Inverting Dirichlet Tessellations

Frederic Paik Schoenberg[1] Thomas Ferguson[1] Cheng Li[2]

[1] Department of Statistics, University of California, Los Angeles, CA 90095–1554, USA.

[2] Department of Statistics, Harvard University, Cambridge, MA 02138, USA.

Corresponding author: Frederic Paik Schoenberg

phone: 310-794-5193

fax: 310-472-3984

email: frederic@stat.ucla.edu

Postal address: UCLA Dept. of Statistics

8142 Math-Science Building

Los Angeles, CA 90095–1554, USA.

Abstract

Given a collection of points in the plane, one may draw a cell around each point in such a way that each point's cell is the portion of the plane consisting of all locations closer to that point than to any of the other points. The resulting geometric figure is called a Dirichlet tessellation. An algorithm for obtaining the boundaries of the cells given the points was derived by Green and Sibson in 1978. Here, methods are described for obtaining the locations of the points, given only the cell boundaries.

Key words: tessellation, Dirichlet, Voronoi, inversion, detection, recognition.

1 Introduction.

Suppose n points $\{P_1, P_2, ..., P_n\}$ are distributed within some metric space $\mathcal{S}$. The *Voronoi tessellation* corresponding to these points is constructed simply by dividing up the space $\mathcal{S}$ into n distinct cells, where cell i consists of all locations in $\mathcal{S}$ which are closest to point P_i. That is, cell i is defined via:

$$T_i := \{X : d(X, P_i) < d(X, P_j)\, for\, j \neq i\}, \tag{1}$$

where d denotes a distance function (typically Euclidean distance). In the special case where the space S is the plane $\mathbf{R}^2$ or a portion thereof, the tessellation is called a *Dirichlet tessellation*. An illustration is provided in Figure 1. The following intuitive description is provided in [1]: "One might think of the points as being the locations of the lairs of competitive predators of equal strength; the region associated with each point is then the area available to the corresponding predator." Dirichlet tessellations have thus been applied in describing populations of species; see e.g. chapter 8.6 of [2].

Chapter 1.2 of [3] provides a survey of the historical development of Dirichlet tessellations, including early applications in diverse fields such as crystallography, ecology, meteorology, epidemiology, linguistics, economics, archaeology and astronomy, dating back to Descartes in the early 17th century. References to numerous further examples are given in chapter 10.2 of [4], including examples in forestry, communication theory, geology, metallography and zoology, and chapter 5.3 of [3] summarizes recent applications in a wide assortment of disciplines.

To distinguish the locations $\{P_1, \ldots, P_n\}$ that determine the tessellation from any other points, in what follows we will call these n points *spots*.

Given a collection of spots $\{P_1, \ldots P_n\}$ in the plane, the algorithm of Green and Sibson [1] (extended in [5] to the multidimensional case) provides a computationally efficient means of constructing the resulting Dirichlet tessellation. The problem addressed here is that of inverting the Green-Sibson algorithm. That is, given the segments demarking the edges of cells in a planar tessellation, we seek to obtain the spots $\{P_1, ..., P_n\}$.

Such an inversion may be useful for a variety of problems. In the territorial context, for instance, one may wish to estimate the locations of the lairs given the borders outlining each predator's territory. Another reason one may be interested in inverting a Dirichlet tessellation is for image compression. A broad range of piecewise constant images may be roughly approximated by Dirichlet tessellations; in fact several algorithms for image compression (e.g. [6,7,8]) begin by constructing a Dirichlet tessellation of a purely random (Poisson) process and subsequently refining the tessellation until it approaches the desired image. Given an image resembling a Dirichlet tessellation, with each cell colored a certain shade, a parsimonious means of compressing the image is to store merely the coordinates of the spots $\{P_1, ..., P_n\}$ that give rise to the tessellation, along with the colors associated with each spot.

The problem of inverting a tessellation known to be Dirichlet is closely related to that of determining whether a given tessellation is Dirichlet. The latter problem is called Dirichlet tessellation *recognition* or *detection* and is investigated in depth by Ash and Bolker [9], who list several properties characteristic of such tessellations. Evans and Jones [10] frame the problem of finding a Dirichlet tessellation (and the corresponding spots) closely approximating a given tessellation as essentially a regression problem and outline three algorithms for its solution. Unfortunately, as noted in [10], all three algorithms require the inversion of poorly-conditioned matrices and may thus be highly unstable. Like the method described in [10], related methods proposed in [11,12,13], are based solely on the observation that in a Dirichlet tessellation, the segment joining any two adjacent spots is perpendicularly bisected by the segment of the tessellation shared by the boundaries of their cells. Okabe et al. [3] propose repeated use of a result in [9] for finding the spot within each cell based on the angles the vertices of the cell make with adjacent vertices. Here, we explore methods involving both inspection of vertex angles and the perpendicular bisector property, with the goal of finding stable, efficient, and resistant methods for Dirichlet tessellation inversion.

The structure of this paper is as follows. In Section 2, basic geometric properties of Dirichlet tessellations are discussed. Section 3 describes algorithms for inverting Dirichlet tessellations. Section 4 reviews performance properties of different algorithms. Some concluding remarks and directions for further research are presented in Section 5.

2 Geometry of Dirichlet tessellations.

Numerous geometric facts about Dirichlet tessellations have been discovered. For instance, the following result is ubiquitous, and follows immediately from the definition of the Dirichlet tessellation.

Lemma 2.1. For any two adjacent cells containing spots P_i and P_j, respectively, the segment common to the boundaries of both cells is a perpendicular bisector of the segment $\overline{P_iP_j}$.

Examples of other basic geometric properties of Dirichlet tessellations are that each cell boundary is a convex polygon, each vertex is the circumcenter of the spots whose cells share the vertex, and the average number of segments per cell may not exceed 6. For derivations of these and other further geometric properties of the Dirichlet tessellation and its dual construct, the Delaunay triangulation, see chapters 2.3 and 2.4 of [3].

For a typical Dirichlet tessellation, each vertex is the intersection of exactly three of the tessellation's segments. However, this is not always the case. For example, if the spots giving rise to the tessellation form a complete rectangular lattice, then each vertex of the tessellation will be the intersection of four segments. More generally, if $k \geq 3$ spots lie on a circle centered at location X and if no other spots are in the interior of this circle, then the resulting Dirichlet tessellation will have a vertex at X where k segments intersect. Any vertex in a Dirichlet tessellation which is the intersection of more than three segments is called *degenerate*.

A key geometric fact facilitating inversion is provided in the following Theorem, versions of which appear in page 186 of [9] and page 67 of [3].

Theorem 2.2. Let T denote a cell in a planar Dirichlet tessellation. Let P_1 denote the spot in cell T. Suppose T has a nondegenerate vertex B, and let $\overline{AB}$ and $\overline{BC}$ denote two of the segments outlining T. Let $\overline{BD}$ be the third segment in the tessellation with an endpoint at B, and let E be any point in the interior of T such that E, B, and D are colinear. Then

$$\angle ABP_1 = \angle EBC, \tag{2}$$

and

$$\angle ABE = \angle P_1BC. \tag{3}$$

Proof.

Let P_2 be the orthogonal mirror image of P_1 across $\overline{AB}$, and let P_3 be the orthogonal mirror image of P_1 across $\overline{BC}$. From Lemma 2.1, P_2 and P_3 are the spots of cells adjacent to cell T.

Let θ_1, θ_2, and θ_3 denote the angles $\angle ABP_1$, $\angle P_1BE$, and $\angle EBC$, respectively, as in Figure 2. It follows directly from Lemma 2.1 that angle $\angle P_2BA = \theta_1$ and angle $\angle CBP_3 = \theta_2 + \theta_3$.

Since the vertex B is nondegenerate, P_2 and P_3 are spots of cells sharing the boundary segment $\overline{BD}$. Thus Lemma 2.1 implies that segment $\overline{BE}$ is a perpendicular bisector of the segment $\overline{P_2P_3}$ and hence angles $\angle P_2BE$ and $\angle EBP_3$ are equivalent. That is,

$$2\theta_1 + \theta_2 = 2\theta_3 + \theta_2, \tag{4}$$

which yields (2).

Adding $\angle P_1BE$ to both sides of (2) immediately establishes (3).

3 Algorithms.

3.1 Preliminaries

A number of software packages contain routines for performing certain basic tasks involving tessellations. A tessellation is typically stored as a list of vertex coordinates and its associated *contiguity lists*: lists which provide, for each vertex, the indices of the other vertices to which it is connected. Note that the cells containing the outermost spots of a tessellation are bounded not just by segments but also by rays extending infinitely in one direction. Such a ray is handled by storing both its starting point X and an arbitrary other point Y on the ray as vertices, where Y is labeled a "dummy" vertex and given no adjacency list. In the input to our programs described in Section 3.2 below, we simply require that the number of ordinary vertices and the number of dummy vertices be specified, and that the dummy vertices be placed at the end of the vertex list. In inverting the tessellation, the dummy vertices are considered degenerate as far as the spot determination algorithms in Section 3.2 are concerned.

The list of vertices and the contiguity lists comprise a condensed means of storing the tessellation; note that no listing of the n *cells* is provided. The algorithms proposed in what follows consider beginning with this information only. We note in passing that, given even less information, namely a list of segments and rays comprising the tessellation, the contiguity lists may readily be formed in an obvious manner. However, the formation of such lists cannot be performed in $O(n)$ operations, whereas the algorithms considered below make use of the contiguity lists to achieve $O(n)$ speed. Note also that our discussion does not apply to software programs that store tessellations by recording the geometry of the *dual* construction, the Delaunay tessellation, from which the geometry of the Dirichlet tessellation may be readily obtained (see e.g. [3]).

3.2 Inversion Algorithms

Several authors have proposed methods for inverting Dirichlet tessellations based on the perpendicular bisector property in Lemma 2.1. For instance, Evans and Jones [10] observe that application of Lemma 2.1 to every pair of adjacent spots yields a system of $2k$ equations in $2n$ unknowns, where n is the number of spots and k the number of edges in the tessellation, and suggest least-squares solutions (constrained and unconstrained). Hartvigsen [12] modifies this procedure in forming a polynomial-time algorithm for inverting Voronoi diagrams in R^d having vertices of arbitrary degree, and Adamatzky [13] discusses how an iterative procedure for finding the spots based on reflecting across tessellation segments may be implemented using a massively parallel algorithm and accelerated with the aid of a cellular automata processor.

As discussed in [10], solution of the linear equations implied by Lemma 2.1 by least squares involves the inversion of matrices that are poorly conditioned, hence numerical stability may be jeopardized. In order to increase stability and precision, it may be preferable to employ methods that use other potentially useful information about the configuration of vertices rather than rely solely on the direct application of the perpendicular bisector property of Lemma 2.1. In addition, the above procedures involve estimating the location of a given spot based on *all* the vertices in the tessellation, and proceed in such a way that errors in spot determinations can be expected to propagate throughout the inversion. But since tessellations are locally defined objects (i.e. a given cell is determined solely by nearby spots), methods for determining spots locally, rather than globally, may be desired. In particular, one might prefer an algorithm whose solution is resistant in the event that one (or a few) of the segments defining the tessellation boundary may be recorded in error.

In light of Theorem 2.2, the three segments sharing a vertex B (i.e. $\overline{AB}$, $\overline{BC}$, and $\overline{BD}$ in Figure 2) define a ray through the point B on which the spot P_1 of the cell bounded by $\overline{AB}$ and $\overline{BC}$ must fall. That is, given points A, B, C, and D as in Figure 2, one may find a line on which P_1 must lie as follows: i) Find any point E by extending the line segment $\overline{DB}$ in the direction away from D, as in Figure 2; ii) Compute the angle $\angle EBC = \theta_3 = \theta_1$; iii) Rotate the segment $\overline{AB}$ by the angle θ_1, keeping the point B fixed. Thus one constructs a ray through B making an angle θ_1 with $\overline{AB}$. By Theorem 2.2, P_1 must fall on this ray.

Similarly, if any other nondegenerate vertex V of the cell inhabited by P_1 exists, then the three segments sharing vertex V define a ray through V on which P_1 must lie. Hence P_1 is simply the intersection of these two rays.

This suggests the following naive algorithm, proposed in [3] for Dirichlet tessellations all of whose cells contain at least two nondegenerate vertices:

```
Algorithm A
Step 1. Determine the cells $C_1, C_2, \ldots, C_n$.
Step 2. For each cell $C_i$:
        a) Find any two nondegenerate vertices outlining $C_i$.
```

b) For each such vertex V, use the other two vertices connected to V to find the slope of the ray extending from V through the spot in C_i, as described above.

c) Find the intersection of these two rays.

Note that Step 1 may be achieved in $O(n)$ time quite simply as follows, using the fact that each cell must be convex. Beginning with a first vertex V_1 and one of its adjacent vertices V_2, find the vertex V_3 adjacent to V_2 such that the path $V_1 \to V_2 \to V_3$ proceeds clockwise. Continue to move clockwise around the cell until returning to vertex V_1. Repeat, beginning with each pair of adjacent vertices, constructing not only a list of vertices in each cell, but also simultaneously an array which lists, for each vertex, the indices of the cells it comprises. In order to avoid duplication, each time a pair of adjacent vertices is selected as a candidate for beginning a new cell, this latter array should be used to verify that the cell has not already been formed.

One shortcoming of Algorithm A is obvious: the requirement that each cell contain at least two nondegenerate vertices. Another is that if the two rays in a cell are perfectly parallel then step 2c will not result in a uniquely determined intersection point. Certainly such an event is highly atypical, and in any event a simple modification is to find an additional ray emanating from a different nondegenerate vertex in the cell, if one exists, should this problem arise. However, there is some apparent numerical instability in step 2c stemming from the fact that only two rays are used to determine the spot in each cell, particularly if these two rays happen to be *nearly* parallel, as is often the case when the two vertices selected in Step 2a are very close together. One would suspect that errors in the spot determination could be minimized by using all the available rays rather than just two. Hence an alternative is the following:

```
Algorithm B:
Step 1. Find the cells $C_1, C_2, \ldots, C_n$.
Step 2. For each cell $C_i$:
        a) Find all nondegenerate vertices outlining $C_i$.
        b) Find the slope of the ray associated with each such vertex.
        c) Find the intersections of every possible pair of the rays.
        d) Average these intersection points.
```

The reader may find it surprising that the spot location errors using Algorithm B are in fact typically considerably *larger* than for Algorithm A, a curious phenomenon which is explained in Section 4.2 below. Since this increase in error is attributable to the instability in intersecting certain select pairs of rays, one may modify Step 2d of Algorithm B by computing a *weighted* average of the intersection points, weighting each point according to an estimate of its stability, as in the following procedure.

```
Algorithm C:
Step 1. Find the cells $C_1, C_2, \ldots, C_n$.
Step 2. For each cell $C_i$:
        a) Find all nondegenerate vertices outlining $C_i$.
```

b) Find the slope of the ray associated with each such vertex.

c) Find the intersections $S_{k,l}$ of every possible pair (k,l) of the rays in cell C_i.

d) For each pair (k,l) of rays in cell C_i, estimate the stability of its intersection by perturbing the slopes of each of the rays by a small amount in either direction and seeing how much the intersection point changes. Record $\delta_{k,l} =$ the sum of the sizes of these changes.

e) Compute a weighted average of the intersection points, giving $S_{k,l}$ the weight $(\delta_{k,l})^{-1}/(\sum_{k',l'} \delta_{k',l'}^{-1})$.

Note that a potential alternative to algorithms B and C is to find the point minimizing some penalty function such as the sum of squared perpendicular distances to the rays. The (weighted) averaging in algorithms B and C is equivalent to finding the location minimizing the (weighted) sum of squared distances to the intersection points of the rays.

Algorithms A, B, and C are all entirely local; each cell is determined solely based on its own vertices and their neighboring vertices. The accuracy of the algorithms can potentially be improved by incorporating information from neighboring cells, e.g. by using the perpendicular bisector relation of Lemma 2.1. This suggests modifying Algorithms A, B, and C as follows, to form Algorithms A', B', and C'.

Algorithm X' (X = A, B, or C):

Perform Steps 1 and 2 of Algorithm X.

Step 3. For each cell C_i:

a) For each segment T outlining cell C_i, find the other cell C_j sharing this segment.

Obtain the estimate of the spot in cell C_j computed in Step 2 and find the mirror image of this spot estimate across the segment T.

b) Average the results from Step 3a together with the estimates from Step 2 to obtain a refined estimate of the spot in cell C_i.

We note in passing that several types of weighted averages are possible in Step 3b above, as well as in Step 2e of Algorithm C. One possibility which is particularly simple and which is shown in Section 4 to work well for each of the algorithms is to assign each of the spot estimates in Step 3a weight m_i^{-1}, where m_i is the total number of estimates for the spot in cell C_i from Steps 2 and 3a combined.

4 Implementation.

4.1 Availability.

Our programs (called *tessinverta, tessinverta2, tessinvertb, tessinvertb2, tessinvertc*, and *tessinvertc2*) are compiled C++ modules freely available at

http://www.stat.ucla.edu/∼frederic/papers/tess/ . In the same directory are the raw C++ code, and instructions detailing how properly to manipulate the input and output of the *tessinvert* programs. The programs take as input the list of vertices and their adjacency lists, as described in Section 3.1 above. One may easily manipulate the inputs to these programs into the required form given the output of a standard tessellation construction program such as the function *voronoi.mosaic* from the *tripack* library of the software package *R*; at the time of submission both *R* and *tripack* are freely available at http://cran.r-project.org/ .

4.2 Errors

Table 1. Log (base 10) of RMS errors of spot location estimates

n	A	A'	B	B'	C	C'
10	-13.6	-14.0	-13.6	-13.8	-13.9	-14.1
50	-12.2	-12.6	-12.1	-12.3	-12.8	-12.9
100	-11.5	-11.9	-11.5	-11.6	-12.3	-12.4
250	-10.6	-11.0	-10.7	-10.9	-11.6	-11.7
500	-9.94	-10.3	-10.1	-10.3	-11.1	-11.2
1000	-9.24	-9.62	-9.53	-9.67	-10.5	-10.7
2000	-8.52	-8.89	-8.97	-9.09	-10.1	-10.3
3000	-8.13	-8.52	-8.54	-8.66	-9.88	-10.0
4000	-7.94	-8.29	-8.32	-8.44	-9.71	-9.82
5000	-7.71	-8.08	-8.23	-8.33	-9.57	-9.69

Table 1 shows how the root-mean-squared (RMS) errors in the spot locations vary with the number of spots, n. Each row in Table 1 reports the logarithm of the RMS errors in spot locations over 1000 realizations each consisting of n spots generated using a uniform distribution on the square region $[0, \sqrt{n}] \times [0, \sqrt{n}]$. Thus all the simulations have on average one spot per unit area. For each simulation, the corresponding Dirichlet tessellation was constructed using the *voronoi.mosaic* function in *tripack*; this tessellation was then inverted using the algorithms of Section 3.2 and the resulting spot locations compared to the locations of the original spots. In Table 1, by "errors" we mean Euclidean distances between the original spots and the reconstructed spots. It should be noted that in Table 1, the errors in the inversion algorithms are confounded with those of the tessellation construction routine, and may thus be considered an upper bound on the errors in the tessellation inversion program.

For all the algorithms proposed in Section 3.2, the computational errors are quite small. In all our simulations there was no single location error larger than 10^{-5}. However, four main features of Table 1 are worth noting.

First, for each row (and in fact for nearly every single simulation) of Table 1, the errors decreased when the mirroring modification of Step 3 was used. In other words, the error in Algorithm X' $<$ the error in Algorithm X.

Second, the errors in Algorithm B are on average very comparable with (and often larger than) those in Algorithm A, and similarly for B' and A'. This result may be surprising, given that Algorithm B works by averaging over all spot estimates corresponding to all pairs of rays for each cell, whereas Algorithm A uses just a single pair of rays. However, it turns out the bulk of the RMS error is attributable to just a few cells, and among those in particular, the errors result from the use of pairs of rays that are very nearly parallel. Algorithm A uses the first two non-degenerate vertices found in each cell: these two vertices are typically contiguous, which means that often the rays stemming from them are rather similar in slope but never *too* similar. By contrast, Algorithm B uses all pairs of rays, including pairs of rays stemming from opposite sides of the cell, which are occasionally *very* nearly parallel. See Figure 3 below, for example. Intersecting these pairs of nearly parallel rays introduces some numerical instability, and the errors in spot estimates resulting from such pairs are often many orders of magnitude greater than typical estimates, hence the errors dominate when all the pairs are combined by simple averaging as in Algorithm B.

Third, it is apparent from Table 1 that for every value of n, Algorithm C' has the least error. The problem of parallel rays does not significantly affect the performance of Algorithms C and C', which give the estimates resulting from such pairs of rays very little weight when computing weighted averages of the estimates.

Fourth, for each of the methods in Table 1, the RMS error increases with n. This may seem curious, given that all the algorithms are local and that the average number of spots per unit area is the same for each simulation. Note however that the larger n is, the greater is the chance of observing a cell that is very long and thin, and the configuration of the outermost cells (which tend to have few non-degenerate vertices) also varies with n. Such cells appear to be responsible for the bulk of the errors.

4.3 Stability

As noted above, the errors in the inversion algorithms proposed in Section 3.2 are very small, especially for Algorithm C'. However, one may inquire about the size of the errors resulting when one of the vertices is recorded substantially in error.

In Section 3.2 it was claimed that because the algorithms proposed here use only local information for each cell, one might expect the resulting spot estimates to be more resistant to errors in a particular vertex location, compared to algorithms based on the perpendicular bisector property which estimate individual spots using information from distant cells.

To verify this, we examined the most recent of these algorithms, that of Adamatzky [13]. Adamatzky's algorithm estimates the location of the spot in an initial cell using an iterative algorithm. This initial spot is mirrored across the segments outlining its cell to determine the spots in neighboring cells, whose spots are mirrored across the segments outlining their cells, and the process repeats until all spots have been found. Unfortunately Adamatzky provides no guarantee that the iteration determining the initial spot will converge to arbitrary precision. Moreover, clearly there is no need to use an iterative algorithm to estimate the spot location of the initial cell; a technique based on angles such as that in Step 2 of Algorithm A is generally preferable. We consider an algorithm based on accurate determination of an initial cell spot (via Step 2 of Algorithm A) followed by mirroring this spot and subsequent spots over each segment of its cell boundary, and call this algorithm a modified-Adamatzky algorithm. The use of this algorithm allows us to investigate the propagation of errors in the Adamatzky algorithm from the mirroring step to determine subsequential spots.

Note that in the Adamatzky (and modified-Adamatzky) algorithms, not all the segments of the tessellation will be used to determine the spots. At each step in the mirroring process, cells with spots that have been determined are mirrored across the segments of their cells to determine subsequent spots. If two neighboring cells have each been determined via mirroring from *other* cells' spots, then the segment joining these two cells will not be used at all in the algorithm. Thus, if one perturbs a vertex of the Dirichlet tessellation input at random, occasionally this will have no effect at all on the output of the tessellation inversion routine, which is an attractive feature.

However, if the perturbed vertex *is* used by the routine, errors may propagate throughout an entire section of the tessellation, For instance, Figure 4 shows the errors in the modified-Adamatzky algorithm when a coordinate in a vertex in the initial cell is perturbed by .001. The perturbed vertex at $(11.7, 15.2)$ is marked with a large asterix. One sees that large errors result in a broad portion of the tessellation. By contrast, Algorithm C' appears to be quite resistant to such a perturbation, as indicated by Figure 5 which shows the errors resulting from Algorithm C' for the same input tessellation as that in Figure 4.

One may inquire about the RMS errors when Adamatzky's algorithm is used on a tessellation that has *not* been perturbed in this fashion. Adamatzky's algorithm requires that the user specify the precision of the initial spot estimate by setting the level at which the iteration used to determine the initial spot estimate should stop. Based on Table 1, errors in the modified-Adamatzky algorithm should be similar to those in the original Adamatzky algorithm when the precision threshold is on the order of the error sizes for Algorithm A in Table 1. Errors in the modified-Adamatzky algorithm are on par with Algorithm C'; hence by comparison the main advantage of C' is not its precision but its resistance to substantial errors in the input vertices.

4.4 Run times

All of the algorithms proposed in Section 3.2 are extremely fast, requiring just $o(n)$ observations, where n represents the number of spots to be determined. This $o(n)$ run time is confirmed by Figure 6 which shows the results for running the *tessinvert* programs on a Sun Ultra Enterprise 3500 at the UCLA Statistics Department. For each simulation, n points were distributed uniformly on a square of area n, as with the simulations described in Sections 4.2 and 4.3.

4.5 Degeneracies and conditions for invertibility.

Although Adamatzky claims (in Lemma 1 of [13]) that all cell spot locations of a Dirichlet tessellation may be determined uniquely if and only if the tessellation has at least one closed cell, this result is not correct. For instance, the left panel of Figure 7 shows an example of a Dirichlet tessellation all four cells of which are open, yet the spots can be uniquely determined since there are cells with more than one non-degenerate vertex, and the other cells' spots may be obtained by mirroring from adjacent cells. Conversely, in cases where every vertex in the tessellation is degenerate (e.g. the spots all lie on a lattice as in the right panel of Figure 7), the locations of the spots are indeterminate, despite the fact that a closed cell exists. In such events a sample collection of spots can typically be found by assigning an initial spot to an arbitrary location and proceeding from there by mirroring across segments of adjacent cells. A correct version of Adamatzky's statement is provided by Ash and Bowlker in Corollary 15 of [9]: If all vertices of a Dirichlet tessellation are non-degenerate and if more than one vertex exists, then the spots can be uniquely determined.

A requisite for Algorithms A, B, and C is that each cell contain multiple (i.e. at least two) non-degenerate vertices, since these algorithms are entirely local and do not even use neighboring cells' spot estimates in determining each cell's spot location. However, for Algorithms A', B', and C' this requirement can be relaxed: all that is required is that every cell should either have multiple non-degenerate vertices *or* have at least one neighboring cell which has multiple non-degenerate vertices. This condition was not violated in any of the simulations we ran. Note also that one way to handle degeneracies in the form of k-valent vertices for $k > 3$, is to enter them in the inputs as $k - 2$ distinct vertices all with the same coordinates, and provided their adjacency lists are suitably entered no major difficulties are presented in Algorithm C'. Cell spot estimates based on treating such vertices as non-degenerate will be highly unstable, but Algorithm C' treats them as such and gives them nearly no weight in computing the weighted average of spot estimates, hence such degenerate vertices do not present a real obstacle to Algorithm C' provided other, non-degenerate vertices exist nearby.

5 Summary and proposed extensions.

Algorithm C' introduced here is shown to be a stable routine for inverting a Dirichlet tessellation in the plane, in order to find the spots given only the segments outlining the cells of the tessellation. The algorithm is very fast and efficient, requiring only $O(n)$ computations and obtaining a high level of accuracy.

The present work may be extended in various ways, including generalizing the above procedure to the problems of inverting higher-dimensional and generalized Dirichlet/Voronoi tessellations, inversion of tessellations given different distance metrics, and tessellations in general metric spaces. Further, there are important classes of other types of tessellations for which stable and efficient methods of inversion do not appear to be currently available, including Johnson-Mehl tessellations and hyperplane tessellations which are described by Okabe et al. [3].

Acknowledgement.

Thanks to Jose Louis Hales-Garcia for technical support and anonymous referees for very instructive comments.

References.

[1] Green, P. and Sibson, R. (1978) Computing Dirichlet tessellations in the plane. *Comp. J.*, **21**, 168–173.

[2] Ripley, B. (1981) *Spatial Statistics.* Wiley, NY.

[3] Okabe, A. Boots, B. Sugihara, K. and Chiu, S. (2000) *Spatial Tessellations, 2nd ed.* Wiley, Chichester.

[4] Stoyan, D. Kendall, W. and Mecke, J. (1995) *Stochastic Geometry and its Applications, 2nd ed.* Wiley, Chichester.

[5] Bowyer, A. (1981) Computing Dirichlet tessellations. *Comp. J.*, **24**(2), 162–166.

[6] Rom, H. and Peleg, S. (1988) Image representation using Voronoi tessellation: adaptive and secure. *Proc. IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 5–9, pp. 125–146. Computer Society Press, Los Angeles.

[7] Chassery, J. M. and Montanvert, A. (1989) A segmentation method in a Voronoi diagram environment. *Proc. Sixth Scandinavian Conf. Image Analysis*, Oulu, Finland, June 19–22, pp. 408–415. Pattern Recognition Society of Finland, Oulu.

[8] Montanvert, A. Meer, P. and Rosenfeld, A. (1991) Hierarchical image analysis using irregular tessellations. *IEEE Trans. Pattern Analysis and Machine Intelligence*, **13**(4), 307–316.

[9] Ash, P. and Bowlker, E. (1985) Recognizing Dirichlet tessellations. Geometriae Dedicata, **19**, 175–206.

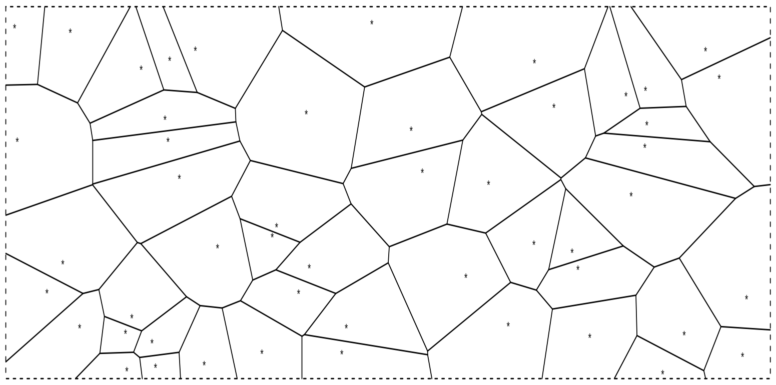
[10] Evans, D. and Jones, S. (1987) Detecting Voronoi (area-of-influence) polygons. *Mathematical Geology*, **19**(6), 523–537.

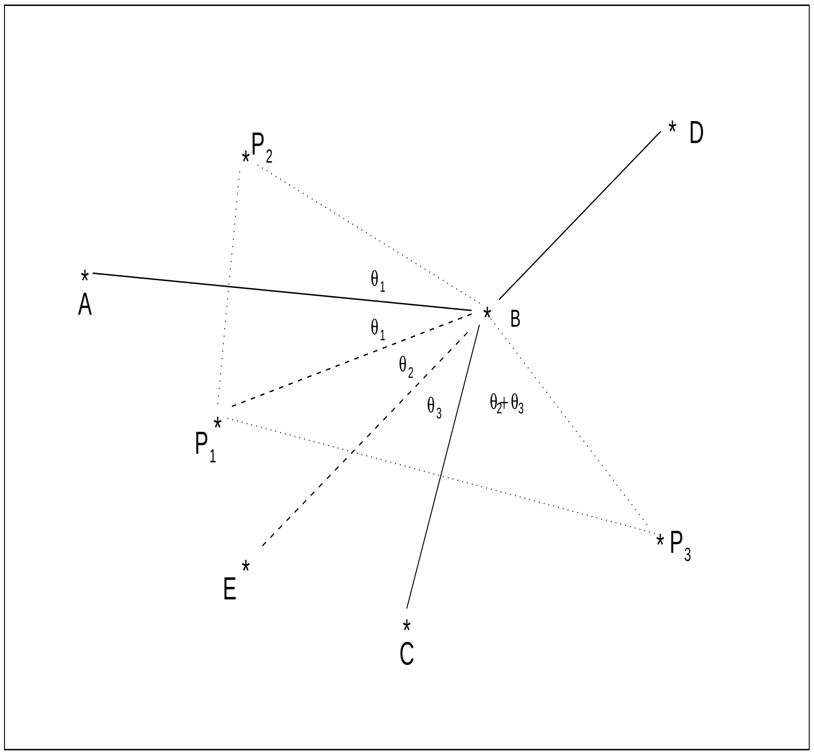
[11] Aurenhammer, F. (1987) Recognising polytopical cell complexes and constructing projection polyhedra. *J. Symbolic Computation*, **3**, 249-255.

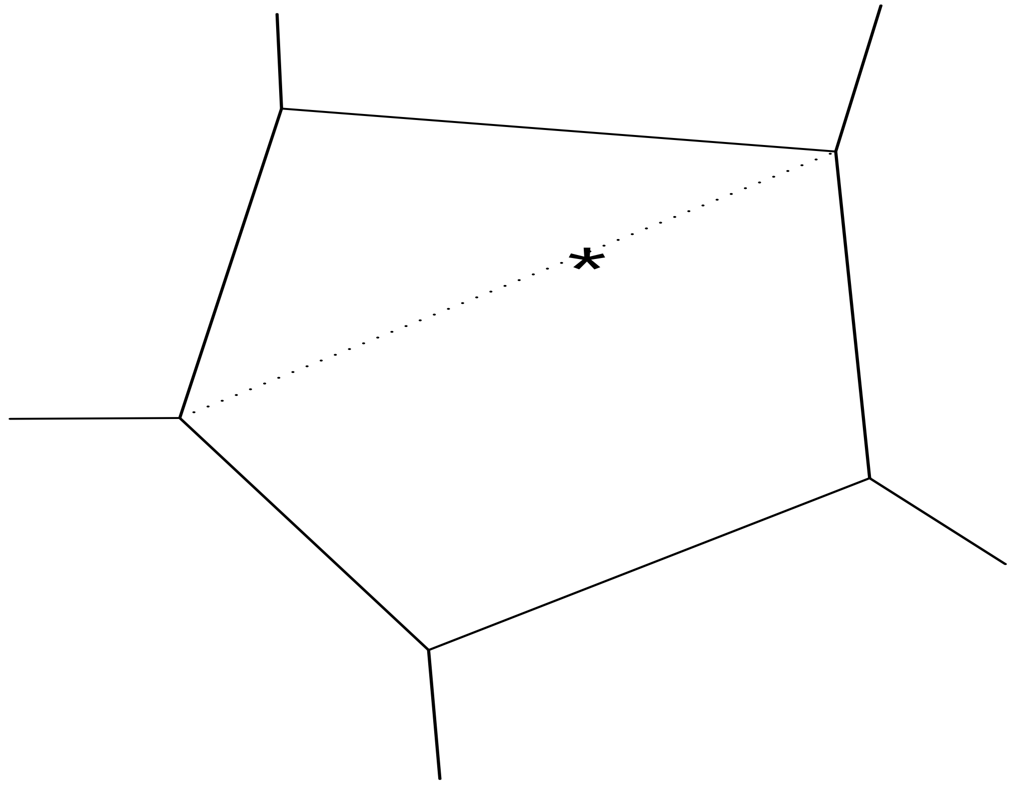
[12] Hartvigsen, D. (1992) Recognizing Voronoi diagrams with linear programming. *ORSA J. on Computing*, **4**(4), 369–374.

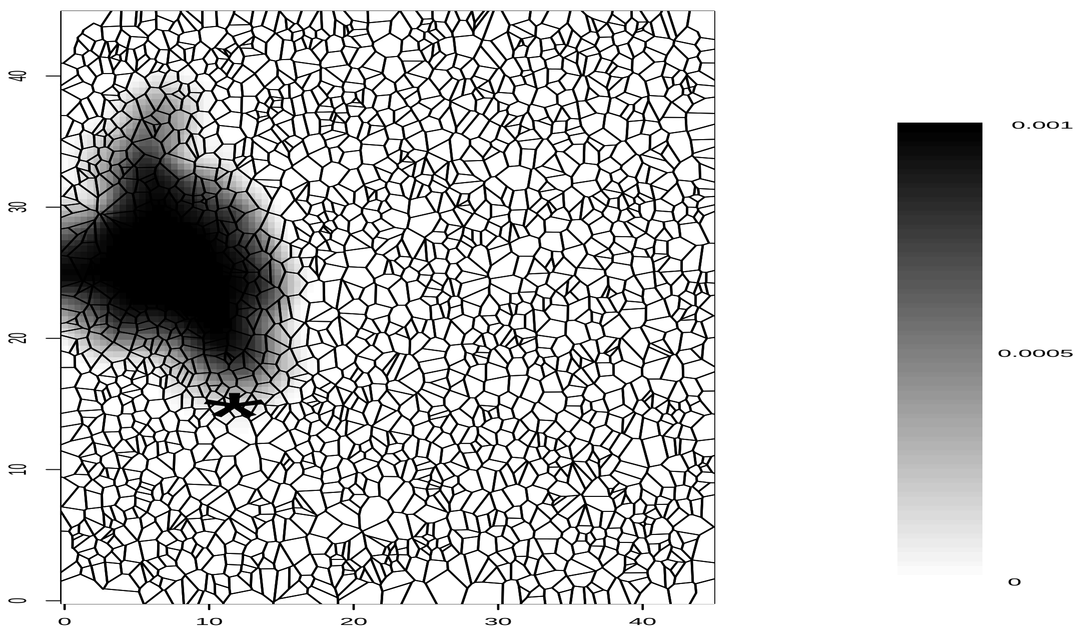
[13] Adamatzky, A. (1993) Massively parellel algorithm for inverting Voronoi diagram. *Neural Network World*, **4**, 385–392.

List of Figures and their Captions:

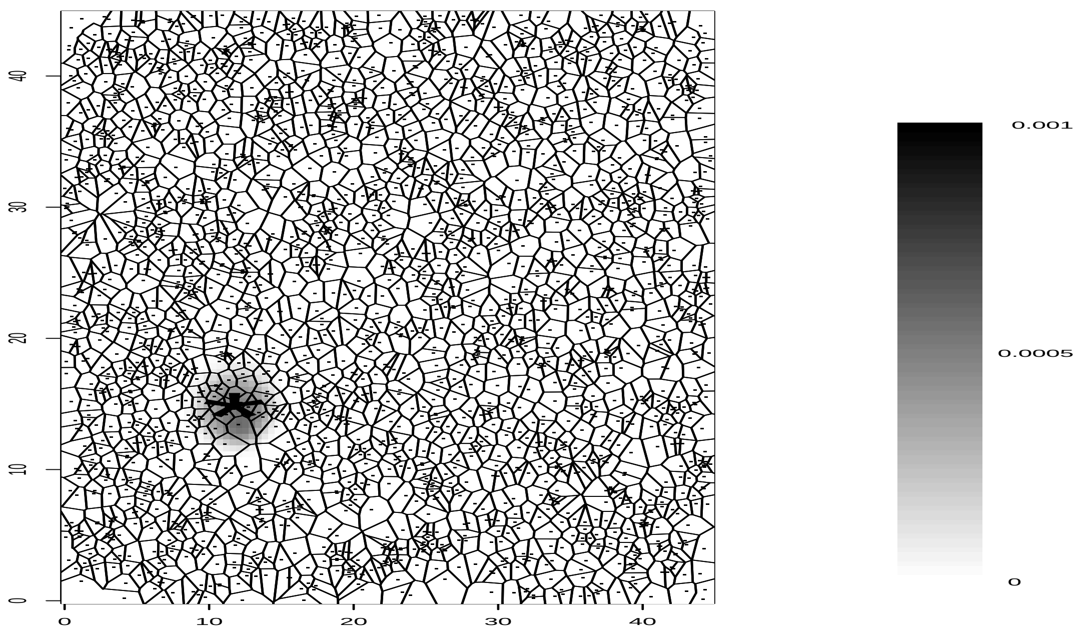



P$_2$
D
A
θ_1
θ_1
B
θ_2
θ_3
$\theta_2+\theta_3$
P$_1$
E
P$_3$
C

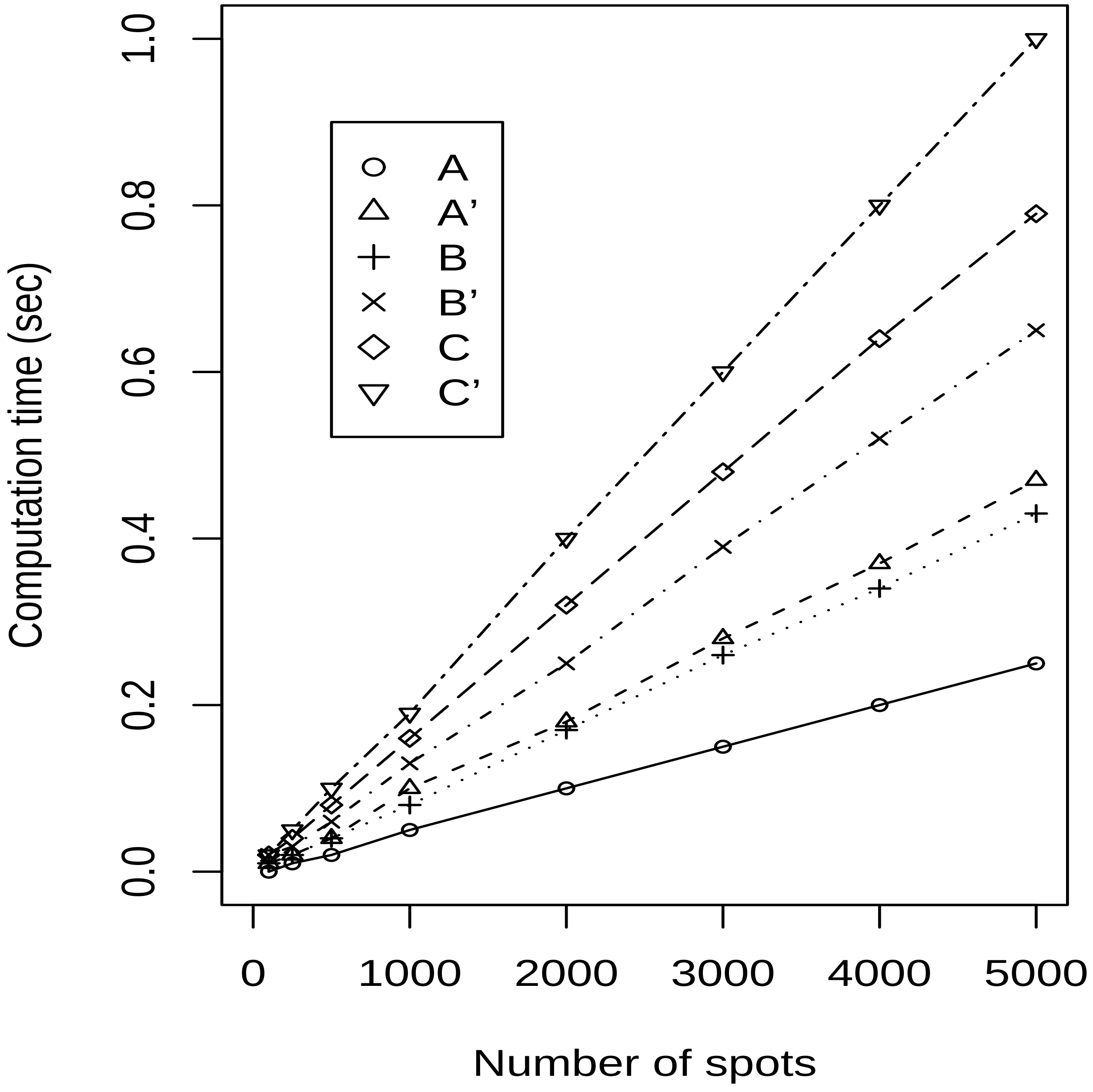




0
10
20
30
40
0
10
20
30
40
0.001
0.0005
0



0
10
20
30
40
0.001
0.0005
0



Computation time (sec)
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
Number of spots
A
A’
B
B’
C
C’