Astronomy and Computing 22 (2018) 48–62

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Delaunay based algorithm for finding polygonal voids in planar point sets

R. Alonso [a], J. Ojeda [a], N. Hitschfeld [a,*], C. Hervías [b], L.E. Campusano [b]

[a] Department of Computer Science, Faculty of Physical and Mathematical Sciences, University of Chile, Chile
[b] Astronomy Department, Faculty of Physical and Mathematical Sciences, University of Chile, Chile

ARTICLE INFO

Article history:
Received 24 November 2016
Accepted 1 January 2018
Available online 11 January 2018

Keywords:
Mesh geometry models
Join algorithms
Data analysis
Large-scale structure of universe

ABSTRACT

This paper presents a new algorithm to find under-dense regions called voids inside a 2D point set. The algorithm starts from terminal-edges (local longest-edges) in a Delaunay triangulation and builds the largest possible low density terminal-edge regions around them. A terminal-edge region can represent either an entire void or part of a void (subvoid). Using artificial data sets, the case of voids that are detected as several adjacent subvoids is analyzed and four subvoid joining criteria are proposed and evaluated. Since this work is inspired on searches of a more robust, effective and efficient algorithm to find 3D cosmological voids the evaluation of the joining criteria considers this context. However, the design of the algorithm permits its adaption to the requirements of any similar application.

1. Introduction

A practical geometrical problem is to find underdense zones in either 2D planar point sets or 3D volumetric point sets and represent their shapes by using simple polygons or polyhedra, respectively. This problem appears in many areas of science but it is particularly relevant in astronomy, where regions almost empty of bright galaxies in the 3-D galaxy distribution are known as cosmological voids.

In the astronomical literature there are several algorithms for finding large empty spaces in the 3-dimensional distribution of galaxies, however there is not yet a generally accepted method to find and characterize them. Some algorithms are based on growing spheres (see El-Ad and Piran (1997), Hoyle and Vogeley (2002) and Foster and Nelson (2009) for example). That is, the algorithm divides the point space (where each point represents a galaxy position) into cells (grid) and then it finds maximal empty balls starting from the center of each cell. Balls are grown in a given direction until their surfaces make contact with a point. Afterwards these balls are merged and processed to identify cosmological voids. Other algorithms first build the Voronoi diagram or Delaunay tessellation of the point set and use it mainly for computing the point density field (Platen et al., 2007; Neyrinck, 2008). In particular, the algorithm proposed in Platen et al. (2007) starts with a Delaunay tessellation but after computing the density fields it works on a cubic grid. Both methods use a flooding algorithm based on the watershed transform to find the voids. It is worth pointing out that they mention that it could be a good idea to work further over the Delaunay tessellation yet they did not. In Hervías et al. (2013), we proposed an algorithm that only uses the Delaunay tessellation to find underdense regions. The algorithm recovered voids of convex shape well but voids of non-convex shape were fragmented. Very recently, an algorithm that also uses the Delaunay tessellation during the whole process was presented in Way et al. (2015). This algorithm applies the watershed transform over tetrahedra to find and build voids.

In computational geometry, a similar and well studied problem is to find large convex holes in a planar point set P. A convex hole is represented by a convex polygon that contains no point of P in its interior. The key question here is the expected size (number of vertices) of the convex polygons that can be found (Pinchasi et al., 2006; Balogh et al., 2013). A special case of this problem, is to find the largest empty rectangle and an extension to the empty rectangle problem is to look for the empty staircase polygon of largest area (Nandy and Bhattacharya, 2003). This last problem appears in Very-large-scale integration (VLSI) layout design. It appears to be that none of the existing solutions use the Delaunay triangulations of the point set as the basis for finding large empty spaces, possibly because they usually deal with convex holes. Such solutions would not generally apply to the problem in astronomy, where the cosmological voids are almost always not convex and may contain a few galaxies in its interior.

Another related problem, the computation of the α-shape of a set of points (Edelsbrunner et al., 1983) which is a generalization of the convex hull of a set of points, can also be seen as a related

* Correspondence to: Beauchef 851, Santiago, Chile.
E-mail addresses: ralonso@dcc.uchile.cl (R. Alonso), jojeda@dcc.uchile.cl (J. Ojeda), nancy@dcc.uchile.cl (N. Hitschfeld), chervias@das.uchile.cl (C. Hervías), luis@das.uchile.cl (L.E. Campusano).

https://doi.org/10.1016/j.ascom.2018.01.001
2213-1337/

problem. The α-shape is represented by a set of polylines, closed or not that adjust "better" to the set of points. The algorithm starts by building a Delaunay triangulation of the set of points, and then, according to the radius of a circular rubber, eliminates all the edges of the triangulation that are greater than this value. The objective of the α-shape algorithm is to represent the shape of the set of points and not to compute the empty spaces inside the convex hull of the set of points. As it is designed, the eliminated edges depend on an user parameter and this value does not change while the algorithm is applied. This is not the case when we find astronomical voids, which is why we did not adapt/extend the alpha shape algorithm to find voids.

As we have said before, cosmological voids are located in the 3D space and a useful algorithm should find these 3D voids. However, first we decided to focus in building an effective and robust 2D algorithm that can be extended to 3D before developing an efficient, effective and robust 3D version. In addition, we think that the 2D algorithm can be useful to solve similar problems in other fields.

This paper presents a new approach to detect and build simple polygons representing underdense regions or voids on a planar point set. The algorithm is divided in two main steps: (1) *Building subvoids* and (2) *Joining subvoids*. In the first step, each triangle is attached to the terminal-edge region (a triangle set) it belongs to. In the second step, fragmented voids, represented by several terminal-edge regions, are joined to build whole voids. Each void is described by a simple polygon defined by the edges which appear only once in the triangle set. The main ideas behind step (1) and its application to the detection of 2D and 3D convex voids were presented in Hervías et al. (2013), without (i) the theoretical foundations that support the algorithm, (ii) the detection of non-convex polygonal voids nor (iii) implementation issues. We have called the algorithm DELFIN (*DEL* aunay void *FIN* der).

The paper is organized as follows: Section 2 recalls the properties of the Delaunay triangulation and the Voronoi diagram and introduces basic geometrical concepts. Section 3 presents the main ideas of the algorithm and the theoretical foundations that support it. Section 4 describes the implementation of the algorithm including the handling of special cases. Section 5 shows how the algorithm was validated with artificial data and Section 6 includes the analysis of the experimental evaluation. Finally Section 7 gives some conclusions and outlines the ongoing work.

2. Basic concepts

The fundamental ideas of the proposed algorithm are based on the properties of Delaunay triangulations. That is why we start this section recalling them. Then we introduce several definitions; some of them are already known in the literature related to triangulations but they are necessary to clearly describe and demonstrate the basis of the proposed algorithm.

2.1. Voronoi tessellations and Delaunay triangulations

Voronoi tessellations (Voronoi, 1908) and Delaunay triangulations (Delaunay, 1934) are dual geometrical structures that allow one to find quickly the input points (sites) that are (locally) close or far from each other. The Voronoi tessellation consists of convex polygonal regions, one for each site, whose interior points are closer to this site than to any other site. Voronoi edges represent points equidistant to two sites. The Delaunay triangulation is obtained by adding edges between the sites that share a Voronoi edge. The endpoints of a Delaunay edge show that these sites are the closest ones in that direction. Since each edge is a Delaunay edge, there exists an empty circle that passes through its endpoints. Then, there are circular regions that do not contain any other site in its interior (O'Rourke, 1998). Such a circle may be centered in any point of the shared edge (Voronoi edge) between the corresponding Voronoi regions. The locally largest circular regions will be considered as the starting point of the proposed algorithm which we are going to explain in detail in the next sections.

2.2. Geometrical concepts

Let T be a conforming triangulation of n points and m triangles. Let t_i be a triangle, $i = 0, \ldots, m-1$.

Definition 2.1. Longest-, second-longest- and shortest-edge: The longest-edge of a triangle is the edge with the largest length; the second-longest edge is the edge with the intermediate length, and the shortest-edge is the one with the smallest length. In case of equilateral or isosceles triangles, the equal length edges are arbitrarily ordered. Then any triangle, including equilateral and isosceles triangles, has a well defined size order for its edges.

In Section 4, we describe how the algorithm gives the arbitrary order to the triangle edges of equilateral and isosceles triangles, and in Section 6.4, we discuss why this order does not impact the results.

The following definitions and theorems constitute the basis of our method and they require and use the concepts of longest-, seond-longest- and shortest-edge. We will use the terms *Longest-edge propagation path* and *Terminal-edge* were first introduced by Rivara in the context of new algorithms for the refinement/improvement of triangulations (Rivara, 1997) to characterize the *terminal-edge regions*, that is, the void candidates.

Definition 2.2. Terminal-triangles and -edges (Rivara et al., 2001): Two triangles are called *terminal triangles* if they share their longest-edge. This shared longest-edge is called *terminal-edge*.

Definition 2.3. Longest-edge propagation path (Rivara et al., 2001): For any triangle t_0 in any conforming triangulation T, the *Longest-Edge Propagation Path* of t_0 (Lepp(t_0)) will be the ordered list of all the triangles $t_0, t_1, t_2, \ldots, t_{l-1}, t_l$, such that t_i is the neighbor triangle of t_{i-1} by the longest-edge of t_{i-1}, for $i = 1, 2, \ldots, l$. The longest-edge shared by t_{l-1} and t_l is a terminal-edge and t_{l-1} and t_l are terminal-triangles.

Definition 2.4. Boundary edge and boundary terminal-edge: The edges that belong to only one triangle of a triangulation are called *boundary edges*. If a boundary edge is the longest-edge of the triangle, it will be called *boundary terminal-edge*.

Definition 2.5. Voids: Let S be a planar set of points and let P be the polygon that represents its convex hull. A *void* is an underdense region inside P. Voids can be represented by any simple polygon (convex or not).

The term *void* is commonly used in astronomy to describe underdense point regions in the space. Note that in one extreme a void can be empty and, if it has a convex shape, can coincide with the concept "convex hole" used in the computational geometry area, but in the other extreme it is enough for the point density inside a polygon to be lower than the background density. The threshold density value to consider a region as a void may be defined by the application.

3. The method

Let S be a set of planar points and T its Delaunay triangulation. Since each edge of T is a Delaunay edge, there exists an empty circle that passes through its endpoints. The question now is where voids can be located. The largest empty circles are natural candidates to

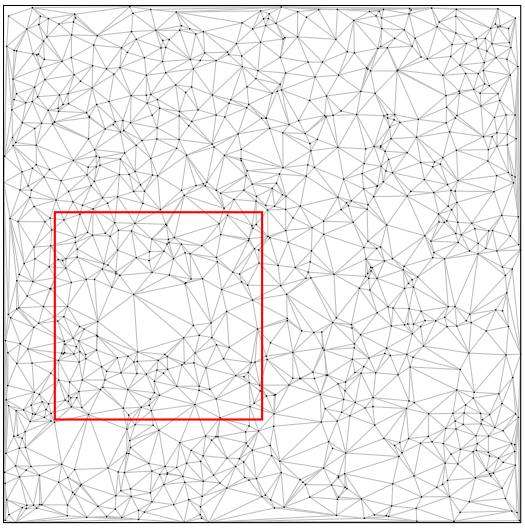


Fig. 1. Delaunay triangulation of a point set. The square window shows a subset of points.

be part of voids. This fact influenced us to design an algorithm that starting from two triangles that share a terminal-edge (initial triangle set), builds the largest possible set of triangles (final triangle set) of equal or increasing density by adding neighboring triangles whose longest edge is shared with a triangle of the current set. The algorithm stops when no neighboring triangle fulfills this condition. The void candidate is defined by the simple polygon formed by the edges that are only once in the final triangle set.

Fig. 1 shows the Delaunay triangulation of a point set. The subset of points and triangles inside the square window will be used in Fig. 2 to illustrate the steps of the proposed method. Fig. 2(a) shows this subset of points and Fig. 2(b) the corresponding Delaunay triangulation clipped by the square window sides. The largest terminal-edge and the two terminal-triangles are drawn as a thick red line and light blue triangles in Fig. 2(c) and Fig. 2(d), respectively. These triangles are the seed triangles to start the computation (initial triangle set). Figs. 2(e) and 2(f) show in light blue the two neighbor triangles that share their longest-edge with the terminal-triangles. These are added to the current triangle set. Figs. 2(g) and 2(h) show the added triangles next and the final triangle set is shown in Fig. 2(i). This region will be considered as a void candidate.

3.1. Theoretical foundations

In this section we define the concept of terminal-edge region and demonstrate that terminal-edge regions partition the space without overlapping. As we have said before, for the cases of equilateral and isosceles triangles the edges of equal length are also assigned an order. Then every triangle has a unique longest-edge, a second longest-edge and a shortest-edge.

Definition 3.1. Terminal-edge region: A *terminal-edge region* R is a region formed by the union of all triangles t_i such that Lepp(t_i) has the same terminal-edge. In case the terminal-edge is a boundary-edge the region will be called *boundary terminal-edge region*.

Fig. 2(i) shows in light blue the terminal-edge region of the terminal-edge shown in Fig. 2(c).

Definition 3.2. Frontier-edge: A *frontier-edge* is an edge that is shared by two triangles, each one belonging to a different terminal-edge region.

Lemma 3.1. *Let t_i and t_j be two triangles that share the edge e. If the edge e is a frontier-edge, then e is neither the longest-edge of t_i nor of t_j.*

Proof. Let us assume that e is either the longest-edge of t_i or t_j. Then t_i would belong to the Lepp(t_j) or t_j belongs to the Lepp(t_i) through e. This means that both triangles belong to the same terminal-edge region and this contradicts that e is a frontier edge. □

Note that the opposite is false. In fact, Fig. 3 shows two triangles t_i, t_j whose shared edge is not the longest-edge of any of them and these two triangles belong to the same terminal-edge region R_i. e is still a particular case frontier-edge in the sense that the region R_i cannot grow through it. But, since e does not separate two different regions we designate this kind of edge as *barrier-edges*.

Definition 3.3. Barrier-edge: A *barrier-edge* of a region R is an edge shared by two triangles of R which is not the longest-edge of any of them.

Definition 3.4. Internal-edge: An *internal-edge* of a region R is an edge that is neither a terminal-edge, a frontier-edge, a barrier-edge nor a boundary-edge.

Lemma 3.2. *The edge e is an internal-edge of a terminal-edge region iff e is the longest-edge of only one of two triangles that share it.*

Proof. The proof is direct from the definition of the Longest-edge propagation path and from the construction of terminal-edge regions. □

Lemma 3.3. *Let T be a conforming triangulation of any set of points P and let CH be the convex hull of P. Then the set of terminal-edge regions in T do not overlap.*

Proof. Let us assume by contradiction that t_j belongs to R_i and R_k, the terminal-edge regions associated to edges e_i and e_k, respectively. Since by Definition 2.1, t_j has only one longest-edge, Lepp(t_j) finishes in either e_i or e_k, but not in both. Then t_j belongs only to one region and this contradicts our assumption. □

Fig. 4(c) shows a set of terminal edge regions built from the triangulation shown in Fig. 4(b). Large terminal-edge regions will be natural void candidates as the one in gray at the center of Fig. 4(c). Note that we cannot always model a large empty space with only one terminal-edge region as shown in Fig. 5(b). In this example we show an artificially inserted non-convex polygonal void and its fragmentation in seven terminal-edge regions generated by our original algorithm (Hervías et al., 2013). Since for some applications such as finding cosmological voids it is important to recognize these subvoids as part of a larger void, in the next section we propose several geometric criteria that can be used to join subvoids before we describe the algorithm implementation.

3.2. Criteria to join subvoids

The criteria to join terminal-edge regions will certainly depend on the problem to be solved. As we have said before, since our motivation is to develop a 3D cosmic void detection algorithm, we have looked for criteria that fulfill the following requirements:

- Easily extensible to 3D.
- Devoid of any preconceived assumptions about shape.
- Appropriate to join adjacent terminal-edge regions (regions that share at least one frontier-edge).
- Use of local information (only from involved terminal-edge regions).

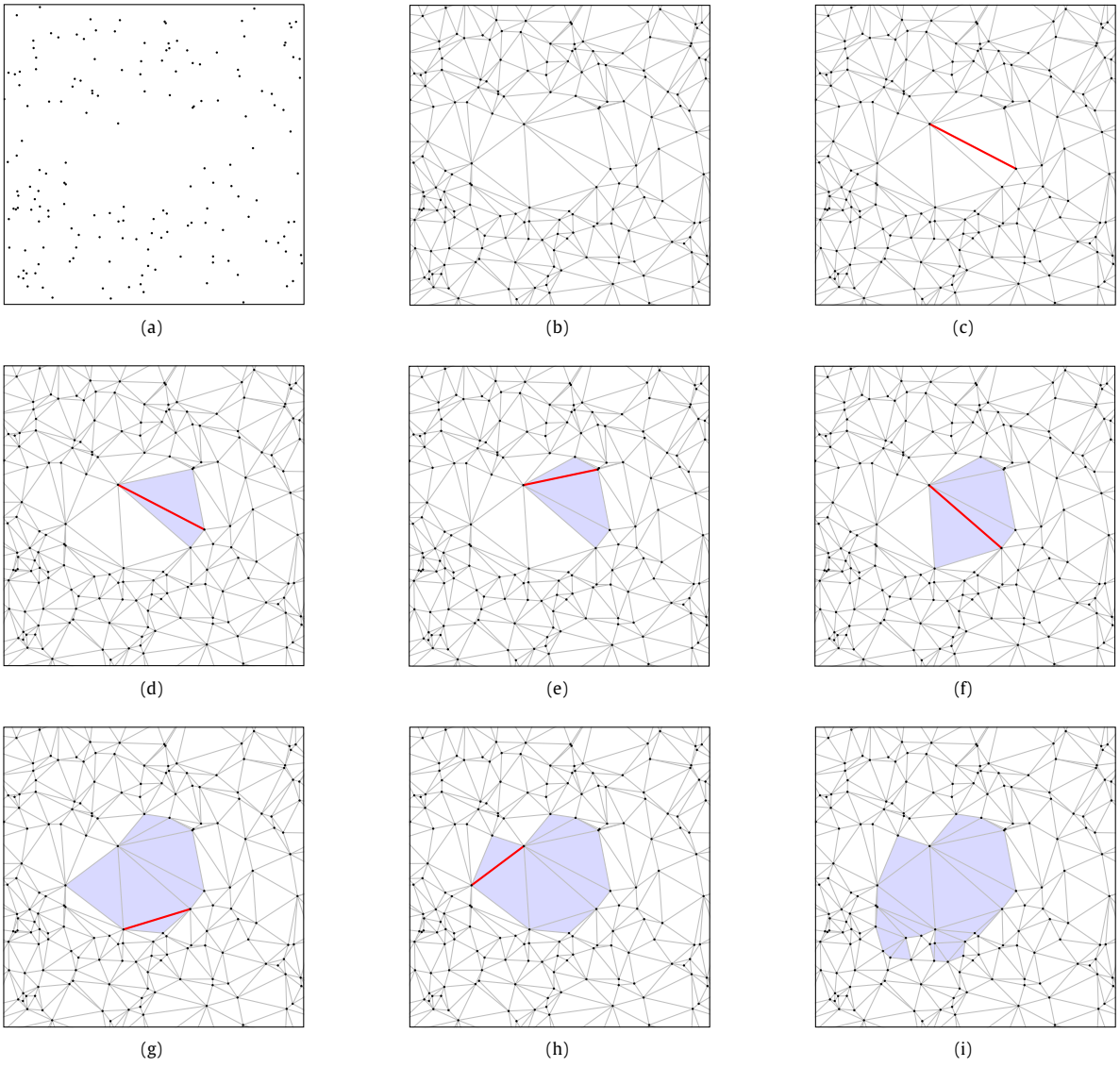


Fig. 2. Illustration showcasing several steps of the algorithm. The longest-edge for the most recently added triangle is shown marked with a thick red line. The current triangle set is shown in gray.

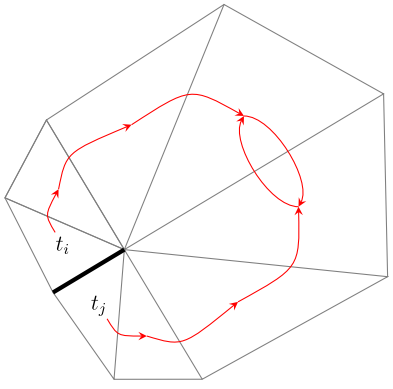
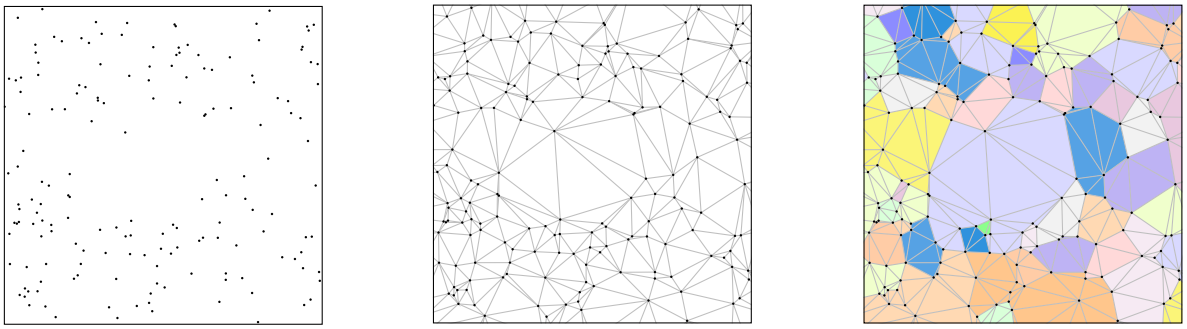


Fig. 3. Example of a barrier-edge (thick line). Red arrows indicate the direction of the Lepp for each triangle towards a terminal-edge. The result is a single terminal-edge region.

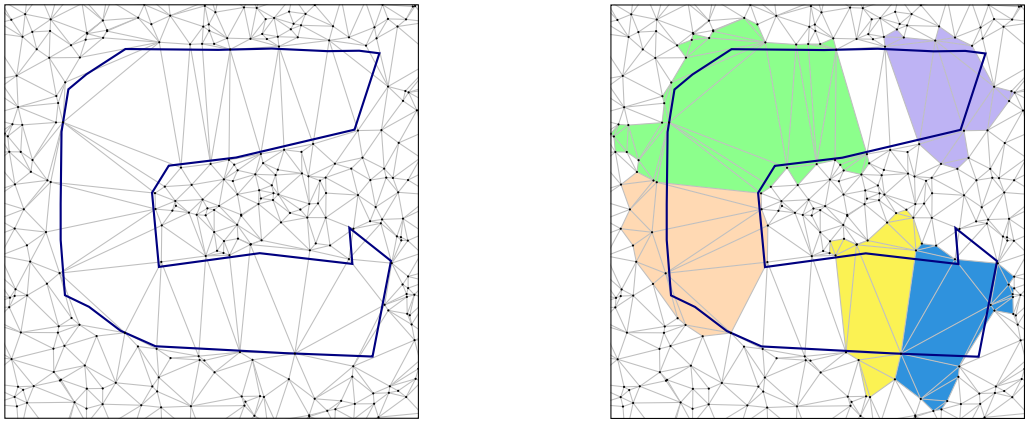
So, we have selected four criteria named *Arc*, *Frontier*, *Second-longest edge*, and *Frontier-edge*. The *Frontier-edge* and *Arc* criteria are based on existing methods (Platen et al. (2007) and Foster and Nelson (2009), respectively) to join 3D cosmic subvoids, and we have adapted them to join 2D artificial subvoids. The *Frontier* and *Second-longest edge* criteria were designed by us and to the best of our knowledge has not been used for the detection of cosmic voids.

- *Arc* criterion: Let $C_i(c_i, r_i)$ be the circle whose center c_i is the centroid of the terminal-edge region R_i and whose radius r_i is such that $area(C_i) = area(R_i)$. Let a_i and a_j be the arcs that define $C_i \cap C_j$. The same notation if valid for $C_j(c_j, r_j)$ and R_j (see Fig. 6(a)). If $area(C_i) \geq area(C_j)$ then, R_i and R_j are joined if $angle(a_i) = \alpha_i \geq \theta_{min}$, and vice-versa.
- *Frontier* criterion: Let p_i be the perimeter of the terminal-edge region R_i and p_j of R_j. Let l_{ij} be the sum of the frontier-edge lengths shared by R_i and R_j as shown in Fig. 6(b). The regions R_i and R_j are joined if the ratio $l_{ij}/max(p_i, p_j) \geq L$
- *Second longest-edge* criterion: two terminal-edge regions are joined if any of their shared frontier-edges is the second longest-edge of at least one of the triangles that share it. In Fig. 6(c), the edge L_{12} shared by the triangles t_1 and t_2 is the second-longest edge of t_1.



(a) (b) (c)

Fig. 4. Terminal-edge regions inside the square window of Fig. 1. Each region is shown with a different color. (a) set of points, (b) Delaunay triangulation, (c) terminal-edge regions. The gray region at the center is the one with largest area.



(a) Artificial non-convex polygonal void.

(b) The void is recognized as seven terminal-edge regions.

Fig. 5. Voids and subvoids. (a) The artificial void boundary is drawn on the triangulation. Neither its points, not its edges are part of the triangulation. (b) The seven found terminal-edge regions are shown using different colors including the two white regions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

- *Frontier-edge* criterion: two terminal-edge regions R_i and R_j are joined if the length of any of their shared frontier-edges is larger than a threshold value. Fig. 6(d) shows a frontier-edge L_{12}, the edge shared by triangles t_1 and t_2, as one of the edges to be checked against the threshold value.

The robustness and effectiveness of these criteria are evaluated in Section 5 by applying them to detect both convex and non-convex artificial void structures.

4. The DELFIN algorithm

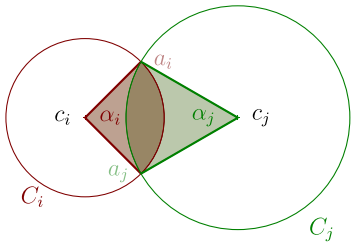
This section presents the algorithm that implements the method described in Section 3. The algorithm was split into two main steps:

1. *Building subvoids*: the algorithm builds terminal-edge regions from the Delaunay triangulation of the input points. Too small terminal-edge regions are discarded by using either area (a_s) or terminal-edge length (e_{min}) threshold values. The remaining regions are considered as subvoids. Both area (a_s) and terminal-edge length (e_{min}) criteria are known in the rest of the paper as discarding or filter criteria.
2. *Joining subvoids*: the remaining terminal-edge regions (i.e. subvoids) are joined into candidate voids if they fulfill the criterion specified by the user (Section 3.2). Candidate voids are returned as voids if their area is larger than a minimum valid void area (a_v).

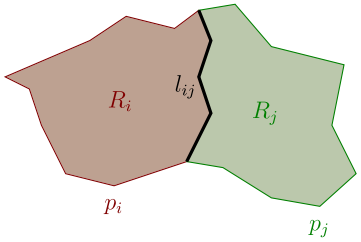
4.1. Building subvoids

The *Building subvoids* algorithm is shown in Algorithm 1. First, it builds the Delaunay triangulation of the input point set. Then, it sorts the triangles by their longest-edge length and these edges are processed in order from the triangle of largest longest-edge to the shortest one. Each terminal-edge region is built as follows: first, the largest non-used longest-edge (either a terminal-edge or a boundary terminal-edge), whose terminal triangles (two or one, respectively) are used to initialize **triangle_set** and represent the seed terminal-edge region. Next, the triangles that share their longest-edge with triangles of this terminal-edge region are added to the current **triangle_set** and labeled as used. When no triangle can be added to **triangle_set**, the terminal-edge region is complete. If this terminal-edge region passes the discarding criterion, it is stored as a valid subvoid in **subvoid_list**. As a result, multiple disjoint sets are obtained, each one corresponding to a subvoid.

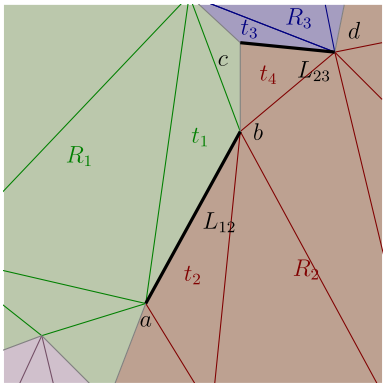
For the sake of clarity, Algorithm 1 considers only terminal-edges shared by two triangles. However the whole algorithm also includes the case when the longest-edge is a boundary terminal-edge and only one triangle is assigned to **triangle_set**. The rest of the algorithm remains the same. It is worth to mention that the equilateral and isosceles triangles are handled in the same way as other triangles. The sorting algorithm creates an ordered list of the triangles according to their longest-edge, while for the isosceles and equilateral triangles, they are included two or three times,



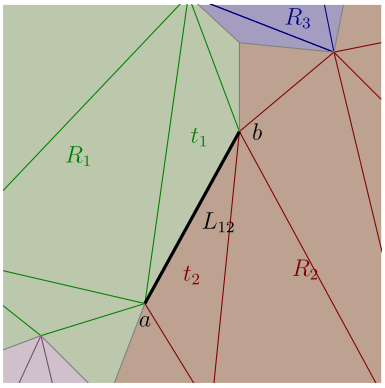
(a) Arc criterion.



(b) Frontier criterion.



(c) 2nd longest-edge criterion.



(d) Frontier-edge criterion.

Fig. 6. Joining subvoid criteria. The green color is used to show the parameters of the terminal-edge region R_i and the red color, the ones for R_j. (a) Arc criterion: Circle C_i is computed from the terminal-edge region R_i and C_j from terminal-edge region R_j. $C_i \cap C_j$ is bounded by a_i and a_j, where a_i is part of the circumference of C_i and a_j is part of C_j. α_i is defined by the size of arc a_i and α_j by the length of a_j. One of these angles will be checked against a threshold value. (b) Frontier criterion: p_i and p_j are the perimeters of regions R_i and R_j, respectively. l_{ij} is the shared perimeter. The ratio between l_{ij} and $max(p_i, p_j)$ is compared to a threshold value. (c) Second longest-edge criterion: L_{12}, the edge shared by triangles t_1 and t_2, is the second longest-edge of t_1. This is an example of two regions, R_1 and R_2, that will be joined. (d) Frontier-edge criterion: L_{12} is an example of a frontier-edge between R_1 and R_2 that will be checked against a threshold value. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Algorithm 1: Building subvoids

```
read the Delaunay triangulation;
read pre_threshold_value;
subvoid_list = ∅;
order triangles by longest-edge;
repeat
    get the two unused triangles t1, t2 sharing 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 discardingCriterion(triangle_set) ≥ pre_threshold_value then
        subvoid_list = subvoid_list ∪ polygon(triangle_set);
    end
until there is no triangle left to process;
```

respectively, in the list. Then, the sorting algorithm gives a natural order to process the edges of isosceles and equilateral triangles which is consistent with Definition 2.1 and the procedure adopted in Sections 2 and 3. Note however that when a longest-edge of an equilateral triangle is used as terminal-edge, the other two edges cannot be used for this same purpose because the triangle has already been labeled as used. The first time it appears as a neighbor triangle of one of the triangles belonging to **triangle_set**, will be added to it and labeled as used. The isosceles triangles with two longest-edges are managed in a similar same way.

As we have mentioned before, we have considered two discarding criteria in this step: (i) the area of a **triangle_set** is less than a threshold area a_s and (ii) the length of the terminal-edge is less than a threshold length e_{min}. These criteria are appropriate while looking for astronomical voids because in order to identify an underdense region as a void, its area and the distance between points are expected to be larger than certain threshold values. In particular, the second criterion allows us for an early halt of the algorithm since terminal-edge regions with its terminal-edge length lower than a threshold value could be discarded.

4.2. Joining subvoids

The *Joining subvoids* algorithm is shown in Algorithm 2 . As we mentioned previously, this step was added because elongated non-convex shapes are likely to be divided into smaller fragments as shown in Fig. 5.

The algorithm works as follows: For each pair of unused subvoids u and v, the specified joining criterion is checked. If u and v fulfill it, they are joined and stored in **void**. The frontier edges shared by u and v are labeled now as internal-edges. For each resulting **void** in **void_list**, a simple polygon is built by using the frontier edges and these polygons are returned as voids if their area is larger than a_v.

Algorithm 2: Joining subvoids

```
repeat
    label subvoid_list as unchanged;
    void_list = ∅;
    foreach unused subvoid v in subvoid_list do
        void = v;
        label v as used;
        foreach unused subvoid u in subvoid_list do
            if void and u meet the specified criterion then
                void = void ∪ u;
                label u as used;
                label subvoid_list as changed;
            end
        end
        void_list = void_list ∪ {void };
    end
    subvoid_list = void_list;
until subvoid_list is not changed;
remove voids from void_list whose area is less than a_v;
```

The detected voids depend on the specified joining criterion to join subvoids. We will discuss the effectiveness of each criterion in recovering artificial void shapes in Section 6.

4.3. Boundary voids

The algorithm does not distinguish boundary voids from internal voids. In some applications, a boundary void should not be considered as a void because it has an open boundary. Even if it is real, there is no way to know it and it could not be described even qualitatively because it is not complete.

We have implemented a very simple strategy having cosmic voids in mind: Any void having one or more edges in the perimeter of the convex hull of the dataset is discarded. An example of a boundary terminal-edge region is shown in Fig. 7. However, if other application requires to keep them, boundary voids can be just properly labeled instead of discarding them.

4.4. Implementation issues

In order to generate a Delaunay triangulation, DELFIN calls the well-known *qdelaunay* (Barber et al., 1996) program to obtain it. The computational cost of the *Building subvoids* step is $O(n \log n)$, where n is the number of points, because of: (i) the construction of the Delaunay triangulation and (ii) the longest-edge ordering.

We are aware that a further performance increase can be achieved by building voids without the longest-edge ordering. Each subvoid can be characterized by a tree whose nodes represent triangles (O'Rourke, 1998). If we set the root of each tree to be a terminal-edge triangle, then we can get the parent of each triangle by determining its longest edge. It is worth noting that this step can be done in any order, so if no pruning is done over the trees, a union-find (disjoint-set) data structure may be used to build voids (and compute for example their area) in quasi-linear time ($O(n\alpha(n))$, where α is the inverse of the Ackermann function and n is the number of triangles). In order to include this improvement, the handling of isosceles and equilateral triangles would require to give an explicit order to their edges. One advantage of the current implementation is that the edge sorting allows the *Building subvoids* algorithm to stop before processing all triangles.

The *Joining subvoids* step takes time $O(cs^2)$, where c is the cost of verifying fulfillment of the criterion and s is the number of remaining subvoids. This is a naïve implementation, however. While it was useful to show the effectiveness and robustness of the algorithm and its cost was not significant in our particular scenarios ($s \leq 500$), it might take a long time if the number of subvoids to join s is far greater than 1000. A better solution would be to maintain a data structure for each void and update the neighboring relations between them. Then, only neighbors of a subvoid would require verification of the criterion. Since the average number of neighbors is capped by a constant, this will take time $O(cs)$ plus an amortized time of $O(s\alpha(s))$, where $\alpha(y) = x$ is the inverse of the Ackermann function $\alpha^{-1}(x) = A(x, x)$. Further implementation details are described in Alonso (2016).

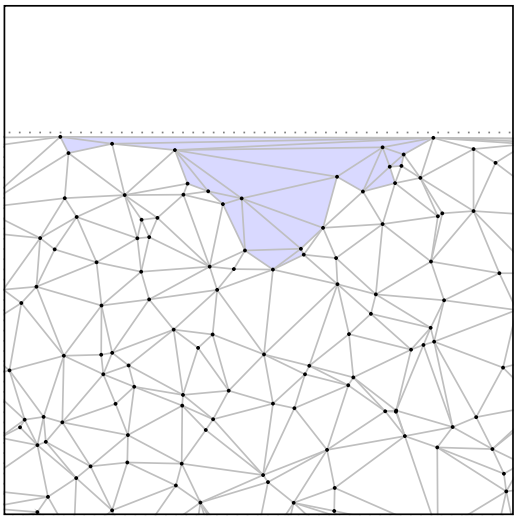


Fig. 7. Boundary terminal-edge region. The dotted points show the boundary.

5. Experimental evaluation

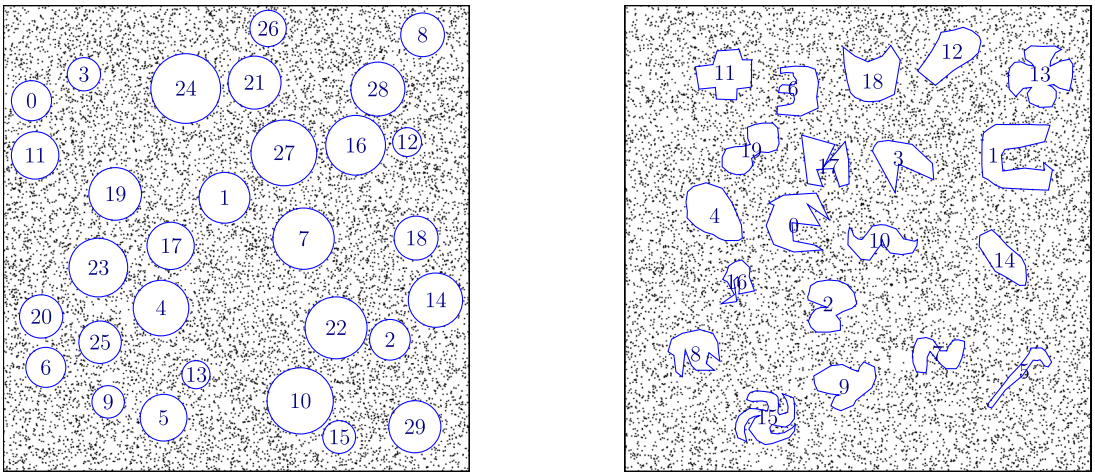
In this section, we evaluate the effectiveness and robustness of the algorithm in terms of how good the artificially inserted voids are recovered. First we describe the data sets we are using, secondly the measures to evaluate the effectiveness of the proposed joining criteria, thirdly the obtained results and finally we discuss the tradeoffs of applying them to different void shapes.

We define *effectiveness* as how well the algorithm retrieves the artificial voids. To measure it, we calculate the usual parameters *recall* and *precision* to compare predictions to expected results. We also define *robustness* as the capacity of an algorithm to deal with erroneous or noisy input. We measure it by the amount an input parameter may change without altering the results.

5.1. Artificial data sets

We have designed artificial data sets to test the algorithm for the search of underdense regions that have convex shapes or irregular shapes. The reason is to evaluate the capacity of the algorithm for finding not only shapes similar to those of the 2D cosmic voids which are mostly convex, but also to search for underdense regions in data sets from other applications.

We have designed two templates, one for convex shapes represented as circles and the other for non-convex shapes represented by non-convex polygons. We have chosen a simple square domain of side $a = 2000$. The first template contains 30 circles generated by randomly chosen centers over a uniform distribution inside an $a \times a$ square space and randomly chosen radii over a uniform distribution over the range $[0.03a, 0.075a)$. The second template contains 20 manually-drawn non-convex polygons also in an $a \times a$



(a) Circular voids. (b) Non-convex voids.

Fig. 8. Artificial data sets of size $n = 10\,000$ and $\rho_B = 1/400$. The artificial voids numbering is consistent with the void labels (Void ID) in the tables.

square space, with their centers and areas also randomly chosen in similar ranges.

Randomly generated points (also over a uniform distribution) were inserted over the square area minus the generated voids. This process was repeated for each template, in order to generate data sets of size $n = 5000$, 10 000, 50 000 with background densities ρ_B respectively equal to 1/800, 1/400 and 1/80 points per square. As a result, our generated voids are completely devoid of points, and their areas are in the range of [10 000, 60 000] (a background point density of $\rho_B = 1/400$ is approximately what is expected while looking for cosmic voids in galaxy distributions).

Fig. 8 illustrates two of the generated test examples, both for $n = 10\,000$ points. Each one shows the size, location and identification number of each void. The identification number will be useful later to identify which shapes are easily recovered and which are not. In particular, Fig. 8(a) shows the 30 empty circular areas and Fig. 8(b) shows the 20 empty non-convex polygonal areas, where the former illustrate convex underdense regions and the latter underdense regions with irregular shapes. Note that the non-convex polygons labeled as 4, 12 and 14 are shapes reminiscent of cosmic voids, while those numbered 2, 17 and 19 could describe interacting voids; the rest of polygons illustrate more complex shapes that may be relevant to other applications.

The input of the algorithm consists on the set of randomly generated points. The set does not include additional points corresponding to the polygons used to represent the artificial voids.

5.2. Threshold values

The algorithm needs either the threshold value a_s or e_{min} to discard very small terminal-edge regions at the end of the *Building subvoids* step, and a_v to produce as output the list of voids with significant area at the end of the *Joining subvoids* step.

The threshold values a_s, e_{min} and a_v must be empirically adjusted according to characteristics of the empty spaces to be looked for. In our case, the values were selected by taking into account the size of voids, the number of points and the average point density. Table 1 shows the threshold values given as input to the algorithm for each n, where n is the number of points. A discussion about how these values were obtained is given in Section 6.3. Note that we kept the square area constant. Then, when increasing the number of points n, the point density ρ_B increases too.

In a similar way, we have chosen the following threshold values for the joining criteria. We have introduced the *none* criterion to represent the results of the original version of DELFIN (Hervías et al., 2013), i.e., the DELFIN output without applying the *Joining subvoids* step. Note that the a_v filter is applied.

Table 1
Threshold values used for each dataset size n and point density ρ_B. a_s and e_{min} are the area and terminal-edge length threshold values, respectively, used to discard the small terminal-edge regions after the *Building subvoids* step, and a_v is the area threshold value used after the *Joining subvoids* step.

n	ρ_B	a_s	e_{min}	a_v
5 000	1/800	6000	80	12 000
10 000	1/400	3000	65	8 000
50 000	1/80	600	35	4 000

- *None* criterion. The joining step is not applied.
- *Arc* criterion: the threshold angle is $\theta_{min} = \pi/3$.
- *Frontier* criterion: the threshold ratio is $L = \frac{1}{16}$.
- *Second longest-edge* criterion: threshold value is not required.
- *Frontier-edge* criterion: for cosmic voids, the minimum edge length is defined by r_3 (Foster and Nelson, 2009):

$$r_3 = d_3 + \lambda\sigma_3 \tag{1}$$

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

The threshold values for the *Frontier* and *Arc* criteria were empirically computed. The *Frontier-edge* threshold value is computed from the input points and it is commonly used in the detection of cosmic voids (Foster and Nelson, 2009).

5.3. Evaluation metrics

In order to show the effectiveness of DELFIN for void retrieval, we have selected two metrics that are commonly used for this purpose in the machine learning and information retrieval literature (Manning et al., 2008). The first one is r_v, the recall rate for void v, and the second is e_v, the overdetection rate for void v. r_v corresponds to the detected fraction of the original void and e_v the non-void fraction of the detected void. More precisely, we define the recall rate of an artificial void v as $r_v = \frac{area(v^* \cap v)}{area(v)}$ where v^* is the void retrieved by the algorithm. Similarly, we define the

Table 2
Number of subvoids remaining after applying the area filter a_s, the joining criteria for recovering each one of the artificial non-convex shapes shown in Fig. 8(b) and the final filter a_v, to keep only the voids with significant area. The criterion labels are: N: *None*, A: *Arc*, F: *Frontier*, S: *Second-longest-edge*, E: *Frontier-edge*. The first column shows the void identification, the next five columns show the fragmentation on low background point density $\rho_B = 1/800$ ($n = 5000$), the second five on middle background point density $\rho_B = 1/400$ ($n = 10\,000$) and the last five, on high background point density $\rho_B = 1/80$ ($n = 50\,000$).

Void	Joining criterion														
#	N	A	F	S	E	N	A	F	S	E	N	A	F	S	E
0	1	1	1	1	1	3	2	1	1	1	2	2	2	1	1
1	4	3	1	2	2	5	4	2	2	1	7	4	2	1	1
2	2	1	1	2	1	3	1	1	1	1	2	1	1	1	1
3	2	2	1	1	1	1	1	1	1	1	2	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1	2	2	2	1	0	1	3	2	1	1	2	1	1	1
6	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
7	2	2	3	5	3	3	3	1	2	3	2	3	1	1	1
8	1	2	1	2	2	1	2	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	3	3	1	1	1	1	1	1
10	1	1	1	0	1	2	1	1	0	1	3	3	1	1	1
11	1	1	1	1	1	1	2	3	2	1	1	1	1	1	1
12	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	2	2	1	2	1	5	3	1	1	1
14	2	2	3	2	1	1	1	4	4	1	2	2	1	1	1
15	1	1	2	1	1	1	2	3	1	1	1	4	2	1	1
16	2	2	1	0	1	1	1	1	1	1	1	2	2	2	1
17	2	1	0	1	2	3	1	0	1	1	2	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1
19	2	2	0	0	1	2	2	1	0	1	2	2	1	1	1
	$n = 5000$					$n = 10\,000$					$n = 50\,000$				

overdetection or error rate of an artificial void v as $e_v = 1 - \frac{area(v^* \cap v)}{area(v^*)}$ where $\frac{area(v^* \cap v)}{area(v^*)}$ is known as the precision rate. Note that a recall rate equal to 1 means the void was completely recovered, while a recall rate equal to 0 means the void was not found. An error rate equal to 0 means that the whole retrieved void area is included in the original void and an error rate close to 1 means that the most of the retrieved area falls out the original void.

The effectivity of different criteria for joining subvoids depends on the shape of the non-convex polygonal voids. Thus we have included a metric to quantify the fragmentation a joining criterion produces over the final retrieved voids. We have expressed this metric as the average number of void fragments (subvoids) that can be associated to each artificial void after a joining criterion was applied.

5.4. Experimental results

We applied a postprocessing step to relate each artificial void v_i to one or more retrieved voids v_j^*. Let V_i^* be the set of retrieved voids v_j^* such that $v_i \cap v_j^* \neq \emptyset$. If $|V_i^*| > 1$ means that the void v_i is fragmented. In this case, the recall rate r_v and the error rate e_v are calculated by using the retrieved void v_i^* ($v_i^* \in V_i^*$) whose centroid is the closest one to the centroid of the given artificial void v_i.

Tables 2 and 3 summarize the fragmentation (number of subvoids) generated for each artificial non-convex shape shown in Fig. 8(b). The first column shows the void identification and the rest, the number of subvoids remaining after applying each joining criterion and the final filter a_v, to keep only the subvoids with area greater than or equal to a_v, the minimum valid void area. The first five columns show the fragmentation on low background point density, the following five on middle background point density and the last five, on high background point density. *N* corresponds to *None* criterion, A refers to the *Arc* criterion, F is the *Frontier* criterion, S is the *Second-longest-edge* criterion and E, the *Frontier-edge* criterion. Note that the *None* criterion represents the number of subvoids generated by the *Building subvoids* step, filtered by the a_v, the minimum area of a valid void. That is why, for void ID 5 and $n = 10\,000$, the number of subvoids is 0. However for the same artificial void, in the case where the *Arc* or *Frontier-edge* criteria were applied, some terminal-edge regions were joined, and the area of only one of the subvoids was larger than a_v. The union generated by applying the *Frontier* and *S*econd-longest-edge generated three and two subvoids larger than a_v, respectively.

From Tables 2 and 3, we obtain two new tables, Table 4(a) for $n = 10\,000$ and Table 4(b) for $n = 50\,000$, to display the average number of void fragments after applying each joining criterion to all to the non-convex voids shown in Fig. 8(b). An average number equal to 1 means that all voids were retrieved as a single polygon, thus no void was fragmented. We have only considered detected voids in order to avoid underestimating the number of fragments; voids with 0 fragments were not considered for this table.

Table 5 summarizes the average recall and error rates including their standard deviation when circular voids are recovered from point sets of size $n = 5\,000$, $n = 10\,000$ and $n = 50\,000$ and Table 6 shows the same information for non-convex voids. The table also specifies which discarding criterion was used to eliminate too small terminal-edge regions by the end of the *Building subvoids* step: the minimum terminal-edge e_{min} or the minimum area a_s.

Fig. 9 displays the error rate for the convex shapes shown in Fig. 8(a). Fig. 10 shows the recall and error rates for the non-convex voids shown in Fig. 8(b), both for $n = 10\,000$. The recall rate for circular voids was not included as it was always either 1 or 0 depending on whether two different voids were joined into one or not. The recall and error rates are shown for each void and joining void criteria.

6. Analysis and discussion

In this section, we first analyze the recall and error rates for convex and non-convex underdense regions, and determine which joining criterion is more effective for the recovery of either type of void shape. Secondly, we discuss how sensitive the results are to changes in the threshold values to discard false subvoids. Finally, we explore on what happens when the artificial voids are not absolute (some points inside) and on the solutions we plan to implement.

Table 3
Number of subvoids remaining after applying the edge length filter e_{min}, the joining criteria for recovering each one of the artificial non-convex shapes shown in Fig. 8(b) and the final filter a_v, to keep only the voids with significant area. The criterion labels are: N: *None*, A: *Arc*, F: *Frontier*, S: *Second-longest-edge*, E: *Frontier-edge*. The first column shows the void identification, the next five columns show the fragmentation on low background point density $\rho_B = 1/800$ ($n = 5000$), the second five on middle background point density $\rho_B = 1/400$ ($n = 10\,000$) and the last five, on high background point density $\rho_B = 1/80$ ($n = 50\,000$).

Void	Joining criterion														
#	N	A	F	S	E	N	A	F	S	E	N	A	F	S	E
0	1	1	1	1	1	3	2	1	1	1	5	4	2	1	1
1	6	5	2	2	2	8	5	2	2	1	9	5	2	1	1
2	3	2	2	3	2	3	1	1	1	1	2	1	1	1	1
3	4	3	1	1	1	2	2	1	1	1	3	2	2	1	1
4	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1
5	1	1	1	1	1	5	3	1	1	1	6	5	1	1	1
6	2	2	1	1	1	2	2	1	1	1	4	4	3	1	1
7	4	3	1	3	2	3	3	1	2	2	5	3	1	1	1
8	3	3	2	2	3	1	1	1	1	1	3	3	2	1	1
9	2	2	2	1	2	1	1	1	1	1	1	1	1	1	1
10	2	2	1	1	1	3	2	1	1	1	4	4	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	2	2	1	2	1	5	3	1	1	1
14	4	4	2	2	3	1	1	1	1	1	2	2	1	1	1
15	5	2	1	2	1	4	4	2	2	3	15	11	4	1	1
16	2	2	1	0	1	2	1	1	1	1	4	3	1	1	1
17	2	1	0	1	2	3	1	0	0	1	4	3	1	2	1
18	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1
19	2	2	1	1	1	2	2	1	0	1	2	2	1	1	1
	$n =$ 5000					$n =$ 10 000					$n =$ 50 000				

Table 4
Average number of void fragments remaining after applying each joining criteria. Filter represents the discarding criterion (a_s or e_{min}) used after the first algorithm step (*Building subvoids*) in order to eliminate too small subvoids before the joining step is applied. A value equal to 1.0 indicates zero fragmentation.

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

		Joining criterion				
		None	Arc	Frontier	2nd longest-edge	Frontier-edge
Filter	a_s	1.84	1.60	1.63	1.55	1.10
	e_{min}	2.50	1.90	1.11	1.22	1.15

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

		Joining Criterion				
		None	Arc	Frontier	2nd longest-edge	Frontier-edge
Filter	a_s	1.95	1.85	1.25	1.05	1.00
	e_{min}	3.90	3.00	1.45	1.05	1.00

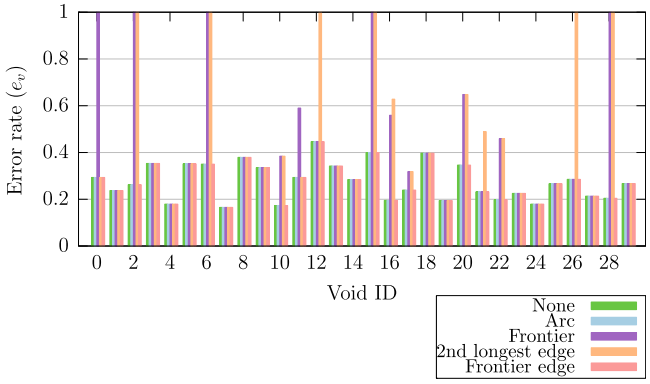


Fig. 9. Error rate after applying each joining criterion over circular voids with $n = 10\,000$ and subvoid filter e_{min}. The horizontal-axis represents the void ID (see Fig. 8(a)) and the vertical-axis the error rate. An error rate equal to 0 means that the whole retrieved void area is included in the original void and an error rate close to 1 means that the most of the retrieved area falls out the original void. The green bar represents the error rate for the voids recovered by the algorithm without applying any joining criterion (None). Arc and Frontier-edge criteria (light blue and pink bars, respectively) generate a similar error rate to the None criterion. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Independent on the void shape, the areas of well-retrieved voids are usually larger than the ones defined for the simulated voids. This behavior can be seen in Figs. 11(a) and 11(f). It is not surprising that for smaller values of n (lower point density) the error rate increases because on that condition, the width of voids in some sections becomes comparable to the average distance between points.

6.1. Retrieval of convex artificial voids

All the convex voids in the artificial data, in fact adopted circular, are successfully retrieved by DELFIN with no need to resort to joining criteria (pure DELFIN). Thus, just the *Finding subvoids* step is the choice when the voids are known a priori to be convex. However, in general the geometrical nature of the voids in a point set is not known. In this section we address the following question. If the void shapes in a point set are actually convex, and they are blindly searched, which of the joining criteria will approach best the result of the pure DELFIN algorithm.

The results of the application of the *Joining subvoids* step show that the recall rate drops considerably when applying the *Frontier* and *Second-longest-edge* criteria (see Table 5). A way to understand this behavior is the following: if subvoids are surrounded by very low point-density regions, these may be incorrectly identified also as subvoids and the just mentioned criteria have a tendency to overjoin neighboring voids. In case of the *Second-longest-edge* criterion, the identification of false neighboring subvoids is also possible but for higher background density distributions. This diagnostic is corroborated in Fig. 9 where the overdetection rate is shown for each the circular voids drawn in Fig. 8(a). The voids identified as 2, 6, 12, 15, 26 and 28 were missed by the *second longest-edge* criterion (their error rate is 1). This is due to the fact that subvoid regions matching these voids were joined to several adjacent regions and discarded as a result.

Table 5
Circular voids rates. Recall rate (a) and error rate (b) including their standard deviation. The n column corresponds to the size of the point set: $n = 5000$, $n = 10\,000$ and $n = 50\,000$. The *Filter* column specifies which discarding criterion (a_s or e_{min}) was used to eliminate too small terminal-edge regions at the end of the *Building subvoids* step. The rest of the columns summarizes the results of applying each one of the joining criteria. An average recall rate equal to 1 means that all voids were recovered.

(a) Circular voids: recall rate						
n	Filter	Joining criterion				
		None	Arc	Frontier	2nd l. edge	Frn. edge
5 000	a_s	1.00 ± 0.00	1.00 ± 0.00	0.63 ± 0.49	0.57 ± 0.50	0.97 ± 0.18
	e_{min}	1.00 ± 0.00	1.00 ± 0.00	0.67 ± 0.48	0.57 ± 0.50	0.97 ± 0.18
10 000	a_s	1.00 ± 0.00	1.00 ± 0.00	0.80 ± 0.41	0.73 ± 0.45	1.00 ± 0.00
	e_{min}	1.00 ± 0.00	1.00 ± 0.00	0.83 ± 0.38	0.80 ± 0.41	1.00 ± 0.00
50 000	a_s	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.87 ± 0.35	1.00 ± 0.00
	e_{min}	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.90 ± 0.31	1.00 ± 0.00
(b) Circular voids: error rate						
n	Filter	Joining criterion				
		None	Arc	Frontier	2nd l. edge	Frn. edge
5 000	a_s	0.35 ± 0.08	0.35 ± 0.08	0.67 ± 0.28	0.73 ± 0.29	0.38 ± 0.15
	e_{min}	0.35 ± 0.08	0.35 ± 0.08	0.64 ± 0.29	0.71 ± 0.30	0.38 ± 0.15
10 000	a_s	0.28 ± 0.08	0.28 ± 0.08	0.49 ± 0.29	0.55 ± 0.30	0.28 ± 0.08
	e_{min}	0.28 ± 0.08	0.28 ± 0.08	0.44 ± 0.28	0.46 ± 0.30	0.28 ± 0.08
50 000	a_s	0.14 ± 0.03	0.14 ± 0.03	0.14 ± 0.04	0.31 ± 0.29	0.14 ± 0.03
	e_{min}	0.14 ± 0.03	0.14 ± 0.03	0.14 ± 0.03	0.25 ± 0.27	0.14 ± 0.03

Table 6
Non-convex void rates. Recall rate (a) and error rate (b) including their standard deviation. The n column shows the size if the point set: $n = 5000$, $n = 10\,000$ and $n = 50\,000$. The *Filter* column specifies which discarding criterion (a_s or e_{min}) was used to eliminate too small terminal-edge regions at the end of the *Building subvoids* step. Each one of the remaining columns show the average recall and error rates with their standard deviation after applying the *Joining subvoids* step with the specified joining criterion.

(a) Non-convex voids: recall rate						
n	Filter	Joining criterion				
		None	Arc	Frontier	2nd l. edge	Frn. edge
5 000	a_s	0.73 ± 0.26	0.82 ± 0.21	0.83 ± 0.32	0.69 ± 0.38	0.87 ± 0.21
	e_{min}	0.73 ± 0.26	0.81 ± 0.23	0.89 ± 0.26	0.79 ± 0.31	0.89 ± 0.20
10 000	a_s	0.66 ± 0.29	0.77 ± 0.27	0.88 ± 0.26	0.74 ± 0.37	0.90 ± 0.20
	e_{min}	0.67 ± 0.28	0.76 ± 0.27	0.88 ± 0.24	0.79 ± 0.31	0.91 ± 0.15
50 000	a_s	0.61 ± 0.30	0.69 ± 0.30	0.95 ± 0.07	0.96 ± 0.14	0.99 ± 0.03
	e_{min}	0.58 ± 0.34	0.67 ± 0.32	0.95 ± 0.07	0.97 ± 0.09	0.98 ± 0.04
(b) Non-convex voids: error rate						
n	Filter	Joining criterion				
		None	Arc	Frontier	2nd l. edge	Frn. edge
5 000	a_s	0.41 ± 0.10	0.43 ± 0.11	0.58 ± 0.18	0.61 ± 0.20	0.44 ± 0.09
	e_{min}	0.41 ± 0.10	0.41 ± 0.10	0.50 ± 0.15	0.50 ± 0.15	0.44 ± 0.08
10 000	a_s	0.32 ± 0.17	0.35 ± 0.16	0.45 ± 0.18	0.55 ± 0.23	0.35 ± 0.08
	e_{min}	0.31 ± 0.10	0.32 ± 0.10	0.38 ± 0.17	0.44 ± 0.22	0.34 ± 0.08
50 000	a_s	0.17 ± 0.04	0.19 ± 0.05	0.20 ± 0.06	0.27 ± 0.07	0.21 ± 0.05
	e_{min}	0.17 ± 0.06	0.19 ± 0.06	0.20 ± 0.04	0.20 ± 0.05	0.20 ± 0.05

In contrast, the *Frontier-edge* and *Arc* criteria generate similar overdetection rates than the pure DELFIN (see again Fig. 9). More precisely, for higher point density distributions of the artificial data all voids were recovered and the error rates comparable to those obtained by the pure DELFIN.

6.2. Retrieval of non-convex artificial voids

The picture, however, is different in the case of non-convex voids. Table 4 shows the average number of void fragments associated with the polygonal voids shown in Fig. 8(b) for $n = 10\,000$ and $n = 50\,000$. It can be observed that the use of no criterion always retrieves fragmented voids independent of the background point density, and in average, the *Frontier-edge* criterion seems to be the most effective. This fact is corroborated in Table 6 since for the *Frontier-edge* criterion, the average of the recall rates is higher than for the other criteria. Additionally, the average of the overdetection rates is generally lower than the ones obtained after applying other criteria.

Depending on each void shape, the peculiarities of each void joining criterion come to light. This fact can be observed in Fig. 10, where the recall and error rates are shown for each non-convex polygonal void drawn in Fig. 8(b), and in Fig. 11, in particular for the polygonal void labeled as 1.

The *Arc* criterion is usually effective to join voids detected as two subvoids, but it fails to connect the subvoids of elongated figures as shown in Fig. 11(c). This criterion did not work in this case and the void was recognized almost in the same way as if no criterion was used (Fig. 11(b)). It can also be observed in Fig. 10 that the recall rate was small for the voids labeled as 1 and 5.

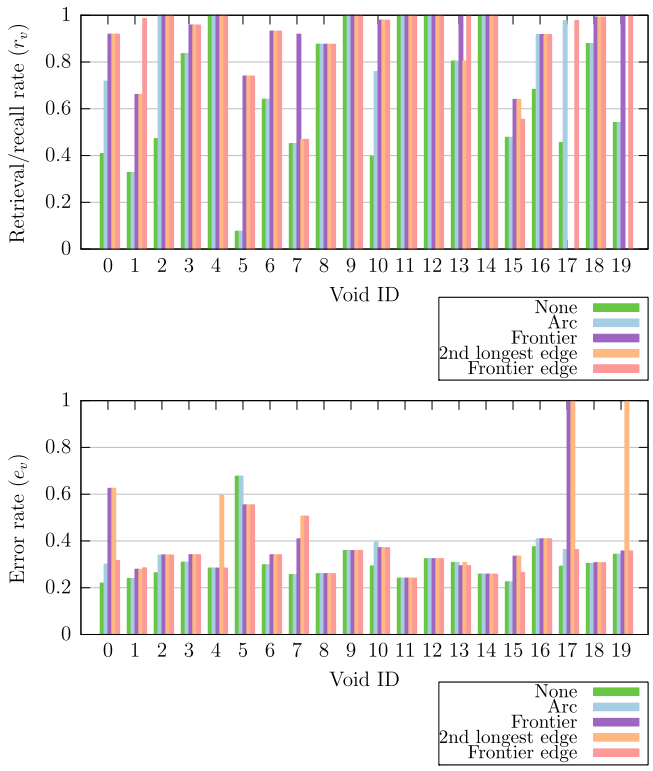


Fig. 10. Retrieval and error rates after applying each joining criterion over non-convex voids with $n = 10\,000$ and subvoid filter e_{min}. The top and bottom figures respectively show the retrieval rate and the error rate for each void in the vertical-axis. In both figures, the horizontal-axis represents the void ID (Fig. 8(b)). A retrieval rate greater than 0, for example 0.7, means that 70% of the void area was recovered, so a retrieval rate equal to 1 means that it was completely recovered. This is the case of void ID = 9. An error rate close to 1 means that the most of the recovered area does not belong to the artificial void. For void ID = 9, at least 60% of the recovered void corresponds to the expected void. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The *Frontier* criterion has the void shape in consideration, and thus usually obtains a recall rate higher than the *Arc* criterion. However, it shares the inability to join together voids with an elongated shape and its heavy dependence on the order in which fragments are joined. This can be observed in Fig. 11(d).

The *Second longest-edge* criterion joined several terminal-edge regions and obtained quite high recall rates, but also relatively high overdetection rates. It can also be observed that it is also highly dependent on shape. In particular, it worked worse than the *Frontier* criterion on the void shown in Fig. 11 and on bifurcations such as the ones contained in the polygonal voids labeled as 17, 7 and 15 as shown in Fig. 10.

Frontier-edge was the unique criterion that allowed us to recover all polygonal voids on medium and high background point density distribution. This can be observed in Fig. 11(f) for a specific polygonal void and in Table 6 for all data sets. It can be observed in Fig. 10 that the lowest recall rates for this criterion were around 0.5, i.e., half of the void area was recovered. The worst results were obtained for the polygonal voids labeled as 7 and 17. Both were recognized as subvoids in the *Building subvoids* step and these subvoids were not joined in the *Joining subvoids* step because they share too small frontier-edges.

6.3. Discarding subvoid criteria

Our experiments showed that the use of a threshold value associated to the terminal-edge (e_{min}) of a subvoid to discard small terminal-edge regions instead of a threshold value associated to the area of the terminal-edge region (a_s) reduces: (i) the amount of false voids specially in the lowest background point density data sets and (ii) the risk of removing central subvoids of small areas but large enough terminal-edges.

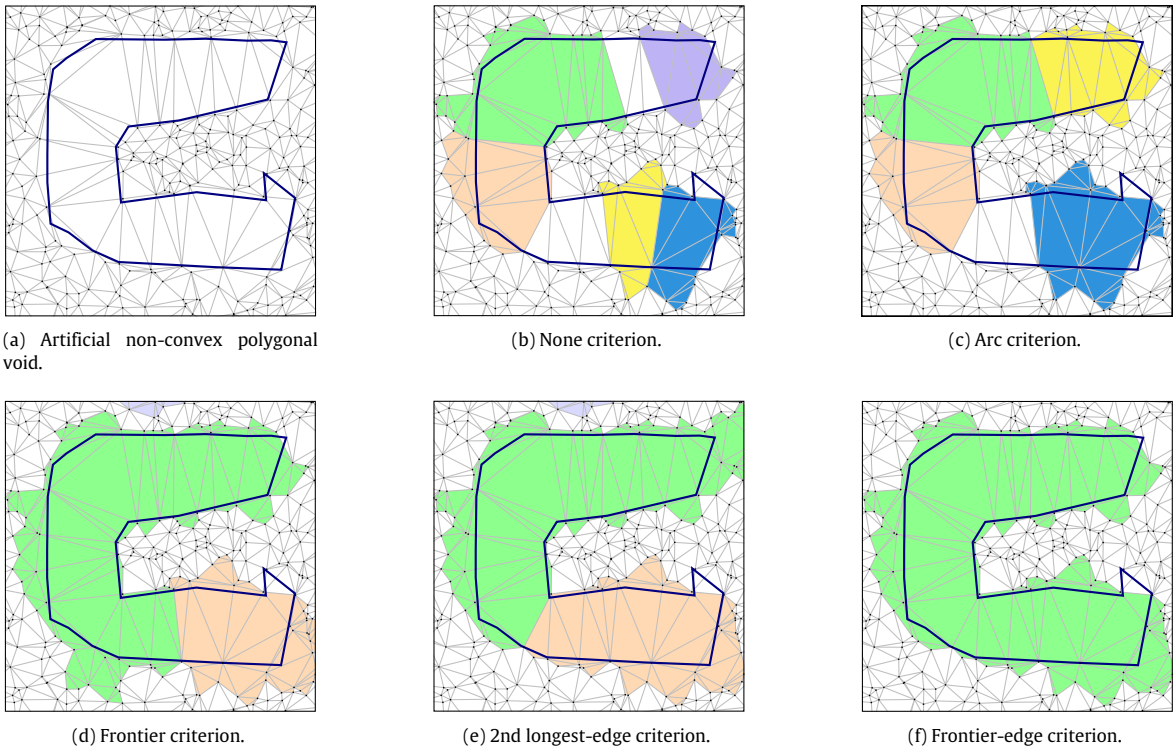
In order to illustrate the previous statement, we present two figures that show each of the identified subvoids by the *building subvoids* algorithm, one ordered by the length of its terminal-edge (Fig. 12) and the other by the area of its terminal-edge region (Fig. 13). Each X symbol represents an identified subvoid; they are scattered according to their terminal-edge length in Fig. 12 and the terminal-edge region area in Fig. 13, and whether they are part of an artificial void (true) or not (false). These figures suggest that terminal-edge length values is a better classifier for subvoids than terminal-edge region areas, thus we would recommend using the former to filter the relevant subvoids obtained by DELFIN. Moreover, the value e_{min} may be changed without affecting the results by moving it in ranges of [79.97, 80.08], [63.37, 67.48] and [33.03, 35.98] respectively for $n = 5000$, 10 000 and 50 000 (Fig. 12). In addition, since terminal-edges are processed in order, the algorithm has no need to compute all terminal-edge regions before moving forward to the next step and also no need to calculate any area.

6.4. Impact of isosceles and equilateral triangles

The proposed algorithm is based on the idea that the edges of any triangle can be classified as the longest-, second-longest- and shortest-edge. A priori, the three or two equal size edges of an equilateral or isosceles triangle, respectively, can become any label. This fact depends on the order the triangle edges are processed by the *Finding subvoids* algorithm. For an equilateral triangle or an isosceles triangle, where the two equal size edges are the largest ones, there are two cases: (a) the triangle is one of the two terminal triangles. Then its longest-edge will be the edge shared with the neighbor terminal triangle. (b) The triangle is part of the set of neighbor triangles of a terminal-edge region that is being built. The edge shared with the terminal-edge region will be considered the longest-edge and this triangle will be joined to the current region. If there are two shared edges, any one will be chosen. Note that in the case of an isosceles triangle, the triangle is joined to the current region only when the shared edge is one of the largest edges following the algorithm strategy.

Depending on the order equal size edges of a triangle are processed, terminal-edge regions of different size can be generated. Can this arbitrary choice or order of the input triangles and edges fragment large empty regions into two small terminal-edge regions that will later not pass to the *Joining subvoids* step due to the threshold values of the discarding filters (a_s or e_{min})? Effectively, this arbitrary choice can have an impact if a_s, the metric that filters by subvoid area, is used to discard small-terminal edge regions. However, if the e_min discarding measure is used, there is no impact in the results, because only the edge length is checked. Small area terminal-edge region (that might contain some equilateral triangles) are not discarded if the terminal-edge length is larger than the chosen threshold value.

As we have discussed in Section 6.3 our experimental results show that e_{min} is more appropriate than a_s not only for a triangulation with isosceles and equilateral triangles, but also for any triangulation. Since the triangulation is Delaunay, each edge has an empty circumcircle that passed through its endpoints, then if the distance between the endpoints is significant, the area of the empty circumcircle can be part of a larger empty region.



(a) Artificial non-convex polygonal void.
(b) None criterion.
(c) Arc criterion.
(d) Frontier criterion.
(e) 2nd longest-edge criterion.
(f) Frontier-edge criterion.

Fig. 11. The DELFIN criteria applied to recover the artificial non-convex shape ID = 1. (a) Triangulation under the non-convex shape. (b) Seven terminal-edge regions were found associated to the non-convex shape ID = 1, all passed the a_s discarding filter, no joining criterion was applied and finally, only five of them passed the a_v filter, the colored ones. (c) The Arc criterion built two larger empty regions by the union of two neighbor terminal-edge regions. The new regions are the yellow and the blue ones. The colored regions passed the a_v filter, and the non-convex void is still fragmented in four parts. (d) The Frontier and (e) the 2nd longest-edge criteria generated a similar result: the seven terminal-edge regions are joined into two larger regions, and both passed the a_v filter. The non-convex shape was fragmented in two parts. (f) The Frontier-edge criterion put together the seven terminal-edge regions in one unique large region. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

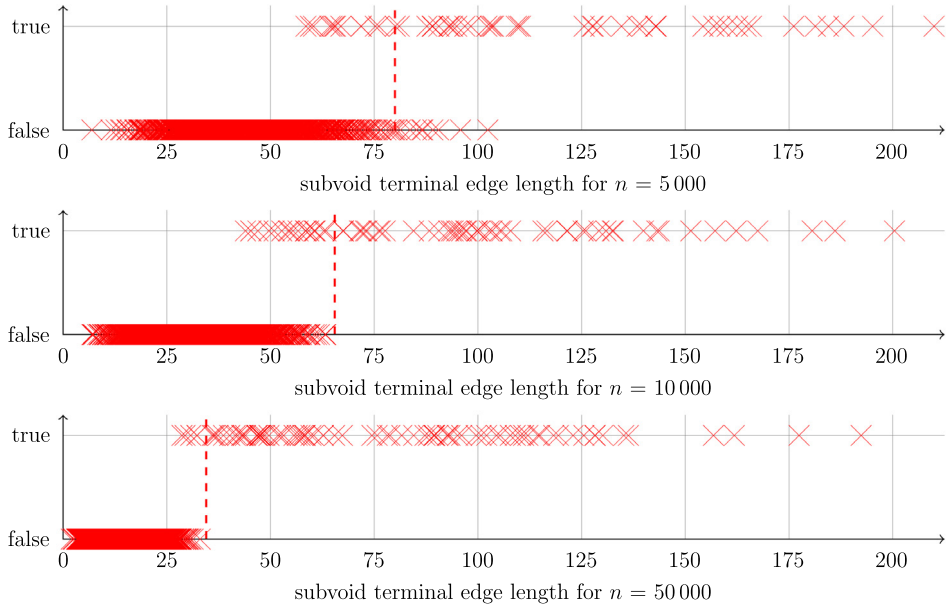


Fig. 12. False and true subvoid positives detected for $n = 5000$ (top graph), 10 000 (medium graph) and 50 000 (bottom graph) as a function of terminal-edge length. Each X symbol represents a subvoid identified by the *Finding subvoids* step: the horizontal-axis indicates its terminal-edge length, while the vertical-axis indicates whether a subvoid belongs to one of the generated artificial voids or not. Terminal edge length (e_{min}) is a relatively good measure of subvoid significance, because it can be used to clearly separate false and true subvoids as shown by the dashed red line in each scatterplot. For example, from the bottom graph, a good e_{min} value could be around 30.

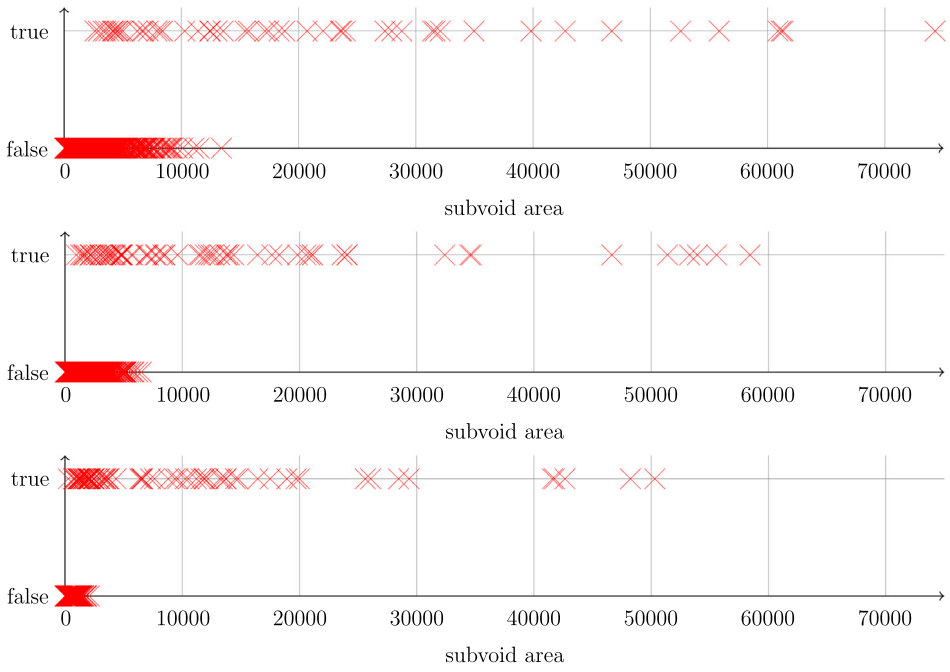


Fig. 13. False and true subvoid positives detected for $n = 5000$ (top graph), 10 000 (medium graph) and 50 000 (bottom graph) as a function of area. Each X symbol represents a subvoid identified by the *Building subvoids* step: the horizontal-axis indicates its area, while the vertical-axis indicates whether that subvoid belongs to one of the generated artificial voids or not. Subvoid area is a relatively good measure of subvoid significance but may underrepresent highly fragmented voids, such as those with irregularities or eccentricities, which are present on our experiment. There are many terminal-edge regions with small area that belong to a larger empty region. a_s is then not as effective as e_{min} to filter false subvoids.

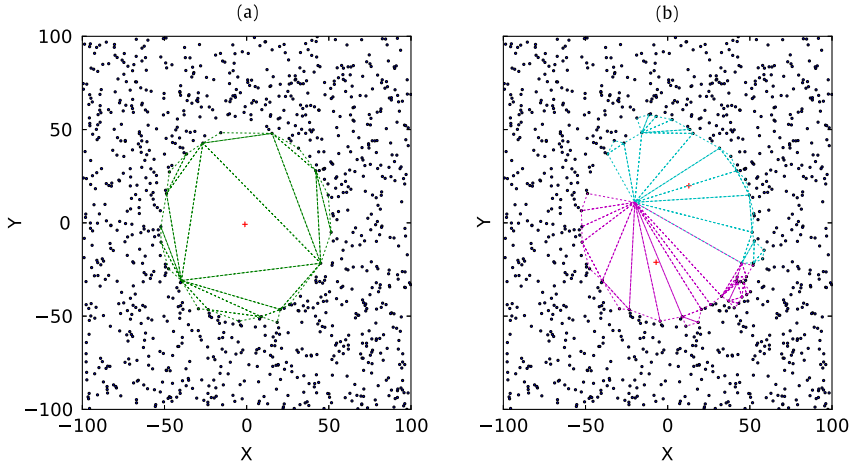


Fig. 14. (a) The void found inside a set of points. (b) The same set of point as (a) but with a point placed inside the void. Two distinct adjacent voids are found. The crosses are the void centroids. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.5. Non-empty voids

Cosmological voids are indeed not completely empty, so one question is how the algorithm could handle this situation. In Fig. 14 we illustrate the effect of placing a point inside a void. What actually happens is a fragmentation of the void into smaller adjacent subvoids, analogous to results encountered with non-convex empty polygons. Voids are fragmented into smaller adjacent components. One way to deal with this problem is the use of the *third neighbor criterion* introduced by some void finding algorithms (El-Ad and Piran, 1997; Hoyle and Vogeley, 2002; Foster and Nelson, 2009). It consists on the computation for each point of the distance to its third nearest neighbor. Then the mean l_3 and the standard deviation σ_3 of these distances are calculated. Every point whose third neighbor distance is greater than $l_3 + \lambda\sigma_3$ are not considered as part of the initial point set to search for voids. For the void search, λ is a constant that differs for every algorithm, typically 1.5 or 2.0. In this approach, isolated points that lie in the interior of a void do not interfere with the void search. In our preliminary 3D version, we have already included this strategy.

7. Conclusions

We have presented a new algorithm called DELFIN to find large empty spaces (voids) in planar point sets. The algorithm was

divided in two steps: (i) The *Building subvoids* step generates a Delaunay triangulation of the input points, sorts terminal-edges from the largest to the smallest one and builds terminal-edge regions around each of them. Each region is a subvoid candidate, and (ii) the *Joining subvoids* step joins subvoids into larger voids according to a specified subvoid joining criterion. Four joining criteria were implemented and evaluated: *Arc*, *Frontier*, *Second longest-edge* and *Frontier-edge*.

The effectiveness and robustness of DELFIN were evaluated on artificial data sets including both convex and non-convex void shapes. The experimental evaluation of the four joining criteria allowed us to choose the *Frontier-edge* criterion as the most effective one: it attained higher recall rates and lower error rates and recovered both convex and non-convex artificial voids. But for applications with underdense regions with other shapes or a subset of the shapes shown here, the user could use some other criteria presented in this work.

The experimental evaluation also showed us that DELFIN is more robust if the terminal-edge length instead of the terminal-edge region area is used to discard small terminal-edge regions or subvoid candidates before the *Joining subvoids* step. To our knowledge there is no other algorithm which looks for this kind of voids in planar point sets; that is why we do not present an empirical comparison with other 2D algorithms.

Some improvements to DELFIN include (i) the use of the union-find data structure to improve the time performance of both the *Building subvoids* and *Joining subvoids* steps and (ii) pruning the smallest triangles of terminal-edge regions in the *Building subvoids* step. We did not include (ii) in this first implementation because we wanted to keep the number of parameters the user has to fix as low as possible.

Currently, we are testing and evaluating the performance, effectiveness and robustness of a 3D extension of DELFIN over both artificial and real data sets (galaxy surveys). We are using the Frontier-edge as joining subvoid criterion and terminal-edge length as discarding criterion. In addition, since there are several 3D algorithms that have public catalogs of recovered voids, we are able to compare how good our void list intersects these catalogs.

Acknowledgments

The authors of this project thank the support of ENL009/15, VID2015, University of Chile, and Anillo Project Number ACT1122 financed by Conicyt.

References

Alonso, R., 2016. A Delaunay Tessellation Based Void Finder Algorithm (Ph.D. thesis). Department pf Computer Science, FCFM, University of Chile, master thesis in Computer Science.

Balogh, J., González-Aguilar, H., Salazar, G., 2013. Large convex holes in random point sets. Comput. Geom. 46 (6), 725–733. doi:10.1016/j.comgeo.2012.11.004.

Barber, C.B., Dobkin, D.P., Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. 22 (4), 469–483. doi:10.1145/235815.235821.

Delaunay, B., 1934. Sur la sphère vide. Bull. Acad. Sci. USSR(VII) 793–800. http://mi.mathnet.ru/eng/izv4937.

Edelsbrunner, H., Kirkpatrick, D.G., Seidel, R., 1983. On the shape of a set of points in the plane. IEEE Trans. Inform. Theory 29 (4), 551–558. doi:10.1109/TIT.1983.1056714.

El-Ad, H., Piran, T., 1997. Voids in the large-scale structure. Agron. J. 491, 421. doi:10.1086/304973.

Foster, C., Nelson, L.A., 2009. The size, shape, and orientation of cosmological voids in the sloan digital sky survey. Agron. J. 699, 1252–1260. arXiv:0904.4721.

Hervías, C., Hitschfeld-Kahler, N., Campusano, L.E., Font, G., 2013. On finding large polygonal voids using delaunay triangulation: The case of planar point sets. In: Proceedings of the 22nd International Meshing Roundtable, IMR 2013, October 13-16, 2013, Orlando, FL, USA, pp. 275–292. doi:10.1007/978-3-319-02335-9_16.

Hoyle, F., Vogeley, M.S., 2002. Voids in the point source catalogue survey and the updated zwicky catalog. Agron. J. 566, 641–651. doi:10.1086/338340.

Manning, C.D., Raghavan, P., Schütze, H., 2008. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA, ISBN: 0521865719.

Nandy, S.C., Bhattacharya, B.B., 2003. On finding an empty staircase polygon of largest area (width) in a planar point-set. Comput. Geom. Theory Appl. 26 (2), 143–171. doi:10.1016/S0925-7721(03)00015-4.

Neyrinck, M.C., 2008. ZOBOV: a parameter-free void-finding algorithm. Mon. Not. R. Astron. Soc. 386, 2101–2109. doi:10.1111/j.1365-2966.2008.13180.x. arXiv:0712.3049.

O'Rourke, J., 1998. Computational Geometry in C, second ed. Cambridge University Press, New York, NY, USA, ISBN: 0521640105.

Pinchasi, R., Radoicic, R., Sharir, M., 2006. On empty convex polygons in a planar point set. J. Comb. Theory, Ser. A 113 (3), 385–419. doi:10.1016/j.jcta.2005.03.007.

Platen, E., van de Weygaert, R., Jones, B.J.T., 2007. A cosmic watershed: the WVF void detection technique. Mon. Not. R. Astron. Soc. 380, 551–570. arXiv:0706.2788.

Rivara, M.C., 1997. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. Internat. J. Numer. Methods Engrg. 40, 3313–3324. DOI: 10.1002/(SICI)1097-0207(19970930)40:18<3313::AID-NME214>3.0.CO;2-#.

Rivara, M.C., Hitschfeld, N., Simpson, R.B., 2001. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. Comput. Aided Des. 33, 263–277. doi:10.1016/S0010-4485(00)00125-1.

Voronoi, G., 1908. Nouvelle application des paramètres continus a la théorie des formes quadratiques. J. Reine Angew. Math. 134, 198–287. https://eudml.org/doc/149291.

Way, M., Gazis, P., Scargle, J.D., 2015. Structure in the 3d galaxy distribution: Ii. voids and watersheds of local maxima and minima. Astrophys. J. 799 (1), 95. arXiv:1406.6111.