Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2015, Article ID 787163, 10 pages
http://dx.doi.org/10.1155/2015/787163

Research Article

An Approach to Generate Spatial Voronoi Treemaps for Points, Lines, and Polygons

Song Tian,[1] Ximin Cui,[1] and Yu Gong[2]

[1]*College of Geoscience and Surveying Engineering, China University of Mining and Technology, Beijing 100083, China*
[2]*South China Sea Branch, State Oceanic Administration, Guangzhou 510310, China*

Correspondence should be addressed to Song Tian; tsisatc@163.com

Received 1 April 2015; Revised 4 July 2015; Accepted 8 July 2015

Academic Editor: George S. Tombras

As a space-filling method, Voronoi Treemaps are used for showcasing hierarchies. Previously presented algorithms are limited to visualize nonspatial data. The approach of spatial Voronoi Treemaps is proposed in this paper to eliminate these problems by enabling the subdivisions for points, lines, and polygons with spatial coordinates and references. The digital distance transformation is recursively used to generate nested raster Voronoi polygons while the raster to vector conversion is used to create a vector-based Treemap visualization in a GIS (geographic information system) environment. The objective is to establish a spatial data model to better visualize and understand the hierarchies in the geographic field.

1. Introduction

As a well-known approach of visualizing hierarchical structures, Treemaps use recursive subdivision algorithms to partition an n-dimensional plane into the nested regions without producing holes or overlaps. The size of each region corresponds to the value of a data element and the nesting depth represents the depth of the data element in the corresponding hierarchical data structure [1]. Treemaps were first presented by Johnson and Shneiderman in 1991 designed to visualize the files on a hard drive [2]. Nowadays, they have been widely applied to a variety of domains, such as the sports reporting [3], the photo management [4], the business data visualization [5], and the repository search visualization [6].

Previously presented Treemap algorithms mainly focus on displaying hierarchical data using the nested rectangles, such as Slice-and-Dice Treemaps [7], Squarified Treemaps [8], Strip Treemaps [4], Spiral Treemaps [9], and Spatially Ordered Treemaps [10]. They have one thing in common: their branches are all given rectangles which are tiled with smaller rectangles in accordance with their respective generation norms. Subsequently, approaches to visualize the hierarchies with the nonrectangular regions were addressed, such as Voronoi Treemaps [11, 12], Jigsaw Treemaps [13], and Circular Treemaps [14].

Voronoi Treemaps enable the two-dimensional subdivisions of the arbitrary polygons following the norms of Treemaps and Voronoi diagrams within the areas of arbitrary shapes. Since introduced by Balzer and Deussen [11, 12] to visualize the software metrics, they have been used in many applications ranging from the biodiversity exhibition [15] to the gene expression [16]. But they are rarely used in the field of geography, let alone integrated with the mainstream GIS software. For better integration of Voronoi Treemaps and GIS, we present an approach of generating spatial Voronoi Treemaps for points, lines, and polygons to better visualize the hierarchies in the geographic field. For simplicity, we use the term "SVT" for the spatial Voronoi Treemaps for points, lines, and polygons; except explicit specification is made.

In Section 2, the principles of the SVT and the relevant concepts are introduced. Section 3 elaborates the implementation of the layout algorithm. Section 4 shows several examples. Discussions and the conclusions are given in Section 5.

2. Preliminary Knowledge

The SVT is a visualization method to create geographical map-like representations of trees with ordinary or weighted

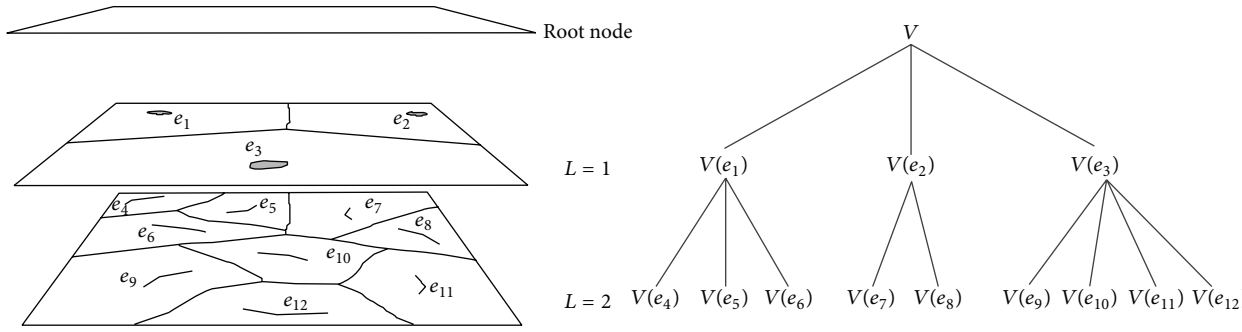

Figure 1: An SVT and its corresponding tree.

Voronoi polygons (or called cells) from a set of generators, such as points, lines, and polygons. As is shown in Figure 1, an SVT is composed of nested cells and their corresponding generators. A level i cell contains all level $i + 1$ generators that are closer to that level i generator than to any other (e.g., a level 1 cell $V(e_1)$ contains three level 2 generators e_4, e_5, and e_6). Each cell in the hierarchy has a size that is the sum size of its contained leaf cells (e.g., the size of $V(e_1)$ is equal to the total size of $V(e_4)$, $V(e_5)$, and $V(e_6)$) and a depth that corresponds to its contained generator's depth. The initial boundary of all generators is regarded as the root node, which has an arbitrary shape, mostly a rectangle.

As a combination of Voronoi Treemaps and Voronoi diagrams, the SVT is motivated by the work of Balzer and Deussen on Voronoi Treemaps [11, 12] and the work of Dong on multiplicatively weighted Voronoi diagrams for point, line, and polygon features [17]. Traditional Voronoi Treemaps consist of nested cells without spatial information. Differently, the nested cells and their corresponding generators, both of which have spatial coordinates and references, are used by our proposed Treemap layout to express hierarchies. Generators can represent various kinds of geographic objects, such as cities, railways, and lakes, and cells are their corresponding dominant areas. Voronoi Treemap algorithms usually use centroidal Voronoi tessellations to create nested polygonal structures. In each recursion step, generators are generated randomly and shift to the center of mass of its cell using Lloyd's method [18], whereas generators of different levels are preset and fixed for an SVT. At each recursive subdivision step, the cells are subdivided into smaller polygons from the generators contained within.

Generators can be obtained from the files in shapefile format for an SVT. The shapefile format is a popular geospatial vector data format for GIS software, which is developed and regulated by Environmental Systems Research Institute (ESRI) as an open specification for data interoperability among GIS software products. It can describe vector spatial features (points, lines, and polygons) with spatial coordinates and references. Each feature commonly has attributes to describe itself, such as the name and the location. To clearly convey the idea of the SVT, several key concepts are introduced first.

2.1. Relevant Definitions

Definition 1 (see [19]). Let p be a point and let e be a generator (a point, line, or polygon) in R^2; the distance between p and e is defined as below:

$$d(p, e) = \inf\{d(p, q) \mid q \in e\}, \tag{1}$$

where the denotation $d(p, q)$ represents a specified distance function between two points $p(x_p, y_p)$ and $q(x_q, y_q)$; mostly the Euclidian metric is used, which is defined as

$$d_e(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}, \tag{2}$$

but others, such as the Manhattan metric or the Chessboard metric, are possible as well.

The Manhattan metric is

$$d_m(p, q) = |x_p - x_q| + |y_p - y_q|. \tag{3}$$

The Chessboard metric is

$$d_c(p, q) = \max(|x_p - x_q|, |y_p - y_q|). \tag{4}$$

In vector mode, x_p and x_q are x-coordinates and y_p and y_q are y-coordinates. In raster mode, the coordinates are defined by integer numbers of row and column of raster pixels. Figure 2 shows three distance metrics in raster mode.

Definition 2. Let $E = \{e_1, e_2, \ldots, e_n\}$ be a set of n distinct generators in R^2. For each $e_i \in E$, a weight $w(e_i)$ is assigned. A weighted Voronoi diagram for points, lines, and polygons $V(E, w(E)) = \{V(e_1, w(e_1)), V(e_2, w(e_2)), \ldots, V(e_n, w(e_n))\}$ denotes a subdivision of R^2 into n cells with a property that a point p lies in a cell $V(e_i)$ if and only if $d_{mw}(p, e_i) < d_{mw}(p, e_j)$ for each $e_i, e_j \in E$ with $i \neq j$. The notation $d_{mw}(p, e_i)$ is defined as

$$d_{mw}(p, e_i) = \frac{d(p, e_i)}{w(e_i)}, \quad w(e_i) > 0. \tag{5}$$

If $w(e_1) = w(e_2) = \cdots = w(e_n)$, $V(E, w(E))$ is reduced to an ordinary Voronoi diagram for points, lines, and polygons $V(E) = \{V(e_1), V(e_2), \ldots, V(e_n)\}$.

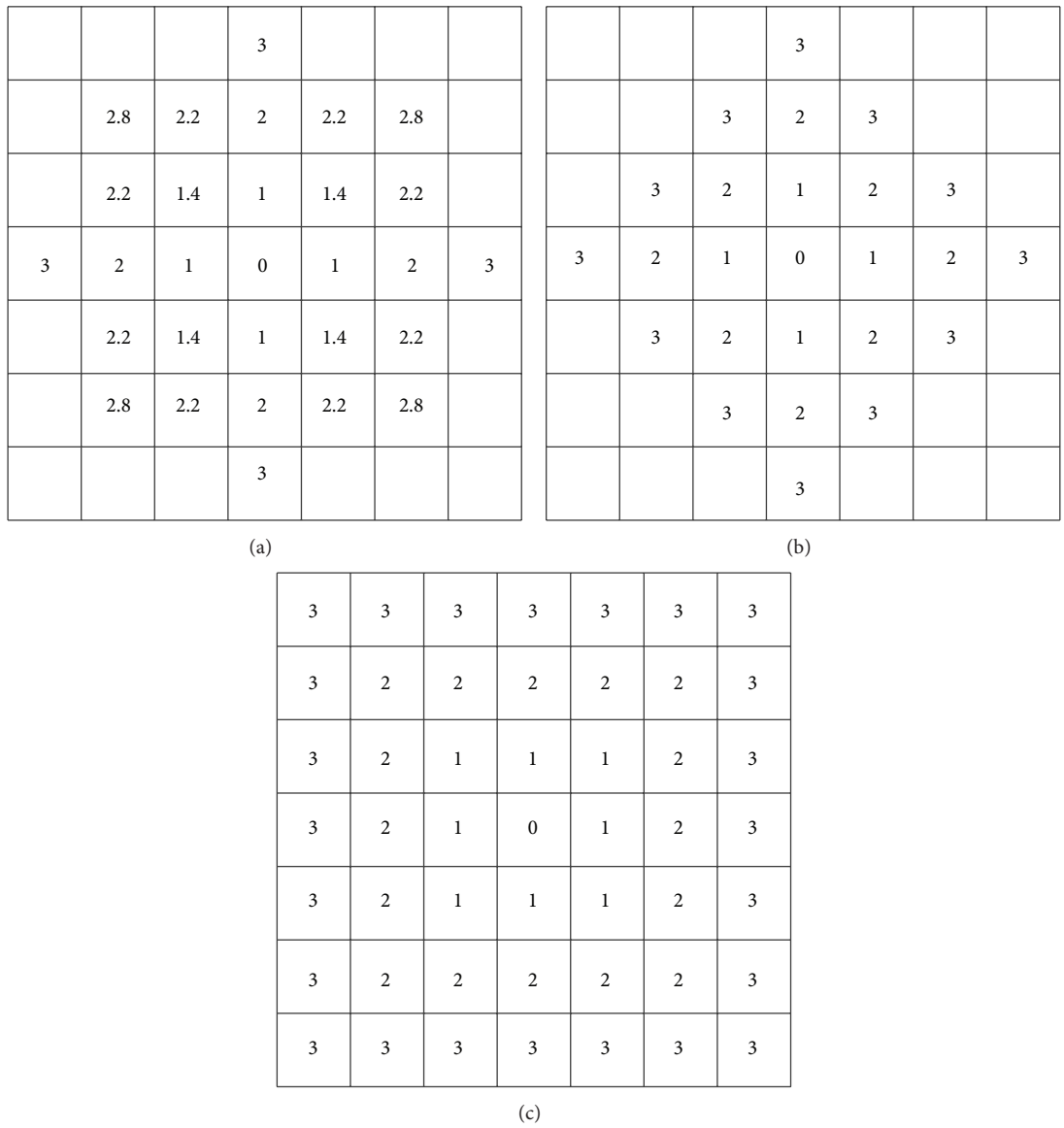

Figure 2: The Euclidean, Manhattan, and Chessboard distance in raster mode. (a) A raster region of $d_e \leq 3$, (b) a raster region of $d_m \leq 3$, and (c) a raster region of $d_c \leq 3$.

Definition 3. Let $V(E, w(E)) = \{V(e_1, w(e_1)), V(e_2, w(e_2)), \ldots, V(e_n, w(e_n))\}$ be a weighted Voronoi diagram for points, lines, and polygons. $V(e_i, w(e_i))$ denotes the cell of the generator e_i with a positive real weight $w(e_i)$. If there are any weighted Voronoi diagrams $V(F_i, w(F_i)) = \{V(f_x, w(f_x)) \mid V(f_x, w(f_x)) \subset V(e_i, w(e_i)),\ f_x \in V(e_i, w(e_i)),\ 1 \leq x < \infty,\ 1 \leq i \leq n\}$ which could meet the criteria $\sum V(f_x, w(f_x)) \equiv V(e_i, w(e_i))$, then the expression $V(E, w(E)) = \{V(F_1, w(F_1)), V(F_2, w(F_2)), \ldots, V(F_n, w(F_n))\}$ is called a weighted SVT. Regardless of weights of generators, the expression $V(E) = \{V(F_1), V(F_2), \ldots, V(F_n)\}$ is named as an ordinary SVT. A weighted SVT is a special case of the ordinary SVT that takes weights into consideration. Both of them are collectively called the SVT.

2.2. Distance Transformation. It can be known from the above contents that an SVT consists of layers of ordinary or weighted Voronoi diagrams for points, lines, and polygons. To construct an SVT, we need to generate its compositions first. The Voronoi diagram or weighted Voronoi diagram algorithms can be divided into two categories: vector-based methods and raster-based methods. Vector-based methods are suitable for point sets but not good for line and area sets. By contrast, the computation of Voronoi diagrams or weighted Voronoi diagrams for line and polygon sets is as easy as that for point sets in raster mode. Computation via distance transformation is one of the important raster-based methods.

A distance transformation is a derived form of a digital image. Supposing that the digital binary image consists of

Input: a set of n generators $E = \{e_1, e_2, \ldots, e_n\}$ meeting the criteria that each $e_k \in E$ corresponds to an unique identifier $i(e_k)$, a level $l(e_k)$ and a weight $w(e_k)$ with $i(e_k) > 0$, $w(e_k) > 0$ and $l(e_k) = 1, 2, \ldots, h$;
V_0: the boundary of E; *Height*: the number of rows of the output image; *Width*: the number of columns of the output image
Output: an SVT $\{V_1, V_2, \ldots, V_h\}$
Begin
$z = Width \times Height$
initialize C: a set of z pixels for V_0
$h = \max(l(e_1), l(e_2), \ldots, l(e_n))$ %Get the maximum level value of E
for $i := 1$; **step** 1 **until** h **do**
initialize G_i: a set of z pixels for V_i
$E_i = GetLevelGenerators(E, i)$ %E_i: an array for level i set of generators.
$PixelsAssignment(G_i, E_i)$
$count = 0$
while $(count \neq z)$ **do**
$PixelsTransformation(G_i)$
$PixelsUpdate(G_i, count)$
endwhile
$C = G_i$
$TreemapExport(G_i)$
endfor
End

Algorithm 1: SVT subdivision.

feature and nonfeature pixels and the features can be points, lines, or polygons, the distance transformation is an operator that converts this binary image to a grey-level image where all pixels have a value corresponding to the distance to the nearest feature pixel [20]. It uses only a structuring mask at a time based on the following idea: global distances in the image are approximated by propagating local distances [20]. There are various types of distance transformations, depending on which distance metric is used to determine the distance between pixels. In this paper, three most basic metrics are used. They are the Manhattan metric (a cross shaped mask), the Chessboard metric (a 3×3 square mask), and the Euclidean metric.

In a digital space, the fulfillment of Euclidean geometry is a complex operation due to the discrete structure [21]. In order to get a better approximation of Euclidean distance transformation, several approaches are proposed, such as chamfer distance algorithms [20], sequence algorithms [22, 23], and parallel algorithms [24]. To simplify the calculation, octagonal distance is used to approximate Euclidean distance transformation, which is an alternate usage of Chessboard distance and Manhattan distance [20]. It needs to point out that it is not our intention to acquire a better approximation of Euclidean distance transformation. Such discussions will be encouraged following this paper.

3. The SVT Algorithm

The SVT uses a recursive layout algorithm to segment a two-dimensional plane into nested cells from a set of generators. Generators can be points, lines, and polygons that represent different forms of spatial objects in the real world, among which points are abstracted as their locations, lines represent their skeletons, and polygons depict their boundaries. Each generator is assigned two initial parameters. One is "level" that represents generator's hierarchy and the other is "weight" that reflects generator's certain property.

The construction of SVTs is as follows: first, classify generators into different sets in accordance with their level values. Then, regard the initial boundary and its contained level 1 set of generators as a digital image and process it via distance transformation. The output is a set of raster cells, which makes up a raster ordinary or weighted Voronoi diagram of level 1. For the next hierarchy level, this procedure is performed recursively for all level $i + 1$ generators within respective level i cells. When the recursion ends, converting the digital images to vector files and overlaying them, a complete layout of SVT is obtained.

For simplicity, let V_i represent an ordinary Voronoi diagram $V(E)$ or a weighted Voronoi diagram $V(E, w(E))$ of level i in the hierarchy, and let $V_i(j)$ denote an ordinary Voronoi region $V_i(e_j)$ or a weighted Voronoi region $V_i(e_j, w(e_j))$ of e_j; except explicit specification is made in this section.

In Algorithm 1, C is a set of z pixels determined by V_0, *Width,* and *Height*. Each pixel $c_j \in C$ corresponds to five properties: an identifier $i_c(c_j)$, an identifier of the father node $i_f(c_j)$, an original transformation speed $t_o(c_j)$, a current transformation speed $t_c(c_j)$, and a status $s(c_j)$. Consider $s(c_j) \in \{-1, 0, 1\}$, where "$s(c_j) = -1$" signifies that c_j is waiting to be processed, "$s(c_j) = 0$" represents that c_j is being processed, and "$s(c_j) = 1$" denotes that pixel c_j has been processed. While initializing C for an ordinary SVT, define $i_c(c_j) = -1$, $i_f(c_j) = -1$, and $s(c_j) = -1$ for each $c_j \in C$. While initializing C for a weighted SVT, define $i_c(c_j) = -1$, $i_f(c_j) = -1$, $s(c_j) = -1$, $t_o(c_j) = 0$, and $t_c(c_j) = 0$ for each $c_j \in C$. G_i is a set of z pixels for V_i, in which each $g_j \in G_i$ has the same properties as c_j in C. While initializing G_i for an ordinary SVT, define $i_c(g_j) = -1$, $i_f(g_j) = i_c(c_j)$, and

Input: a set of n generators $E = \{e_1, e_2, \ldots, e_n\}$, a level value i
Output: the level i set of generators E_i
Begin
Initialize an array E_i for the level i set of generators.
for $j := 1$; **step** 1 **until** n **do**
  if $(l(e_j) = i)$ **then**
    $E_i \leftarrow e_j$%add e_j to E_i
  endif
endfor
End

Algorithm 2: *GetLevelGenerators*().

(a)
Input: a set of m generators $E_i = \{e_1, e_2, \ldots, e_m\}$, a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Output: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Begin
for $j := 1$; **step** 1 **until** z **do**
  for $k := 1$; **step** 1 **until** m **do**
    if $(g_j \cap e_k \neq \{\})$ **then**
    $i_c(g_j) = i(e_k)$
    $s(g_j) = 0$
    endif
  endfor
endfor
End

(b)
Input: a set of m generators $E_i = \{e_1, e_2, \ldots, e_m\}$, a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Output: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Begin
$maxweight = \max(w(e_1), w(e_2), \ldots, w(e_m))$ %Get the maximum weight of E_i
for $j := 1$; **step** 1 **until** z **do**
  for $k := 1$; **step** 1 **until** m **do**
    if $(g_j \cap e_k \neq \{\})$ **then**
    $i_c(g_j) = i(e_k)$
    $t_o(g_j) = w(e_k)/maxweight$
    $t_c(g_j) = t_o(g_j)$
    $s(g_j) = 0$
    endif
  endfor
endfor
End

Algorithm 3: (a) *PixelsAssignment*() for an ordinary SVT. (b) *PixelsAssignment*() for a weighted SVT.

$s(g_j) = -1$ for each $g_j \in G_i$. While initializing G_i for a weighted SVT, define $i_c(g_j) = -1$, $i_f(g_j) = i_c(c_j)$, $s(g_j) = -1$, $t_o(g_j) = 0$, and $t_c(g_j) = 0$ for each $g_j \in G_i$. E_i is a subset of E with $l(e_k) = i$ for each $e_k \in E_i$, which is obtained from the function *GetLevelGenerators*(). *count* is the number of pixels with $s(g_j) = 1$ in G_i, which is obtained from the function *PixelsUpdate*().

The function *GetLevelGenerators*() (Algorithm 2) is to acquire a level i set of generators from E.

A weighted SVT subdivides the two-dimensional space recursively with consideration of both location and weight of each generator, while an ordinary SVT partitions space only considers generator's location. The different division approaches necessitate a different handling of the auxiliary functions *PixelsAssignment*() and *PixelsTransformation*() in Algorithm 1.

The function *PixelsAssignment*() (Algorithm 3) is to assign values of generators to pixels correspondingly if pixels intersect with generators in the two-dimensional raster space, which is a vector to raster conversion of spatial objects in essence. Specifically speaking, assign $i(e_k)$ to $i_c(g_j)$ and set $s(g_j)$ to 0 if a pixel g_j intersects with a generator e_k. In

(a)
Input: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Output: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Begin
for $j := 1$; **step** 1 **until** z **do**
  if $(s(g_j) = 0)$ **then**
    initialize R: a set of x pixels around g_j
    for $k := 1$; **step** 1 **until** x **do**
      if $(i_c(r_k) < 0$ & $i_f(r_k) = i_f(g_j))$ **then**
        $i_c(r_k) = i_c(g_j)$
      endif
    endfor
    $s(g_j) = 1$
  endif
endfor
End

(b)
Input: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Output: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$
Begin
for $j := 1$; **step** 1 **until** z **do**
  if $(s(g_j) = 0)$ **then**
    $t_c(g_j) = t_o(g_j) + t_c(g_j)$
    if $(t_c(g_j) \geq 1)$ **then**
      initialize R: a set of x pixels around g_j
      for $k := 1$; **step** 1 **until** k **do**
        if $(i_c(r_k) < 0$ & $i_f(r_k) = i_f(g_j))$ **then**
          $i_c(r_k) = i_c(g_j)$
          $t_o(r_k) = t_o(g_j)$
          $t_c(r_k) = t_c(g_j) - 1$
        endif
      endfor
      $s(g_j) = 1$
    endif
  endif
endfor
End

Algorithm 4: (a) *PixelsTransformation*() for an ordinary SVT. (b) *PixelsTransformation*() for a weighted SVT.

addition, $t_o(g_j)$ is set to $w(e_k)/\max(w(E_i))$ for a weighted SVT. It can be known that $t_o(g_j) \in (0, 1]$, which determines the speed of distance transformation of pixel g_j. $t_c(g_j)$ is an accumulator of $t_o(g_j)$. In this function, define $t_c(g_j) = t_o(g_j)$. Figures 3(a) and 3(b) describe the variations of pixel properties for a weighted SVT after executing the function *PixelsAssignment*().

The function *PixelsTransformation*() (Algorithm 4) is to create a raster Voronoi diagram or weighted Voronoi diagram via distance transformation. For an ordinary SVT, each pixel g_j with $s(g_j) = 0$ will propagate in each loop in accordance with a structuring mask R, which can be a cross shaped mask (the Manhattan metric), a 3×3 square mask (the Chessboard metric), a disk shaped mask (the Euclidean metric), and so forth. Once a pixel g_j propagates, its state $s(g_j)$ will change to 1, which means that this pixel will cease to be processed in the next loop. If g_j's neighbor x pixels r_k $(k \in [1, x])$ own the same father node with g_j $(i_f(r_k) = i_f(g_j))$ and have not been processed $(i_c(r_k) < 0)$, assign $i_c(g_j)$ to $i_c(r_k)$.

Input: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$, *count*
Output: a set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$, *count*
Begin
for $j := 1$; **step** 1 **until** z **do**
  if $(i_c(g_j) > 0)$ **then**
    if $(s(g_j) = -1)$ **then**
      $s(g_j) = 0$
    endif
    if $(s(g_j) = 1)$ **then**
      count ++
    endif
  endif
endfor
End

Algorithm 5: *PixelsUpdate*().

For a weighted SVT, we use the property $t_c(g_j)$ to control whether a pixel g_j with $s(g_j) = 0$ propagates in each loop. If $t_c(g_j)$ is less than an established threshold, it continues to execute accumulation. Once it exceeds the threshold, g_j will propagate and its state $s(g_j)$ will change to 1. In Algorithm 4, as the maximum of original transformation speed is 1, so the threshold is set to 1. If g_j's neighbor x pixels r_k $(k \in [1, x])$ own the same father node with g_j $(i_f(r_k) = i_f(g_j))$ and have not been processed $(i_c(r_k) < 0)$, the properties of r_k will be assigned as follows: $i_c(r_k) = i_c(g_j)$, $t_o(r_k) = t_o(g_j)$ and $t_c(r_k) = t_c(g_j) - 1$. It can be known from Algorithm 4 that a pixel with a higher original transformation speed owns a higher priority to execute distance transformation. For example, a pixel g_a with $t_o(g_a) = 1$ will propagate every time and a pixel g_b with $t_o(g_b) = 0.5$ will propagate every two times. From the point of view of vector space, it means that a generator with a bigger weight will dominate more area. Figures 3(d), 3(e), and 3(f) describe the first three steps of distance transformation for a raster layer of a weighted SVT and the variations of pixel properties.

The function *PixelsUpdate*() (Algorithm 5) is to update the status of pixels and calculate the parameter *count*. In each loop, if a pixel g_j has been assigned values by a neighbor pixel $(i_c(g_j) > 0)$ yet has not executed distance transformation $(s(g_j) = -1)$, set $s(g_j)$ to 0, which means that this pixel will participate in the next distance transformation. Figures 3(c) and 3(d) show an instance of the function *PixelsUpdate*() and the variations of pixel properties.

The function *TreemapExport*() specifies the output mode of SVTs: for each level set of generators $E_i = \{e_1, e_2, \ldots, e_m\}$ $(i = 1, 2, \ldots, h)$, a level i set of pixels $G_i = \{g_1, g_2, \ldots, g_z\}$ that represents a raster ordinary or weighted Voronoi diagram for points, lines, and polygons is exported correspondingly. Pixels that have the same identifier will be converted to a vector ordinary or weighted Voronoi polygon $V_i(j)$ $(j = 1, 2, \ldots, m)$ through a raster to vector conversion. Then, a vector file in shapefile format V_i with

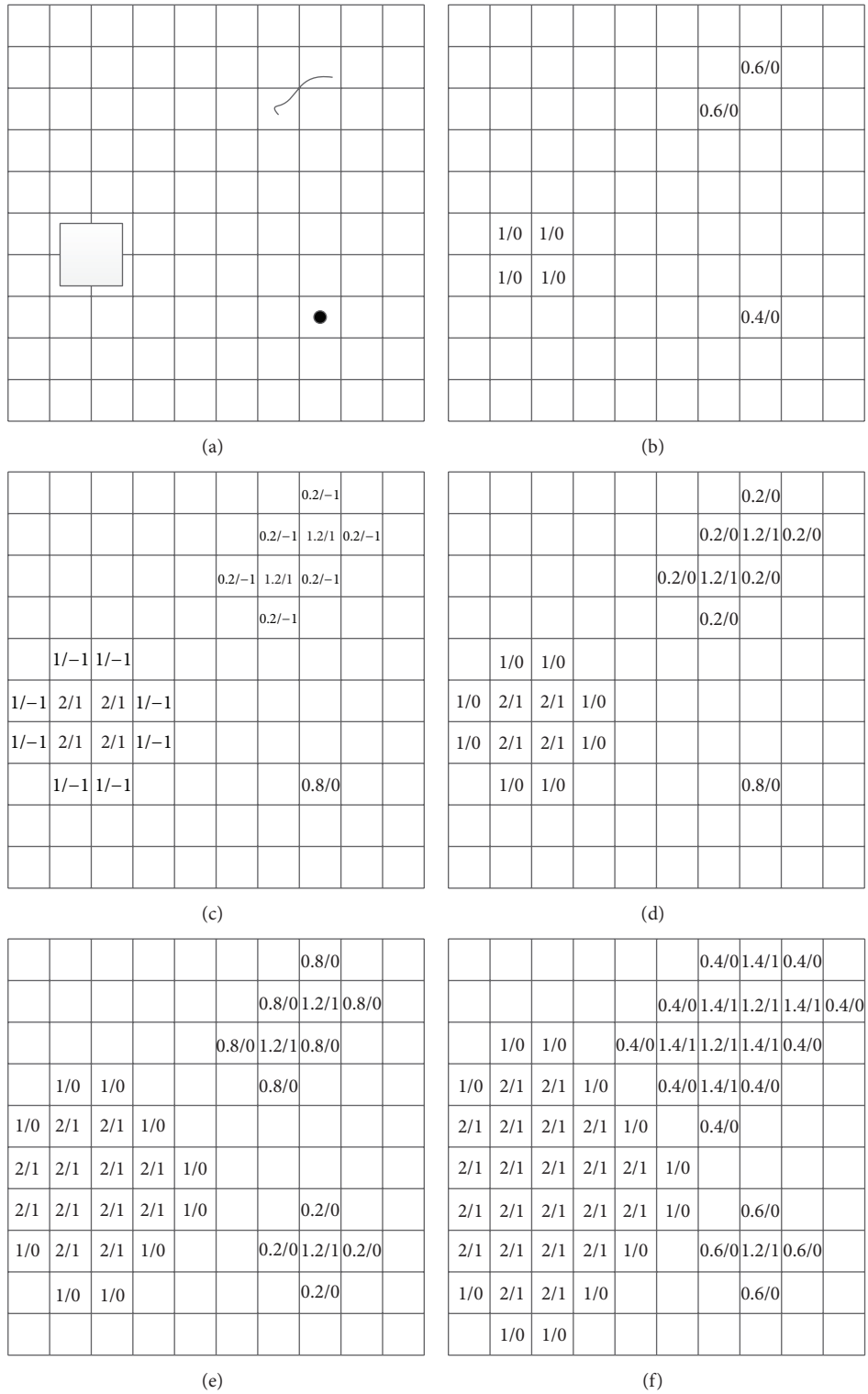

FIGURE 3: The core process of the subdivision for a raster layer of a weighted SVT: (a) three generators e_1, e_2, and e_3 within a 10×10 binary image, where e_1 is a square with $w(e_1) = 10$, e_2 is a line with $w(e_2) = 6$, and e_3 is a point with $w(e_3) = 4$; (b) pixels assignment for e_1, e_2, and e_3. A pixel g_j ($j = 1, 2, \ldots, 100$) will be marked with a sign "$t_c(g_j)/s(g_j)$" if it intersects with a generator, in which $t_c(g_j)$ is the current transformation speed of the pixel and $s(g_j)$ is the status. Blank pixels are marked with "0/ − 1" in default. (c) The Manhattan distance transformation for e_1, e_2, and e_3 for the first time. If $t_c(g_j) < 1$, it continues to execute accumulation, else g_j will propagate and $s(g_j)$ will change to 1. (d) Pixels update for e_1, e_2, and e_3 for the first time. The state $s(g_j)$ of the pixel g_j will change to 0 if g_j has been assigned values yet has not executed distance transformation ($s(g_j) = -1$). (e) The Manhattan distance transformation and pixels update for e_1, e_2, and e_3 for the second time. (f) The Manhattan distance transformation and pixels update for e_1, e_2, and e_3 for the third time.

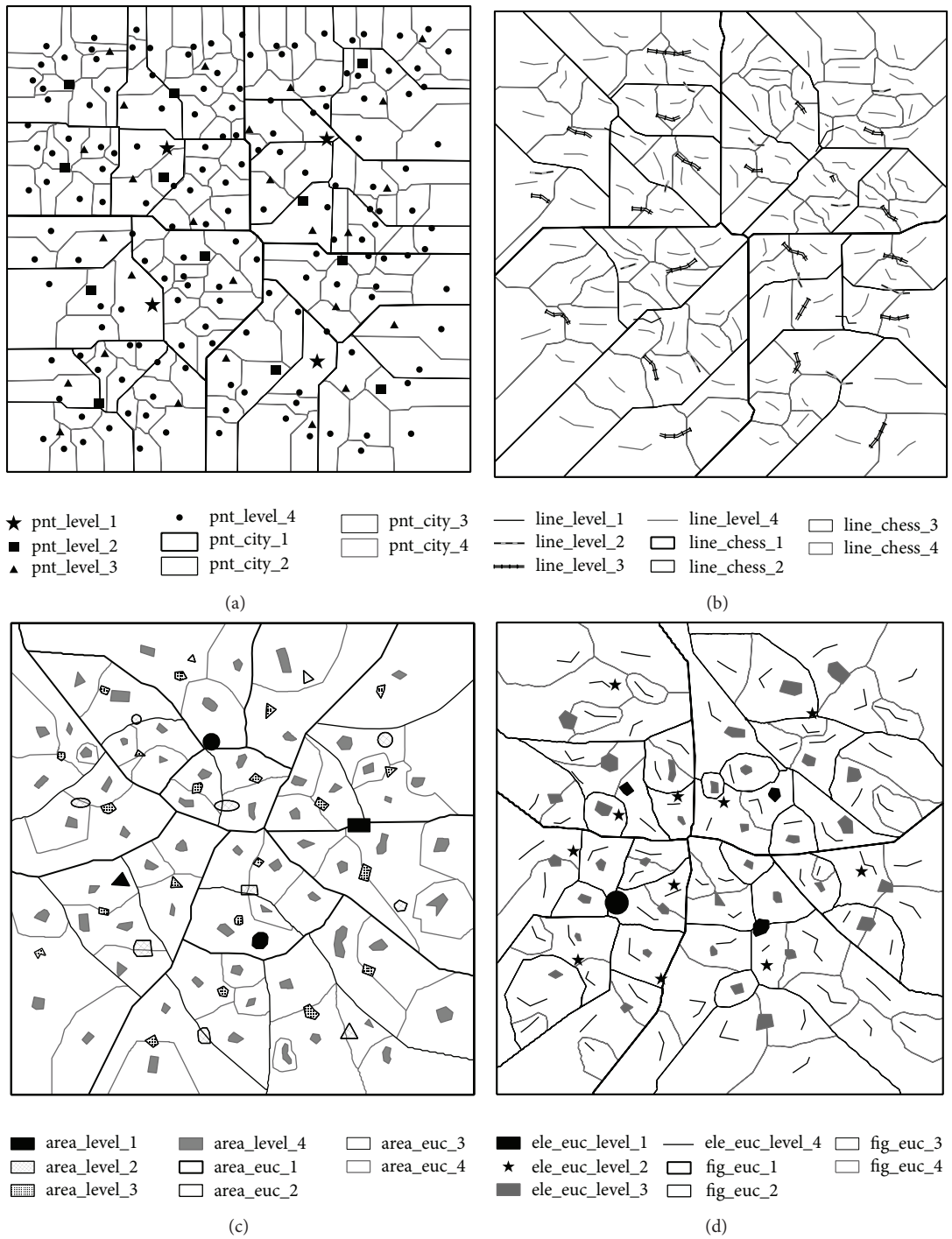

Figure 4: Different types of SVTs. (a) An ordinary SVT for points based on the Manhattan metric; (b) an ordinary SVT for lines based on Chessboard metric; (c) a weighted SVT for polygons based on the Euclidean metric; (d) a weighted SVT for points, lines, and polygons based on the Euclidean metric.

spatial coordinates and references can be obtained. Overlaying $E_1, E_2, \ldots, E_h$ with $V_1, V_2, \ldots, V_h$, we can construct a final hierarchical layout suitable for GIS environment.

4. Examples

Four Treemap layouts shown in Figure 4, respectively, use the data set with 200, 130, 100, and 130 generators that are randomly generated without spatial references at 4 hierarchy levels. Figure 4(a) is an ordinary SVT for points based on the Manhattan metric; Figure 4(b) is an ordinary SVT for lines based on the Chessboard metric; Figure 4(c) is a weighted SVT for polygons based on the Euclidean metric; Figure 4(d) is a weighted SVT for points, lines, and polygons based on the Euclidean metric. Different legend symbols represent different sets of generators. Border thickness expresses the

Table 1: Urban hierarchy by population in China.

Category	Population size	The number of cities	Level
Mega	≥10,000,000	13	1
Macro	5,000,000–10,000,000	76	2
Large/Middle/Micro	0–5,000,000	253	3

different levels of cells. The coarser boundaries indicate a higher hierarchical level of cells in the Treemap layout.

We implemented the SVT algorithm in C# and used ArcGIS Engine (a complete library of embeddable GIS components) to read shapefile and execute conversions between vector and raster data. The hardware of our experiments is an Intel Pentium Dual-Core E5800 3.2 GHz and 4 G RAM, running under Microsoft Windows 7. In this experiment, the program is running in single thread mode using one core of the CPU. The computations of four layouts shown in Figure 4 require 1.80 min, 1.88 min, 1.99 min, and 2.13 min, respectively. For many GIS applications, the speed is acceptable.

To better visualize the spatial hierarchies in the real world, we use the SVT to express the spatial pattern of the population landscape as shown in Figure 5, which is motivated by the research of Mu and Wang [25]. The point generators are abstracted as the locations of cities that are municipal level and above in China (342 cities in all, not including Zhoushan, Hong Kong, Macau, the Hainan Province, and Taiwan Province of China). The boundary is the inland border of China. They are, respectively, obtained from a point file and a polygon file in shapefile format at a scale of 1 : 100,000, that is, the Lambert projection-based maps.

According to [25], we only choose population as the weight of each city. The population data were obtained from the *Tabulation on the 2010 Population Census of the People's Republic of China by Township*. In China, cities can be divided into five categories based on the urban concentration of the population. They are "Mega" city, "Macro" city, "Large" city, "Middle" city, and "Micro" city. Their population sizes are "10,000,000 and above," "5,000,000–10,000,000," "1,000,000–5,000,000," "500,000–1,000,000," and "1–500,000," respectively. As the proportion of "Middle" and "Micro" city is small ("Mega": 13, "Macro": 76, "Large": 213, "Middle": 19, and "Micro": 21), we combined them with the 213 "Large" cities. Thus, an urban population hierarchy can be defined as in Table 1.

Using the SVT subdivision algorithm, we can get a spatial Voronoi Treemap that represents the population landscape pattern of urban hierarchy (Figure 5). In Algorithm 1, the generators $E = \{e_1, e_2, \ldots, e_n\}$ denote the cities in China and $n = 342$. For each $e_k \in E$, the unique identifier $i(e_k)$ is set to k, the level $l(e_k)$ corresponds to the value in field "Level" in Table 1, and the weight $w(e_k)$ is equal to the population of the city. V_0 is the inland border of China.

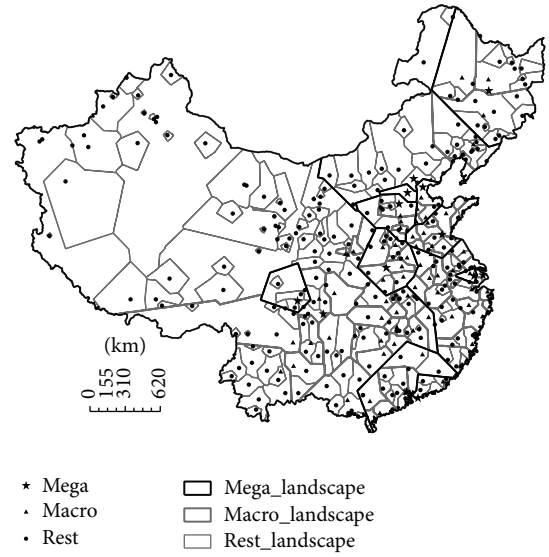

Figure 5: The population landscape pattern of urban hierarchy for China based on the Manhattan metric.

5. Discussions and Conclusions

This paper presents an approach for the generation of SVTs. Contrary to the existent Voronoi Treemaps that are used to visualize hierarchical nonspatial objects, our proposed layout algorithm allows to create Voronoi Treemap visualizations for geographical objects with spatial coordinates and references, which can serve as a spatial data model to express urban hierarchies, organization structures, region differences, and so forth.

The SVT approach accepts point, line, and polygon files in shapefile format, classifies generators into different groups according to their level values in their attribute tables, uses a vector to raster conversion to create an initial raster layer, and then generates raster images using distance transformation. Through a raster to vector conversion, layers of ordinary or weighted Voronoi diagrams in shapefile format can be generated to construct a Treemap layout, which can be applied in mainstream GIS software. Input generators and output Treemap layers can be stored together in geodatabase and flexibly change their styles. All GIS symbols and layout styles can be used to enhance the visual effect of Treemaps and to support the user in the perception and interpretation of the Treemap visualizations; for example, Figure 4(a) uses star, square, triangle, and circle symbols to depict different levels of point features.

The layout algorithm needs to loop through all generators one by one to generate an SVT, so the process may be relatively slow while facing massive input features. Because the major processing steps are raster-based, it is possible that an input generator does not have an output Voronoi polygon if that generator is very close to others and raster cell

size is not small enough. This problem can be eliminated by reducing the cell size.

In future work, the efficiencies and applications of the presented layout method will be studied extensively.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] D. Gotz, "Dynamic Voronoi Treemaps: a visualization technique for time-varying hierarchical data," Tech. Rep. RC25132, IBM, 2011.

[2] B. Johnson and B. Shneiderman, "Tree-maps: a space-filling approach to the visualization hierarchical Information structures," in *Proceedings of the 2nd International IEEE Visualization Conferenc (Visualization '91)*, pp. 284–291, San Diego, Calif, USA, October 1991.

[3] L. Jin and D. C. Banks, "Tennis viewer: a browser for competition trees," *IEEE Computer Graphics and Applications*, vol. 17, no. 4, pp. 63–65, 1997.

[4] B. B. Bederson, B. Shneiderman, and M. Wattenberg, "Ordered and quantum treemaps: making effective use of 2D space to display hierarchies," *ACM Transactions on Graphics*, vol. 21, no. 4, pp. 833–854, 2002.

[5] R. Vliegen, J. J. van Wijk, and E.-J. van der Linden, "Visualizing business data with generalized treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 789–796, 2006.

[6] E. C. Clarkson, K. Desai, and J. D. Foley, "ResultMaps: visualization for search interfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1057–1064, 2009.

[7] B. Shneiderman, "Tree visualization with tree-maps: 2-d space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92–99, 1992.

[8] M. Bruls, K. Huizing, and J. J. V. Wijk, "Squarified treemaps," in *Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pp. 33–42, Vienna, Austria, May 1999.

[9] Y. Tu and H.-W. Shen, "Visualizing changes of hierarchical data using treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1286–1293, 2007.

[10] J. Wood and J. Dykes, "Spatially ordered treemaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1348–1355, 2008.

[11] M. Balzer and O. Deussen, "Voronoi treemaps," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '05)*, pp. 49–56, Minneapolis, Minn, USA, October 2005.

[12] M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in *Proceedings of the ACM Symposium on Software Visualization (SoftVis '05)*, pp. 165–172, Saint Louis, Mo, USA, May 2005.

[13] M. Wattenberg, "A note on space-filling visualizations and space-filling curves," in *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '05)*, pp. 181–186, Minneapolis, Minn, USA, October 2005.

[14] K. Onak and A. Sidiropoulos, "Circular partitions with applications to visualization and embeddings," in *Proceedings of the 24th Annual Symposium on Computational Geometry*, pp. 28–37, College Park, Md, USA, June 2008.

[15] M. S. Horn, M. Tobiasz, and C. Shen, "Visualizing biodiversity with voronoi treemaps," in *Proceedings of the 6th International Symposium on Voronoi Diagrams (ISVD '09)*, pp. 265–270, IEEE, Copenhagen, Denmark, June 2009.

[16] J. Bernhardt, S. Funke, M. Hecker, and J. Siebourg, "Visualizing gene expression data via voronoi treemaps," in *Proceedings of the 6th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD '09)*, pp. 233–241, Copenhagen, Denmark, June 2009.

[17] P. L. Dong, "Generating and updating multiplicatively weighted Voronoi diagrams for point, line and polygon features in GIS," *Computers & Geosciences*, vol. 34, no. 4, pp. 411–421, 2008.

[18] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[19] Y. H. Zhang, "A method to draw weighted Voronoi diagram," *Computer Science*, vol. 28, no. 6, pp. 126–130, 2001.

[20] G. Borgefors, "Distance transformations in digital images," *Computer Vision, Graphics, and Image Processing*, vol. 34, no. 3, pp. 344–371, 1986.

[21] I.-M. Sintorn and G. Borgefors, "Weighted distance transforms in rectangular grids," in *Proceedings of the 11th International Conference on Image Analysis and Processing (ICIAP '01)*, pp. 322–326, Palermo, Italy, September 2001.

[22] B. Nagy, R. Strand, and N. Normand, "A weight sequence distance function," in *Proceedings of the 11th International Symposium (ISMM '13)*, pp. 292–301, Uppsala, Sweden, May 2013.

[23] R. Strand, B. Nagy, C. Fouard, and G. Borgefors, "Generating distance maps with neighbourhood sequences generating distance maps with neighbourhood sequences," in *Proceedings of the 13th International Conference (DGCI '06)*, pp. 295–307, Szeged, Hungary, October 2006.

[24] A. Fujiwara, M. Inoue, T. Masuzawa, and H. Fujiwara, "A cost optimal parallel algorithm for weighted distance transforms," *Parallel Computing*, vol. 25, no. 4, pp. 405–416, 1999.

[25] L. Mu and X. Wang, "Population landscape: a geometric approach to studying spatial patterns of the US urban hierarchy," *International Journal of Geographical Information Science*, vol. 20, no. 6, pp. 649–667, 2006.