prevoids by area, it generated some fake voids as a result of joining smaller segments of average density into a structure large enough to pass the selected area threshold.

Finally, the threshold edge criterion did meet our expectations for an effective joining criterion. Its main weakness would be its dependence on the scale of data. While we did not have any case to evaluate this condition, if two very large voids intersect, this criterion might join them, even though their intersection is not significant compared to their size. A new edge threshold depending on the actual voids being joined might be a workaround to this issue.



(a) Generated void (b) Null criterion (c) Arc criterion

(d) Border criterion (e) 2nd longest edge criterion (f) Threshold edge criterion

Figure 2.7: C-shaped void #1 (in blue, thick) with detected fragments (filled in several colors) using each criterion over the $n = 10\,000$ point set.

Regarding the filtering used for prevoids and voids, using a threshold for the longest edge of each prevoid did reduce false voids, especially in the lower background density data sets.

Its usage is a suitable replacement for the area threshold, lowering the risk of removing central prevoids and effectively filtering false voids.

2.4.4. Performance

We ran our DELFIN implementations over random generated sets of n points in an $N \times N$ square ($N = 2000$ Mpc/h), using the irregular void template. The experiment was repeated for values of $n = 2^{10}, 2^{11}, \ldots, 2^{19}$ running the original algorithm and the optimized version without any joining method. Only the actual running time of the algorithm was considered (file I/O operations, parsing and plot generation times were ignored).

The system used was a dual core Intel Core i3 3110M processor and 4 GB of memory running an up to date Arch Linux distribution. We stopped running the original implementation just before a single run would take more than a day, and the new algorithm once our system started using the swap partition after the system depleted its physical memory. Our results can be seen in figure 2.8.

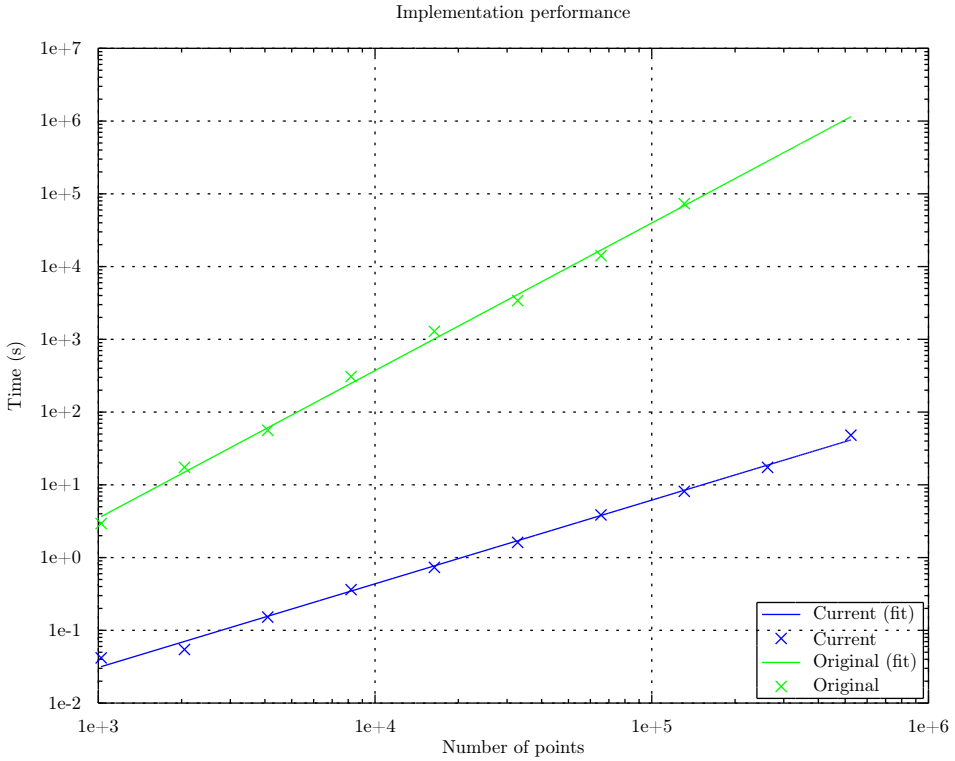


Figure 2.8: DELFIN performance over data sets with $n = 2^{10}, 2^{11}, \ldots, 2^{19}$ points.

Our running time is several orders of magnitude less than the original execution time while preserving exactly the same results. In fact, the original implementation time seems to follow a cuadratic curve, while our new implementation seems to adjust to a linearithmic ($O(n \ln n)$) curve. This difference between linear and cuadratic is reflected by the slopes of curves in the log-log graph (the original implementation's slope is approximately twice of that of our new implementation).

A further performance increase may be achieved by building voids without sorting them by the length of their longest edge. However, improvements in this case will not be as drastic and we would have to sacrifice flexibility as some criteria depending on the facet order or in the top-down construction process would no longer work. For example, pruning voids depending on volume percentage, whose code is currently part of the project (although disabled for users) will have undefined behaviour if voids are not constructed from least dense regions. For this reason, we have chosen to continue sorting voids in our implementation.

Chapter 3

The 3-d Extension of DELFIN Algorithm

On the previous chapter, we have analyzed four void joining methods and concluded that the edge threshold method provided the least shape-dependent results. This last method has been included in the 3-d version of our **DEL**aunay Edge Void **FIN**der algorithm. In this chapter we will overview the 3-d version of DELFIN, evaluate its behavior over different density contrasts and compare it with Foster & Nelson's implementation of their algorithm[7].

3.1. The Algorithm

The algorithm is similar to the 2-d version, and features the same extensions used for the 2-d version of DELFIN. However, some modifications are due since 3-d edges are no longer limited to a pair of simplices, but an undetermined number of them.

First, the Delaunay tessellation is obtained for the sample set, and the resulting tetrahedra are sorted by their longest edge. We considered evaluating tetrahedra according to other features, such as their volume or largest face area, but elongated, icicle-shaped tetrahedra would be underrepresented by these parameters.

Then, locally longest edges are chosen as a starting point for a new void, starting from the longest. A locally longest edge is defined as the longest edge for all tetrahedra containing it. All tetrahedra sharing this edge are grouped into what is the base of our prevoids.

In an iterative process, the outer (unvisited) edges of new tetrahedra are visited, and tetrahedra which both do not belong to another prevoid and which have any of these edges as their longest edge are added to the prevoid (figure 3.1). Note that the new tetrahedron may share no faces with other tetrahedra from the void being processed. The previous iteration stops when no tetrahedra satisfying the previous conditions are found. These steps are repeated for the following locally longest edges.

Note that while the 2-d algorithm allowed to build voids without sorting triangles by longest edge, that is no longer possible in 3-d with tetrahedra if we want to build a robust

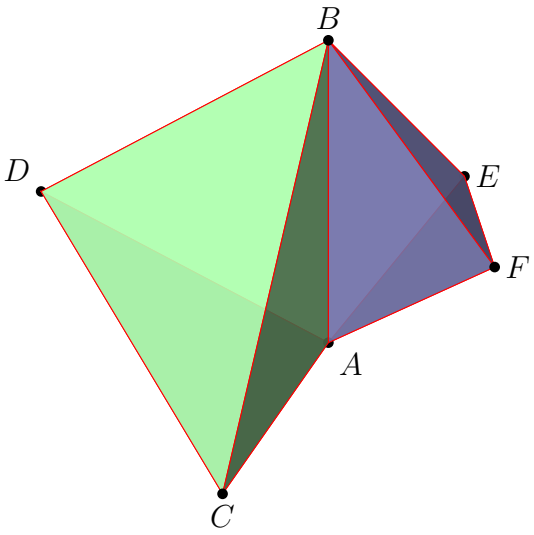


Figure 3.1: Edge-adjacent tetrahedra $ABCD$ and $ABEF$. Since AB is the longest edge of $ABEF$, it will be joined to the first void reaching this edge, which could or could not be the void containing tetrahedron $ABCD$.

algorithm. In fact, if we try to join each tetrahedron to the tetrahedron with which it shares its longest edge, we will notice this edge is shared with at least two more tetrahedra. So, which of these should it be joined to? The answer would be the tetrahedra belonging to the prevoid spawned from the longest edge; however, knowing this requires sorting at least the longest local edges, which is needed for our algorithm to work and takes time $O(n \log n)$.

From the resulting regions, we considered 3-d extensions for the two methods from the 2-d algorithm to choose which regions would considered a prevoid. Recalling from section 2.1 that if a selecting criterion is not used, the resulting regions will tessellate the convex hull of the set of points; this property holds for the 3-d implementation as well. The first method consists on filtering voids whose volume V is below a threshold $V_{\min}$, while the second method discards voids whose longest edge length l is lower than a threshold $l_{\min}$. Although both could be used at the same time, we will analyze them separately during validation.

Once prevoids have been filtered by one of the methods described above, the final step consists in prevoid joining. After evaluating the joining criteria from section 2.2.3, we have chosen to include the threshold edge criterion to 3-d DELFIN using the same equation 2.1 from the 2-d version.

See algorithm 3 for a summarized version of the algorithm.

3.2. Implementation

By the beginning of this thesis work, a non validated 3-d prototype had already been developed. However, it had not been validated, and did not consider the modifications we have made since. We continued from this prototype to obtain a validated and efficient

Algorithm 3: 3-d DELFIN Algorithm

```
read the Delaunay tessellation;
read threshold_volume;
void_list = ∅;
order tetrahedra by longest-edge;
repeat
    if longest edge of next unvisited tetrahedra is shorter than threshold_length then
        break;
    end
    if longest edge of next unvisited tetrahedra is the longest for all sharing tetrahedra
     then
        get all tetrahedra t_1, ..., t_n sharing the longest-edge;
        tetrahedra_set = [t_1, ..., t_n];
        label t_1, ..., t_n as used;
        foreach tetrahedron t neighbor of tetrahedra_set do
            if t shares its longest-edge with a tetrahedron of tetrahedra_set then
                tetrahedra_set = tetrahedra_set ∪ t;
                label t as used;
            end
        end
        if volume of tetrahedra in tetrahedra_set ≥ threshold_volume then
            void_list = void_list ∪ polygon(tetrahedra_set);
        end
    end
until there is no tetrahedron left to process;
foreach edge edge in edge_set do
    if length of edge is greater than threshold_length then
        join subset of prevoids from void_list sharing edge;
    end
end
remove voids from void_list with an edge from the boundary;
```

implementation of the extended DELFIN algorithm.

The implementation has been written in C++ using *qdelaunay*[3] externally to obtain the 3-d Delaunay tessellation of a set of points, as we did for the 2-d implementation. The features C++ provides also allow for greater control over memory usage and more efficient code than Python, although somewhat more verbose. *Qdelaunay* uses the quickhull algorithm on the dataset projection over a paraboloid, and later discards the extra dimension, keeping only the resulting neighbor information and edges.

The algorithm runs in time $O(t \log t)$ in the number of tetrahedra t: in practice and for efficiency reasons, we sort edges e by length instead. It is known that in every 3-d Delaunay tessellation, $3 + 3t \geq e$. Such a claim may not be made for vertices: in the worst case, t is $O(n^2)$. However, Delaunay tessellations in practice have a linear number of tetrahedra. Once

sorted, each edge is visited once by evaluating them from longest to shortest (e traversals), and at most once for every tetrahedra containing it ($6t$ traversals, as each tetrahedron may visit its 6 edges). Each tetrahedron can be visited once for each edge it contains, so it will be visited at most 6 times ($6t$ traversals). All of the previous neighbor index information is stored in C++ vectors, but the actual elements are stored in a C++ map, so access time is $O(1) + O(\log t)$. Since elements are accessed a linear number of times, we also achieve $O(t \log t)$ for these operations. Prevoid joining, on the other hand, is done using a Union-Find disjoint set structure, with an amortized time of $O(v\alpha(v))$ over the number of prevoids, where $\alpha(n)$ is the inverse of the function $f(n) = A(n, n)$ and A is the Ackermann function. Thus, it can be ignored for complexity analysis.

3.2.1. Structure

Figure 3.2 shows the class diagram for our 3-d implementation of DELFIN.

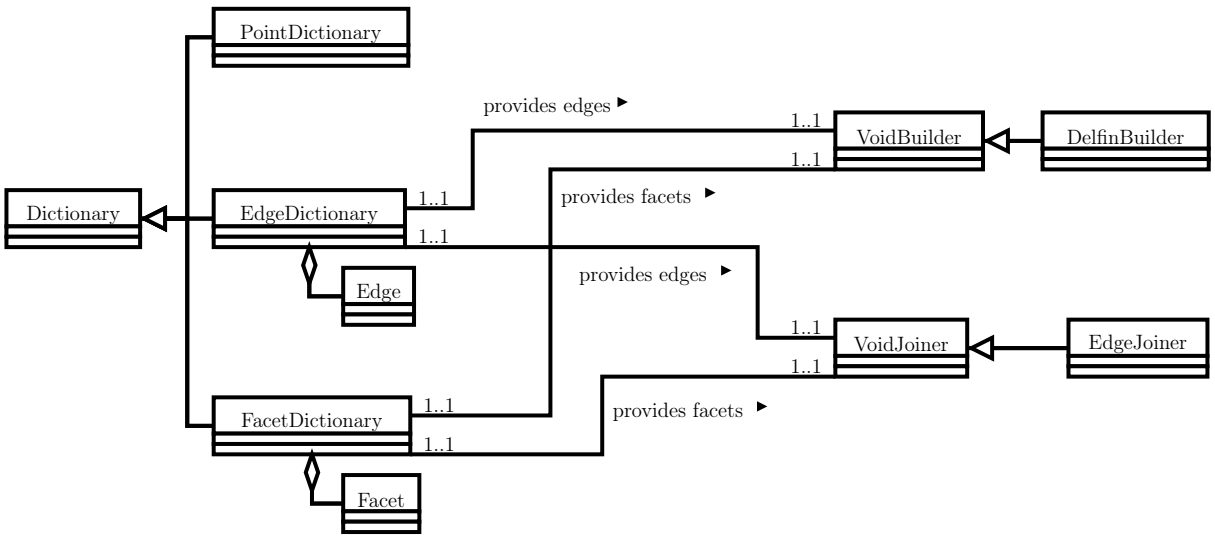


Figure 3.2: Class diagram for the 3-d DELFIN implementation.

The main program constructs dictionaries of points, edges and facets (tetrahedra), which are then given as parameters to a `VoidBuilder`. We have implemented a single non-abstract `VoidBuilder`, class `DelfinBuilder`. This class builds the prevoids and filters them according to either volume or longest edge. Then, one or several `VoidJoiner`s may receive a set of prevoids and join them according to a certain criterion. As of now, our only method to join voids is the threshold edge, as other methods proved insufficient during the 2-d evaluation because of characteristics which extend to 3-d. Our implementation of the threshold edge joiner is contained in the class `EdgeJoiner`.

3.3. Validation

We evaluated DELFIN by analyzing its performance over several data sets, namely, several toy distributions with multiple spherical and cubic voids, and a fraction of the SDSS DR5 survey data.

3.3.1. Artificial data evaluation

To evaluate our algorithm's behavior over voids regarding their density, we generated multiple 3-d data sets, this time producing non-empty voids. To characterize them, we will use a density contrast parameter $\hat{\delta} = \bar{\rho}/\rho$ instead of the more common overdensity parameter $\delta = 1/\hat{\delta} - 1$.

First, we evaluated DELFIN over spherical underdensities. We generated a $m \times m \times m$ cubic volume ($m = 100h^{-1}$ Mpc) filled with $q = 20$ randomly placed spheres of radius $r = 25h^{-1}$ Mpc, only restricting them to avoid intersections between spheres and to keep them from intersecting the cubic boundary. Then, the background volume, that is, the volume inside the cube excepting that occupied by the q spheres, was filled using a random generator in such a way that the density on this background would be $\bar{\rho} = 4/1000$ particles per cubic Mpch^{-1}, and then filled the spheres randomly until their average density would be ρ. We chose different values for ρ with the intent of evaluating the algorithm over different values of $\hat{\delta}$.

Since voids in practice are not spherical, we also evaluated DELFIN over cubic underdensities. The setup was similar to that of spherical voids: a $m \times m \times m$ cubic volume ($m = 100h^{-1}$ Mpc) filled with $q = 20$ randomly placed and oriented cubes of edge length $s = 30h^{-1}$ Mpc. We constrained the positions in such a way that no cube would intersect either the cubic boundary nor any of the other s-sized cubes. We used the same set of densities ρ and $\hat{\rho}$ for the s-sized cubes and for the background, respectively.

For each of these datasets, we computed both the retrieval and overdetection rates, r_v and e_v, whose definition is an extension of that used in 2-d but evaluating volumes instead of areas. In this way, we measure the effect of density contrast over the effectiveness of our algorithm.

Results

The resulting voids found by our algorithm were non-convex, roundish in shape polyhedra. As the density contrast $\hat{\delta}$ increased, the same effect seen for planar voids was seen here: the actual voids were larger than the theoretical voids we used to generate our data sets. The results obtained over the spherical and cubic voids are shown in figures 3.3 and 3.4.

Our void joining phase effectively joined together the different fragments or prevoids corresponding to the same sphere or cube. This is a marked improvement over our original

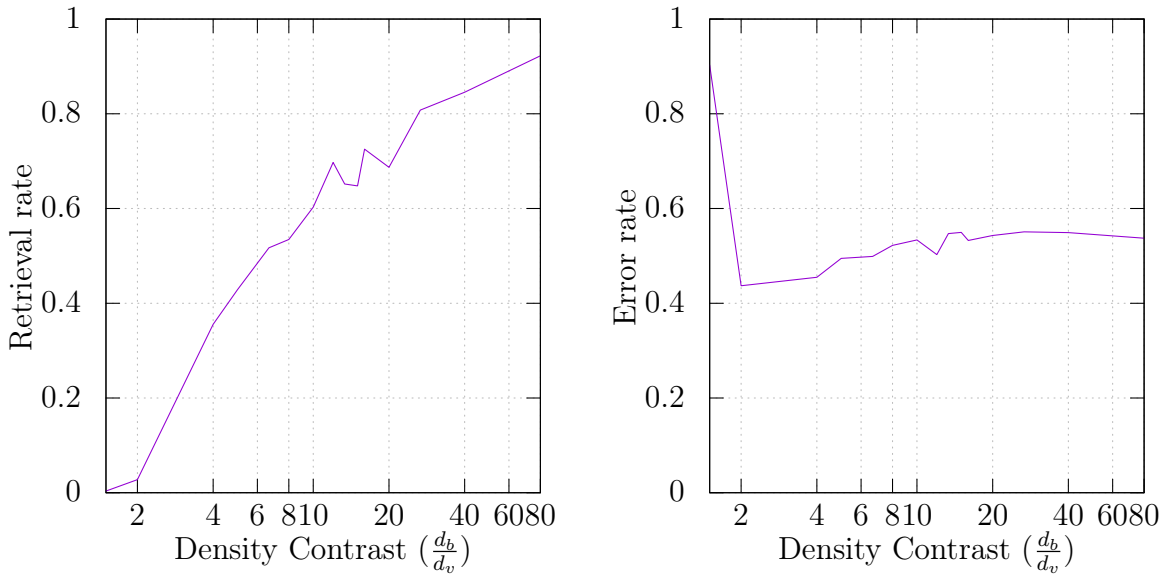


Figure 3.3: Retrieval and error rates for spherical underdensities

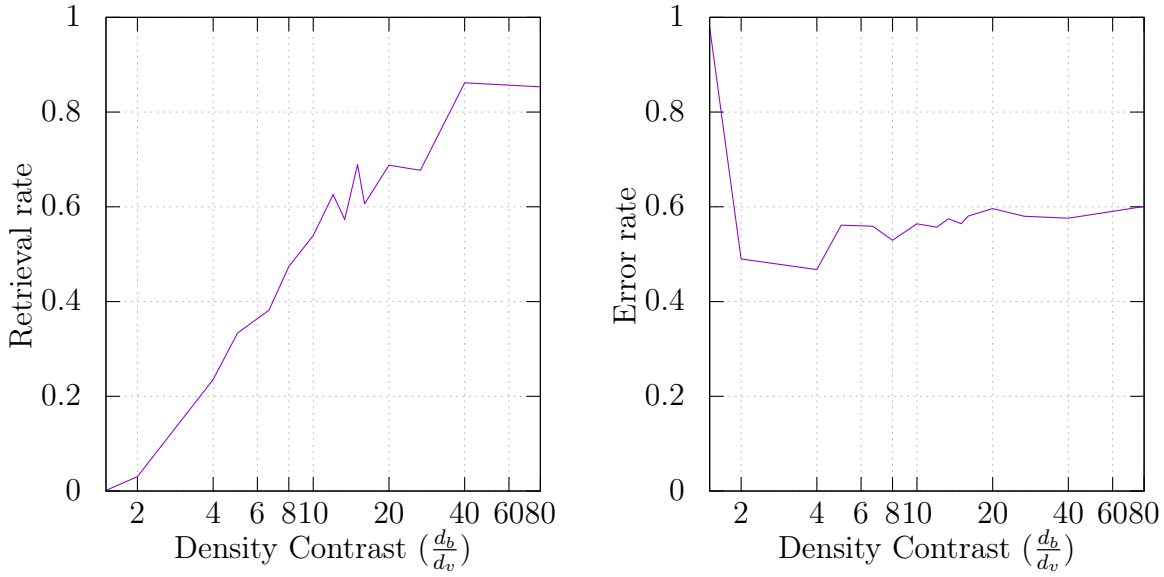


Figure 3.4: Retrieval and error rates for cubic underdensities

algorithm. Recall from our original algorithm that whenever an irregular void or a void with a point in its interior was scanned with the original algorithm (i.e. without the void joining phase), the void would split into two or more voids. This is no longer the case with high density contrast rates, where we effectively find voids almost completely.

Naturally, as we dropped the density contrast $\hat{\delta}$, the algorithm had increasing difficulty identifying voids. More specifically, as $\hat{\delta}$ dropped below 10, our retrieval rates were less than 60%. A careful look into these definitions reveals that such density contrasts produce, on average, Delaunay edges less than $\sqrt[3]{10}$ longer than those on the background, and spurious voids of comparable size to our fragments might appear in the background. At this point, our algorithm is quite conservative to avoid detecting spurious voids resulting from the random

distribution of the points we have generated.

However, we must consider that real voids not only possess a lower density than the average, but also reach their lowest density at their center, and then this density increases as one moves towards its limits. As our algorithm builds voids from least dense regions towards strictly increasing regions, is more suited to work over such a void profile, and thus should produce better results over such real world voids.

3.3.2. Catalog data evaluation

Although we can prepare artificial voids and compare them with results from DELFIN, this is not possible with real datasets. Since there is no consensus on a void definition, there is no canonical void catalog to compare with, and thus no simple way to evaluate performance with such data. However, since previous works on voids are based on those found by currently existing algorithms, a reasonable albeit subjective method to review an algorithm is evaluate its results compared to those from an existing void finder.

Foster and Nelson's algorithm[7] is the most recent revision of the base ideas published by El-Ad & Piran[6]. Since their voids are built from maximal spheres with a minimum radius, which implicitly share many properties with Delaunay tessellations (read the final paragraph from section 1.2 for more details), it is a natural choice to compare our algorithm with theirs. They (F&N) have run their algorithm over a subset of SDSS Data Release 5[1], and the catalog of the voids they found has been made public. Namely, their subset consists of galaxy redshifts between a right ascension of 140°and 320°, a declination between 30°and 65°, absolute magnitude between -22.3 and -20.8, and redshift up to 0.16.

Since any voids touching the sample boundary are discarded by our algorithm, voids from F&N's catalog touching and/or crossing the boundary[1], will not have a match in our results. Thus, it is reasonable to discard F&N's boundary voids as well for a fair comparison with our algorithm. Thus, we have discarded all voids from F&N's catalog crossing either the right ascension boundaries ($140° \leq \alpha \leq 320°$) or the declension ones ($30° \leq \alpha \leq 65°$). We have also discarded those voids crossing the $z = 0.16$ limit, as the background density in SDSS DR5 drops considerably beyond this point as a result of extinction. This resulted in a significant reduction of voids: from the 232 original voids, only 123 were left.

We ran our algorithm over the filtered SDSS DR5 subset, i.e, after removing field galaxies, and obtained a resulting void catalog (figure 3.5). Then, we compared our catalog with F&N's before and after discarding boundary voids.

The comparison is not trivial, though. Our voids are non convex polyhedra while F&N's voids are unions of spheres of different sizes. While possible, calculation of the exact intersection and union volumes requires solving two difficult problems: union of spheres and intersection of these unions with general polyhedra. The first problem is analytically simple in the case of two spheres, but becomes increasingly complex when three or more spheres have a non-zero intersection. The second problem would require either **(a)** partitioning space

[1]Some of these voids are explicitly allowed by them as they extend the boundary by 5-10 Mpc/h.

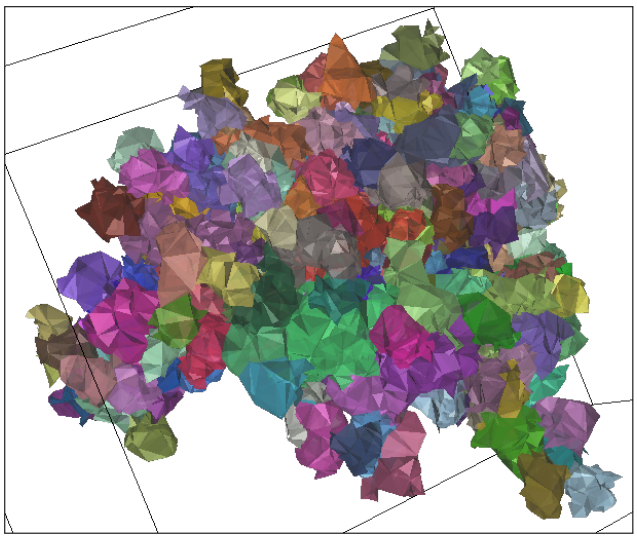


Figure 3.5: 3-d visualization of voids found using DELFIN with the volume filter over a SDSS DR5 subset.

according to spheres and tetrahedra faces, and then summing the volumes of those regions which are part of either void (or both of them in the case of intersection), or **(b)** obtaining the surface of both voids, cutting them in their intersection curve, and integrating the volume under the correct pieces. Both methods require 3-d integration of planar and spherical surfaces over a complicated 2-d shape, with both elliptical and straight sections.

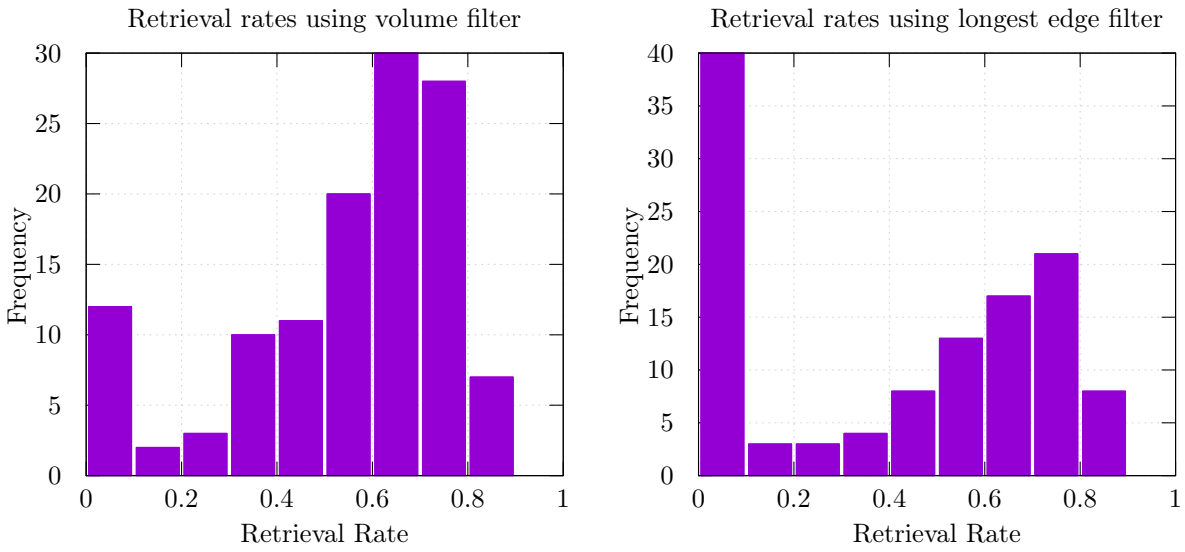
The previous methods are beyond the scope of this work, and instead, we have used Montecarlo methods to estimate these intersections. First, for each pair of voids, we obtained a bounding box for each of them and checked for intersection. Those voids might intersect only if the boxes intersect. For each box intersection, we made a bounding box for both voids. and used $k = 10\,000$ randomly placed particles to estimate the intersection. A point is inside a F&N's void if it is contained in any if the conforming spheres, and it is inside one of our voids if it is inside any of its tetrahedra.

Results

Using the volume and longest edge filters, respectively, we found 242 and 255 polyhedral voids.

Considering F&N's voids as the real voids, we respectively obtained average retrieval and overdetection rates equal to $r_v = 0.55$ and $e_v = 0.32$ by using the volume filter, and $r_v = 0.39$ and $e_v = 0.55$ by using the longest edge filter. We evaluated the distribution of r_v and e_v per each void. Figures 3.6 and 3.7 respectively correspond to histograms with the distribution of retrieval and overdetection rates, one for each filter, allowing several of our voids to match a single F&N.

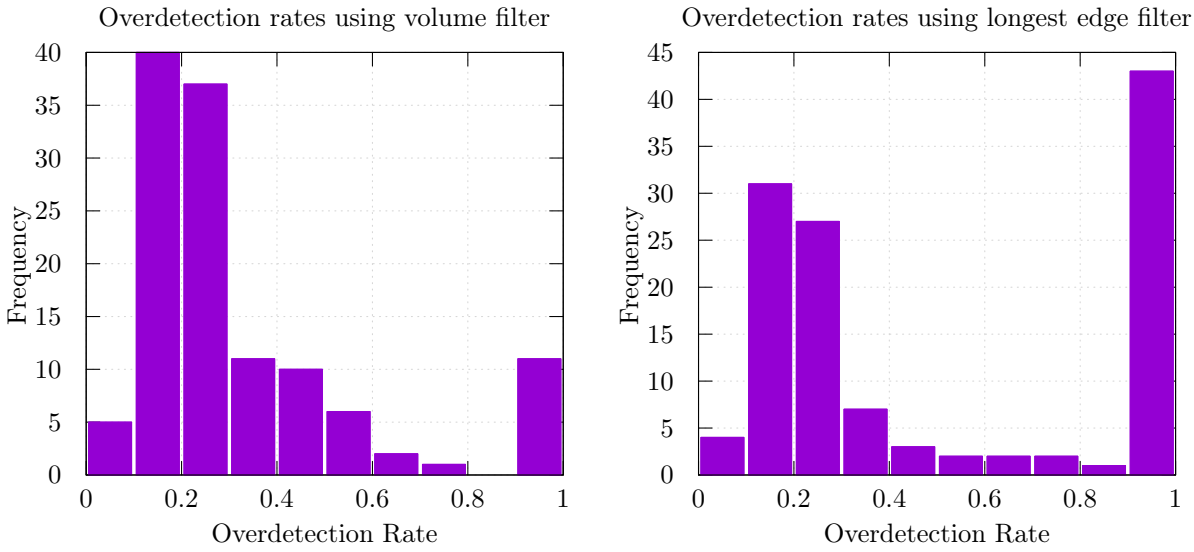
We evaluated these voids for a second time, now allowing only a one-to-one void matching between our voids and each F&N void; we show our retrieval rate in figure 3.8, and the



(a) Volume filter

(b) Longest edge filter

Figure 3.6: Retrieval rates considering multi-matching.



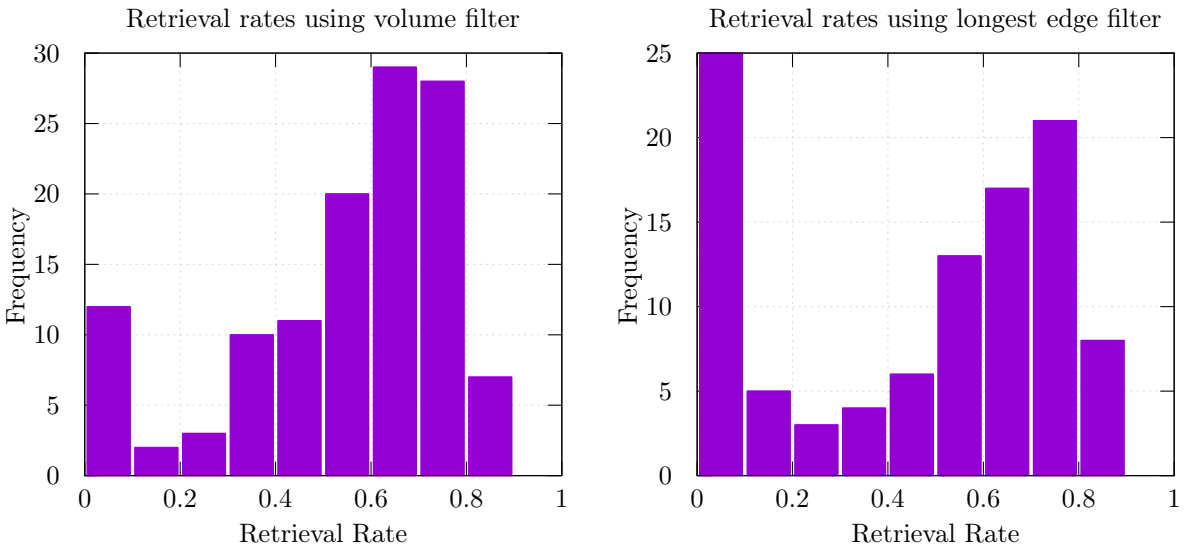
(a) Volume filter

(b) Longest edge filter

Figure 3.7: Overdetection rates considering multi-matching.

overdetection rate in figure 3.9. The retrieval and overdetection rate averages in this situation are $r_v = 0.54$ and $e_v = 0.32$ for the volume filter, and $r_v = 0.38$ and $e_v = 0.50$ for the longest edge filter.

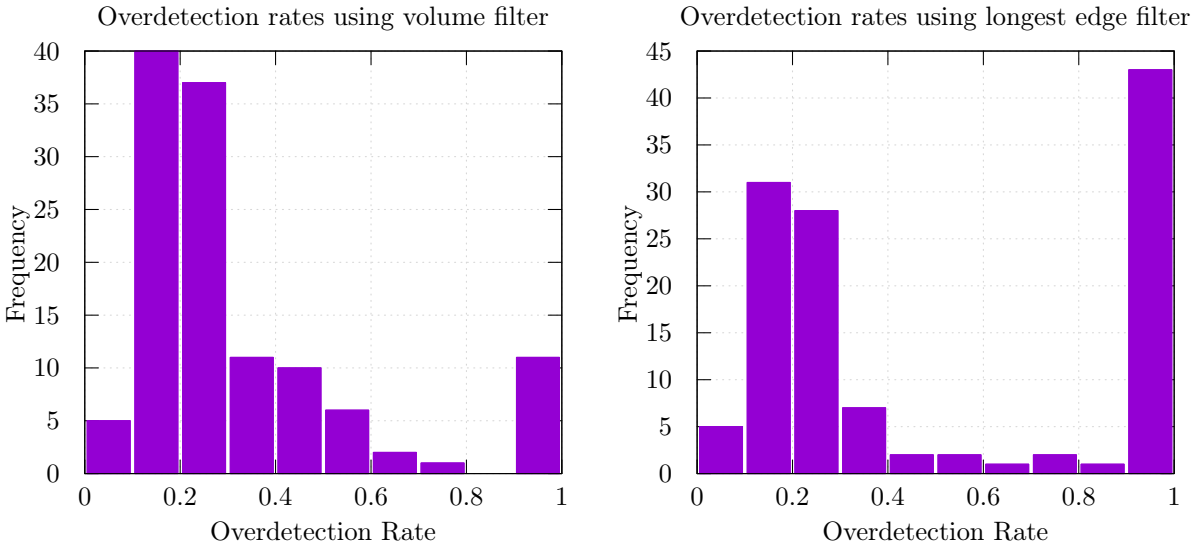
If we consider a retrieval rate of 50% or greater as acceptable (using the 1-to-1 matching), then we have found 84 (68%) acceptable F&N voids using the volume filter, while by using the longest edge filter, we have found 59 (48%).



(a) Volume filter

(b) Longest edge filter

Figure 3.8: Retrieval rates considering 1-to-1 matching method.



(a) Volume filter

(b) Longest edge filter

Figure 3.9: Overdetection rates considering 1-to-1 matching method.

A high retrieval rate 1-to-1 match (using the volume filter) is shown in figure 3.10. Our algorithm adds some tetrahedra whose circumsphere is either smaller than F&N's minimum allowed sphere or is simply not added to the void. In the case our tetrahedra's circumspheres would correspond exactly with the spheres from F&N's void, then it would be completely contained into those spheres (because each tetrahedron would be completely contained in its own circumcircle), and our void would not be visible.

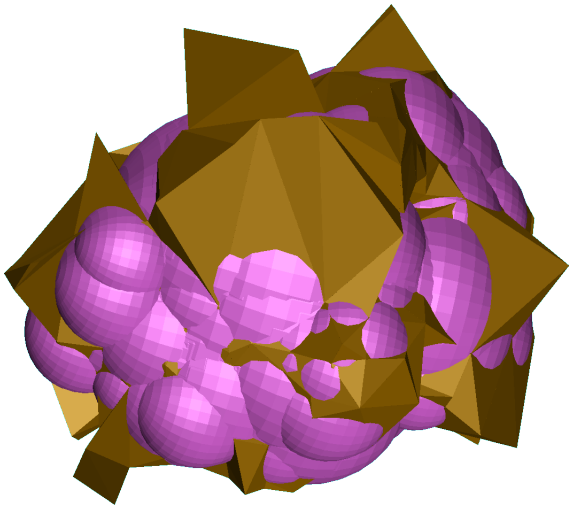


Figure 3.10: Close match of a F&N void (purple) and a DELFIN void (brown) using the volume filter, with $r_v = 0.865$ and $e_v = 0.133$.

The median match, that is, the match for the F&N void whose retrieval rate is the median, is shown in figure 3.11. It shows a void we have detected which resembles relatively closely one of Foster's voids.

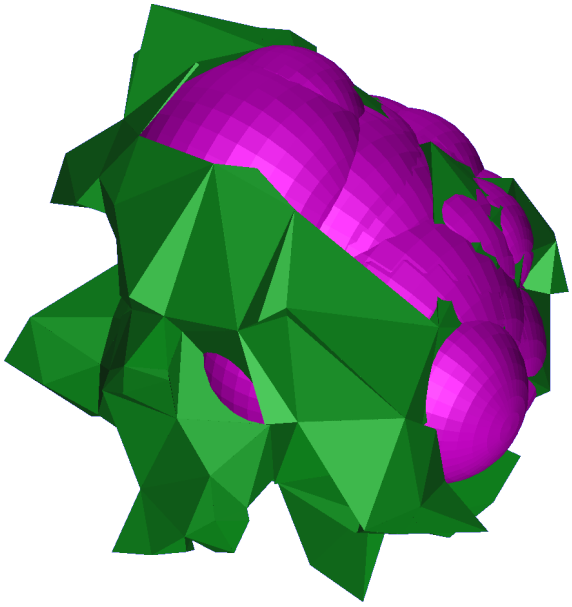


Figure 3.11: Median match between F&N voids and DELFIN voids using the volume filter, with $r_v = 0.610$ and $e_v = 0.380$.

A mismatch is shown in figure 3.12. This kind of result occurs whenever our prevoids and F&N holes are constructed in different ways, as a result of the different rules we have chosen to obtain a void. In this case, the circumspheres (holes) for our larger tetrahedra were discarded either by F&N's void building criterion or our border criterion (in case the F&N void expanded beyond the boundaries of the data sample). As mentioned in the introduction in page 1, there is no consensus in what exactly is a void, so these differences in voids' expansion (and results) are expected and will occur as long as our results are not exactly the

same, which is not the goal of our algorithm.

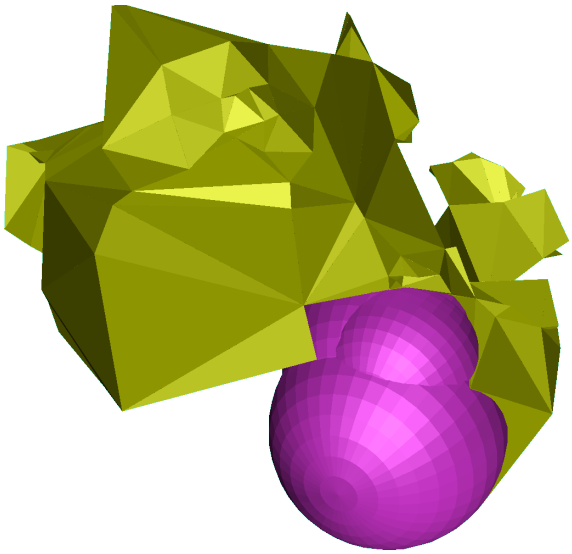


Figure 3.12: Poor match of a F&N void and a DELFIN void ($r_v = 0.242$, $e_v = 0.909$), respectively in magenta and yellow.

A split match is shown in figure 3.13. Occasionally, our prevoid joining method will reject a join as their shared edges are too short, whereas F&N would join their spheres because of the different methods used to expand voids as they are found. In these cases, the density on the shared border between prevoids is high enough for our algorithm to keep voids apart; it is an expected result from the threshold edge criterion we have used, and it may be adjusted to allow a different quantity of void joinings.

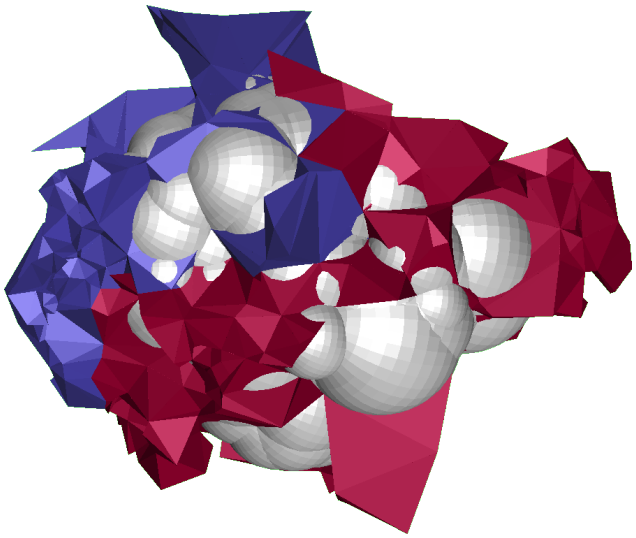


Figure 3.13: Split match of a F&N void (white) and two DELFIN voids (maroon and blue). Together, they achieve a retrieval rate of $r_v = 0.824$ for the white F&N void.

3.3.3. Analysis and Discussion

We like to see our algorithm as a bridge between maximal sphere algorithms and watershed-based algorithms. Our algorithm takes as input a Delaunay tessellation, which corresponds to the dual of the Voronoi diagram, over which most watershed-based algorithms build their voids. Yet, the Delaunay tessellation, over which our algorithm is based, is directly related to the maximal spheres since, as we saw in section 1.2.1, such spheres correspond to the circumspheres of the Delaunay tetrahedra.

Considering we are comparing polyhedral voids with sphere-union shaped voids, our retrieval rates are reasonable and even higher than expected. Most of the reference voids are detected in a significant fraction by our algorithm; this is mostly because the starting point for our void construction are locally longest edges, which are usually found where the largest circumspheres are, and consequently having a similar core to F&N's. The growth process is also similar; while we do not evaluate neither the tetrahedra volume or their circumspheres, both methods attempt to grow voids in a direction of increasing density, so some similarity was expected.

Although the longest edge criterion had better results over artificial datasets both in 2-d and 3-d, the volume criterion's results were more similar to F&N. This does not imply that the longest edge criterion will perform worse than the volume void finder, but further refinement might be required to its use over real galaxy data.

While the circumpspheres' volume of our tetrahedra will always be greater than the corresponding tetrahedra's volumes (and thus, if DELFIN used the same criteria than F&N, our voids would always be smaller in comparison), in practice our voids were not significantly smaller. First, our algorithm had no cap on the size of the tetrahedra a void may include, and thus, the boundaries of our algorithm were different and could extend further than F&N. Furthermore, several joined tetrahedra may share similar circumcircles, which in our algorithm only adds extra volume to our voids, whereas on F&N's algorithm, most of this volume was already present on the other maximal sphere, marginally increasing the void volume and thus reducing the effect described in heading. Our algorithm is also more flexible in the sense that we allow joining of prevoids if there is a long enough edge being shared among them (as [14] does when void patch boundaries were not significant in their void hierarchy), breaking the paradigm of growing voids strictly towards increasing density regions. These might be two voids going through a merging process, and in this case, we interpret these voids as a single one when they share an edge of a length expected to be found between neighboring void galaxies.

Several other ideas in this direction might improve DELFIN further. For example, a more radical modification to make our algorithm more flexible in terms of the size of voids it may find, could be adding a new prevoid filter to discard voids whose longest to shortest edge ratio is lower than a constant, that is, explicitly accepting or discarding a void depending on the density contrast between its center and its boundaries, ignoring the average density of the sample.

We also believe that the boundary void handling could be revised. In fact, assuming that

the galaxy/particle sample is immovable, the existing tetrahedra in the tessellation may only change if a new point is added inside its circumcircle. So, those triangles whose circumcircle lies completely inside the boundary of the sample will not be altered by the presence of new points outside the sample boundary. Then, a stricter criterion to handle border voids could be marking those tetrahedra which might change given certain distribution of points beyond the sample boundary (i.e., those whose circumsphere extends beyond the sample boundary). The only voids which may not change no matter the distribution of points beyond the boundary are those which **(a)** have no marked tetrahedra in their interior and **(b)** do not share any edges with a marked tetrahedron. While it will effectively discard any possible spurious voids, this criterion might still be too strict and will have to be studied further before being implemented.

3.3.4. Performance evaluation

We ran our 3-d DELFIN implementation over random generated sets of n points in an $N \times N$ square ($N = 100$ Mpc$/h$), using spherical void datasets. The experiment was repeated for values of $n = 1000 \times 2^0, 1000 \times 2^1, \ldots, 1000 \times 2^{10}$ using a density contrast equal to 10. Only the running time of our own algorithm was considered, discarding the time required for qdelaunay to generate the Delaunay tessellations (a very rough estimate of the time required considering both qdelaunay execution and our code is about 1.2 times the times described here).

The system used was a dual core Intel Core i3 3110M processor and 4 GB of memory running an up to date Arch Linux distribution. We stopped testing when we used 1024000 points since the memory required for the next test (with 2048000 points, roughly 5.8 GB of memory usage for points, edges and tetrahedra) would require more than our available memory. Our results can be seen in figure 3.14.

Our running time adjusts well to a linearithmic curve ($O(n \log n)$), satisfying the complexity analysis we had done earlier (under the assumption that the number of tetrahedra is linear in the number of vertices). We consider our algorithm runs in a reasonable time frame, making its use practical and comfortable (as a reference, F&N's implementation runs in quadratic time, taking approximately an hour to process the SDSS dataset we used previously). Hypothetically, assuming this $O(n \log n)$ curve does not change and given enough memory, our algorithm could process the full sample of 12 million DM particles used in the Aspen-Amsterdam project in roughly 7-8 hours.

Still, the algorithm can potentially be parallelized at several stages of its runtime. For example, the prevoid construction might be parallelized if race conditions over edges are handled correctly; as other operations over geometrical data are expensive, this syncing would not represent such a high impact to performance. However, a more thorough analysis must be done before attempting such modification.

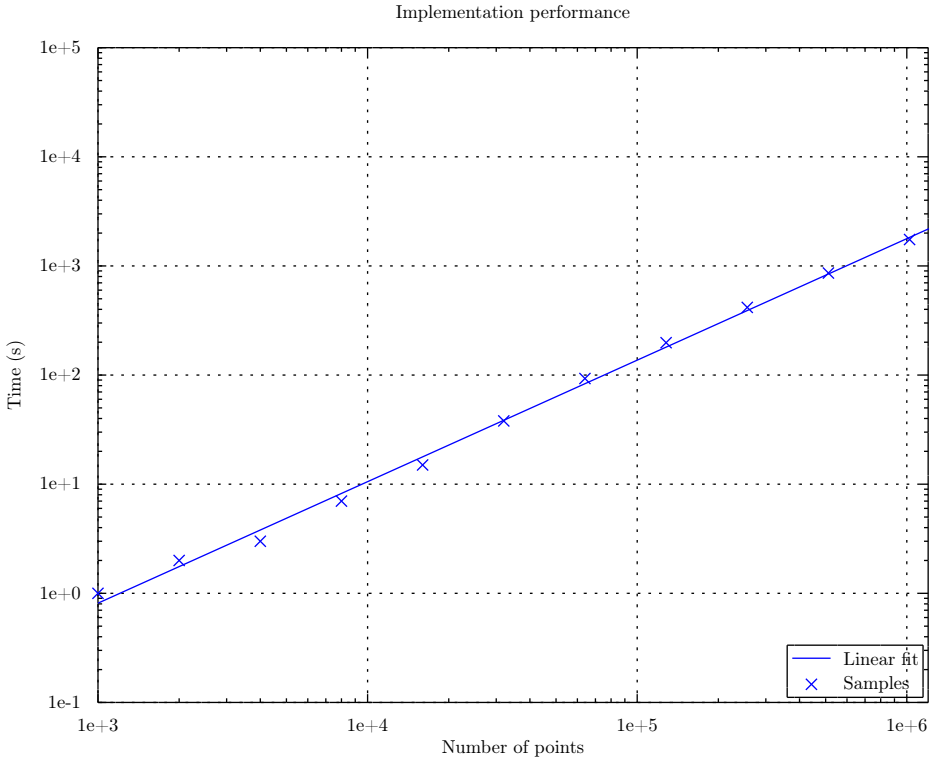


Figure 3.14: DELFIN performance over data sets with $n = 1000, 2000, \ldots, 1024000$ points.

Conclusion

During this work, we have summarized some approaches to the void finding problem. We have also studied a novel approach to the void finding problem by using Delaunay tessellations, and evaluated several extensions. First, we considered four different post processing method for the 2-d version. Our results show that the threshold edge method is the most robust under our mock data sets; the properties of this method extend naturally to 3-d. We also evaluated an alternative prevoid filter based on the longest edge of a void. This filter performed better than the area filter in both 2-d and 3-d artificial datasets. However, its results diverge from those of Foster & Nelson[7]. This was unexpected since longest edges are indirectly related to sphere radii, whereas total void volumes are not.

We have presented our algorithm as a convergence point between existing watershed transform-based and maximal sphere-based algorithms. Delaunay tessellations, in the context of void finding, provide most of the benefits of both maximal circumspheres and Voronoi tessellations while eluding several of their shortcomings. Although we had intended to fix a void definition, we decided not to do so as such a choice would be arbitrary. However, we implicitly assume voids to be those found by our definition, in particular when discarding boundary voids.

Our extensions to the DELFIN algorithm provide an efficient and robust algorithm to detect voids, applicable over large galaxy surveys. Both 2-d and 3-d implementations have been validated and benchmarked with satisfying results. Our algorithm is highly efficient, running in time $O(n \log n)$ in practice. It also provides high flexibility, allowing different strategies or paradigms by making only a few changes.

We have also validated our algorithm by running it over artificial data sets and by comparing it against the void catalog obtained by F&N. Over artificial datasets, our algorithm had a good behavior up from a density contrast equal to 10, degrading rapidly as the density contrast dropped closer to 1. For SDSS data, our algorithm identified 68% of F&N's catalog voids and obtained a median retrieval rate of 61% under 1-to-1 matching. Also, our algorithm identified some new voids over the SDSS DR5 sample used by F&N.

Future Work

It would be interesting if other variants of our void finding algorithm could be developed and tested against current results. We have mentioned some of them, such as developing a completely scale-agnostic algorithm, based exclusively on density contrasts. Such an algorithm would be able to find voids of any size, given they are well defined. However, further research and validation would be required before it could be used for void research.

A discussion about the weaknesses of our current boundary void handling was presented in section 3.3.3. We have given a first approach to discard spurious boundary voids based on circumsphere coverage, however it might be too strict and reject some voids whose shape can be proven definitive according to our implicit void definition.

Lastly, we recognize the need for further validation and testing. It would be interesting to observe how does our algorithm compare to a watershed based algorithm, to prove our claims about our algorithm being a middle point between maximal sphere and watershed algorithms.

Glossary

α-shape In two or more dimensions, given a set of points S and a real number as α, the α-shape of S is the intersection of all spheres of radius $1/\alpha$ which contain every point in S. Note that α may be 0: in this case, the spheres correspond to half-planes, and the 0-shape is exactly the convex hull of S. Additionally, when $\alpha < 0$, the alpha shape becomes the intersection of all of the complements of a sphere of radius $1/\alpha$, where the complement contains every point in S. Also, the α-complex is the set of edges and/or simplices which are at least partially contained in the α-shape. It can be shown that for any $\alpha < 0$, these simplices correspond exactly to those whose circumcircle radius is less than $-1/\alpha$.

Convex Hull The convex hull of a set of geometrical elements, such as points, curves and polytopes, is the smallest convex polytope containing all of the elements in the set. For example, in 2-d, the convex hull of a set of points is the smallest convex polygon containing all of these points. The vertices of the convex hull of a set of points are always a subset of the set of points. In 2-d, ordered vertices are enough to represent the convex hull, but additional edge or face information is required in a 3-d scenario to describe it completely.

$\mathbf{L}^*$ A characteristic luminosity, corresponding to a total magnitude of $M_B = -20.6$.

Megaparsec/h (Mpc/h) A unit of distance used to measure distance between astronomical objects at intergalaxy scales. One Megaparsec is equal to 3.09×10^{19} km. The value h is a constant which depends on the Hubble constant for the expansion rate of the universe, and is approximately 0.68.

Union-find Data structure used to track disjoint sets among a group of elements. The initial state of the structure keeps each element in a different set. The structure supports *union* operations which, given two elements, unites the sets to which they belong (optionally indicating if the elements already belong to the same set); and *find* operations which, given any element, provides an unique representative for the set containing the element. Each operation takes amortized time $O(n\alpha(n))$, where $\alpha^{-1}(n) = A(n, n)$ and A is the Ackermann's function.

Bibliography

[1] J. K. Adelman-McCarthy et al. The fifth data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 172(2):634, 2007.

[2] S. Alam and et al. The Eleventh and Twelfth Data Releases of the Sloan Digital Sky Survey: Final Data from SDSS-III. *The Astrophysical Journal Supplement Series*, 219:12, July 2015.

[3] 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.

[4] J. M. Colberg et al. The Aspen–Amsterdam void finder comparison project. *Monthly Notices of the Royal Astronomical Society*, 387(2):933–944, 2008.

[5] M. Colless et al. The 2dF Galaxy Redshift Survey: spectra and redshifts. *Monthly Notices of the Royal Astronomical Society*, 328(4):1039–1063, 2001.

[6] H. El-Ad and T. Piran. Voids in the large-scale structure. *The Astrophysical Journal*, 491(2):421, 1997.

[7] Caroline Foster and Lorne A. Nelson. The size, shape and orientation of cosmological voids in the sloan digital sky survey. *Astrophys. J.*, 699:1252 – 1260, 2009.

[8] Carlos Hervías, Nancy Hitschfeld-Kahler, Luis E. Campusano, and Giselle Font. *Proceedings of the 22nd International Meshing Roundtable*, chapter On Finding Large Polygonal Voids Using Delaunay Triangulation: The Case of Planar Point Sets, pages 275–292. Springer International Publishing, Cham, 2014.

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

[10] 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, 1991.

[11] R. P. Kirshner, A. Oemler, Jr., P. L. Schechter, and S. A. Shectman. A million cubic megaparsec void in Bootes. *The Astrophysical Journal*, 248:L57–L60, September 1981.

[12] Mark C. Neyrinck. ZOBOV: a parameter-free void-finding algorithm. *Monthly Notices*

of the Royal Astronomical Society, 386(4):2101 – 2109, 2008.

[13] P. J. E. Peebles. The void phenomenon. *The Astrophysical Journal*, 557(2):495, 2001.

[14] Erwin Platen, Rien van de Weygaert, and Bernard J. T. Jones. A Cosmic Watershed: the WVF Void Detection Technique. *Monthly Notices of the Royal Astronomical Society*, 380:551 – 570, 2007.

[15] V. Springel et al. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435:629–636, June 2005.

[16] P. M. Sutter et al. Voids in the SDSS DR9: observations, simulations, and the impact of the survey mask. *Monthly Notices of the Royal Astronomical Society*, 442(4):3127–3137, 2014.

[17] Anton Tikhonov and Anatoly Klypin. *Galaxies in the Local Volume*, chapter Properties of Voids in the Local Volume, pages 31–36. Springer Netherlands, Dordrecht, 2008.

[18] M. J. Way et al. Structure in the 3d galaxy distribution. ii. voids and watersheds of local maxima and minima. *The Astrophysical Journal*, 799(1):95, 2015.