Raster-Based Parallel Multiplicatively Weighted Voronoi Diagrams Algorithm with MapReduce

Ming Xu, Han Cao and Chang-ying Wang

Abstract While Voronoi diagram has been used in many fields, most vector-based methods of generating Voronoi diagrams focus mostly on point features, but they have difficulties in handling generators like lines or areas, which can be easily generated by raster-based methods, however, with substantial calculation cost. For the sake of integrating Voronoi diagram models with Web GIS, which inevitably encounters generators like lines and areas, we present a parallel algorithm with MapReduce for generating raster-based multiplicatively weighted Voronoi diagrams. The experiments and case studies show that the algorithm significantly improves the efficiency of generating Voronoi diagrams on large-scale raster data with potential use in urban public green space planning and optimal path planning.

Keywords Weighted Voronoi diagram · Parallel algorithm · Hadoop · MapReduce

1 Introduction

In mathematics, a weighted Voronoi diagram (WVD) in *n* dimensions is a Voronoi diagram for which the Voronoi cells are defined in terms of a distance defined by some common metrics modified by weights assigned to generator points. The multiplicatively weighted Voronoi diagram (MWVD) is defined when the distance between points is multiplied by positive weights [1].

M. Xu · H. Cao (✉)
Department of Computer Science, Shaanxi Normal University, Xian 710062, China
e-mail: caohan@snnu.edu.cn

C. Wang
Department of Computer Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China

The dominance region defined by subsequent Voronoi regions is generalized by the type of generator, such as points, lines, and polygons, various weights, plane constraints, and metric space used. Many researchers have focused on issues of WVD generation.

Hu et al. [2] used map algebra method to solve the problem of generating MWVDs for point, line, and polygon features on raster data. Wang et al. [3] presented a raster-based algorithm supported by the ArcInfo, which is capable to generate Voronoi diagrams for points, lines, and polygons in two-dimensional space. On the basis of Ref. [4], Dong [5, 6] provided a discrete algorithm for generating line segment WVD. However, the algorithm did not consider more complex generators such as line features with an arbitrary shape. Wu and Luo [7] proposed an algorithm based on cellular automata for constructing Voronoi diagram of complex entities in grid space. Dong [8] noted applications of MWVDs for points, polylines, and polygons using ArcGIS. Fan [9] presented a vector-based algorithm for generating WVDs, which, however, is also not suitable for complex spatial objects. Gong et al. [10] developed a vector-based algorithm to generate and update MWVDs for points, polylines, and polygons in C# and present several examples. However, like all other vector-based methods, when dealing with complex generators such as lines and areas, the algorithm needs to first decompose them into simpler elements, which may violate spatial integrity. Very recently, many researchers have used graphics hardware to improve the performance of computing Voronoi diagram. Rong et al. [11] presented a GPU-assisted Voronoi diagram algorithm for computing centroidal Voronoi tessellation (CVT). Xu et al. [12] proposed a raster-based algorithm to construct WVDs with GPU, which is capable to generate discrete WVDs in real time. Besides, Afsin et al. [13] proposed an approach of generating spatial index, Voronoi diagram, and efficient processing of a wide range of geospatial queries.

Although theoretical and computational aspects of WVDs have been extensively discussed, there are still some problems especially in case of complex generators. There are three major issues of generating WVDs: (1) Vector-based algorithms handle so many factors in calculation and storage, that it is difficult to generate WVDs for complex generators directly. (2) Raster-based approaches involve judging and computing distances for each grid, so it costs large amount of calculation and has to tolerate precision limitation. (3) Since sequential algorithms, especially raster-based approaches are inefficient to handle large-scale massive map data, parallel and high-efficiency methods for complex spatial features based on cloud computing are reasonable. Hence, considering types of generators and various weights, we present a raster-based parallel approach with Hadoop to generate MWVDs for polygons.

The remainder of this chapter is organized as follows. In Sect. 2, we first provide a review of relevant concepts of Voronoi diagrams, which build the foundation of proposed method in the chapter. In Sect. 3, raster-based parallel algorithm with pseudo-code is discussed. Section 4 presents experimental results to verify the performance and scalability of proposed approach. Potential applications of proposed approach are discussed in Sect. 5. Finally, Sect. 6 gives conclusions of the chapter and directions for future work.

2 Background: Voronoi Diagram

Voronoi diagram divides a space into disjoint polygons where the nearest neighbor of any generator inside a polygon is the generator of the polygon. In this section, we review the relevant concepts of the Voronoi diagrams [14].

2.1 Ordinary Voronoi Diagram

Consider a set of limited number of points, called generator points, in the Euclidean plane (in general, generators can be any type of spatial object). Every location in the plane can be assigned to the closest generator(s) with a certain distance metric. The set of locations assigned to each generator forms Voronoi polygon of that generator. The set of Voronoi polygons associated with all the generators is called the Voronoi diagram. The Voronoi polygon and Voronoi diagram can be formally defined as follows: Assume a set of generators $G = \{g_1, g_2, \ldots, g_n\}$, where $2 < n < \infty$ and $g_i \neq g_j$ for $i \neq j$, $i, j \in I_n = \{1, \ldots, n\}$. For each generator, the set of Voronoi polygon given by

$$V(g_i) = \bigcap_{j \neq i} \{p \mid d(p, g_i) < d(p, g_j)\} \tag{1}$$

where $d(p, g_i)$ specifies the minimum distance between p and g_i and is called the Voronoi diagram generated by G.

2.2 Weighted Voronoi Diagram

In many applications, not only the location but also the weight (or importance) and the spatial extent of a site should be taken into account. The influence of different generators on the surrounding is different, so the ordinary Voronoi diagram always cannot meet the needs of general spatial analysis. We need to improve and generalize the approach. WVDs can be divided into two types: MWVDs [15] and additively WVDs. For the former, the distance between points is multiplied by positive weights. For the latter, positive weights are subtracted from the distances between points. Based on the above concept, now we add the weight value of distance to ordinary Voronoi diagram.

Let $g_i \in G$ be an element with positive weight ω_i. The weight distance $d_\omega(g, g_i)$ between p and g_i is $d_\omega(p, g_i) = d(p, g_i)/\omega_i$. Then, the dominance of generators, called $V_\omega(g_i)$, can be represented by

$$V_\omega(g_i) = \bigcap_{j \neq i} \{p \mid d_\omega(p, g_i) < d_\omega(p, g_j)\}. \tag{2}$$

2.3 Weighted Voronoi Diagram for Polygons

In general, generators can be any type of spatial object, such as points, lines, and polygons. WVD for polygons [9, 16] is an important generalization of the ordinary Voronoi diagram in two sides of generator and weight. Most methods of computing Voronoi diagrams have some difficult in handling complex generators (not points). Many scholars have discussed the Voronoi diagram for lines or for polygons. Next, we focus on the WVD for polygons.

Assume a set of polygons $P = \{p_1, p_2, \ldots, p_n\}$, $d(p, p_i)$ specifies the minimum distance between p and p_i, and then, the Voronoi Diagram generated by P can be represented by

$$V_\omega(P_i) = \bigcap_{j \neq i} \left\{ p \,\middle|\, \frac{d(p, p_i)}{\omega_i} < \frac{d(p, p_j)}{\omega_j} \right\}. \tag{3}$$

3 Raster-Based MWVDs Parallel Algorithm with MapReduce

The general idea of the raster-based method [17] is to calculating raster distance of points and obtaining neighbor relationship between generators based on the idea of distance transformation. After map rasterization, all spatial features are translated into grid points. In the raster metric space, points, lines, and polygons are processed at spatial raster. All diagrams can be represented by a discrete grid lattice of size $N \times N$, which gives N^2 points in the plane. A unique identifier was placed in each grid cell. But there have some difference between three situations. Our approach focus on finding the raster point sets of polygons. Taking polygon feature as an example, our method increases the process of edge points extraction which is based on traditional Voronoi diagrams generation for points.

3.1 Raster-Based Weighted Voronoi Diagram Generation

After rasterization of primitive data, complex generators, such as polygons, can translate into lots of grid cells. Then, it makes huge computational burden and increases the time, especially if a large number of features are involved for generating Voronoi diagrams. While calculating the distance, we only need to get the minimum distance between polygons and other points. On the other hand, assume area features without internal voids, let a raster point specifies any grid cell except generators, the minimum distance can be the distance between the point and

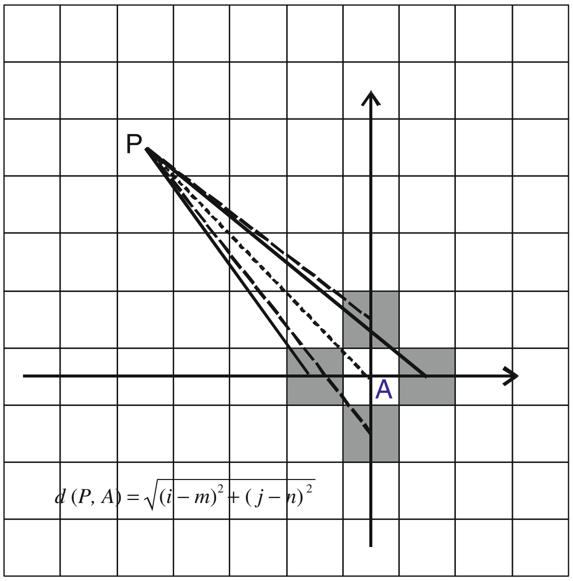

Fig. 1 Distance calculation from point to polygons

generator edge. Therefore, we need to get a simplified treatment of polygons to shorten the calculation time.

Edge points extraction of polygon namely judge whether a raster point is a boundary point or not. Our proposed approach uses a simple rule of criteria for determining pixels based on judging its four-neighbor points. If there are blank cells, the raster point can be a boundary point, otherwise delete it (Fig. 1).
In the raster space, point $P(m, n)$ is any other blank cell and $A(i,j)$ is one raster point; the shadow part is its four-neighbor. If its four-neighbor is all not blank cell, so point A is not a boundary point, then the distance between $P(m, n)$ and $A(i,j)$ cannot be the minimum distance between the five distances. So it is unnecessary to compute the distance between internal points (like A) and blank cells. We just need compute the distance between boundary points and blank points.

The steps of the method are as follows:

Step 1. Rasterization of primitive data. For every area generator, use a limited number of raster points to approach its region. And other regions are translated into blank cells.

Step 2. Edge points extraction of polygon. Traverse all raster points of generators. Judge its four-neighbor points are blank cells or not. If it is, mark the point as a boundary point and record it.

Step 3. Computing the weight distance and deciding each grid's character. Calculate the weighted distance between every blank cell and set of boundary points. Label the character of boundary point of nearest distance until values of all blank cells are determined.

Step 4. Output the final matrix.

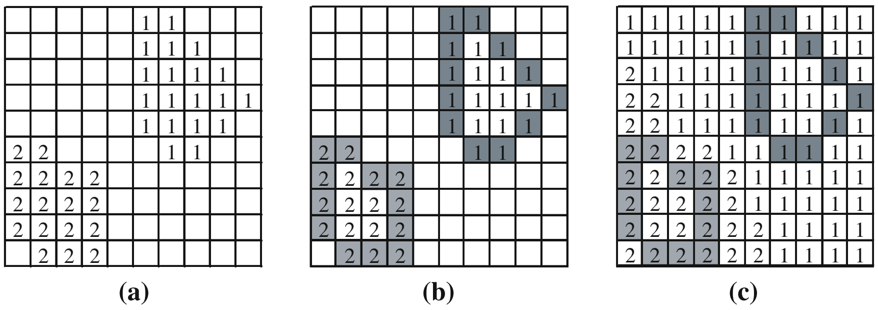

Fig. 2 Weighted Voronoi diagrams generation for polygons

Taking a discrete grid lattice of size 10×10 as an example, Fig. 2 shows the evolution process of generating WVD for polygons. There are two area features as generators which are marked by 1 and 2 (Fig. 2a). After rasterization, they translate into 37 cells. The distance between P and polygon 1 can be represented by $D_1(p, p_1) = d(p, p_1)/\omega_1$. To get $\min\{D_1(p, p_1), D_2(p, p_2)\}$, there would be 37 calculation times. After edge points extraction of polygon (Fig. 2b), the times of computation is 24. The computation cost is greatly reduced by using this method. Let $\omega_1 : \omega_2 = 3 : 2$.

3.2 MapReduce Parallelization

MapReduce programming model is simply represented in two functions, namely a map function and a reduce function. The MapReduce job processes a key/value pair to generate a set of intermediate key/value pairs in the map function, while merges all intermediate values associated with the same intermediate key [18]. The two functions are written by the user. It makes programmers design parallel and distributed applications easily. In our Hadoop implementation, the generation of WVD is implemented in one MapReduce job (Tables 1 and 2).

The task of map function is to determine the minimum distance between blank cells and generators and deciding each grid's character. We choose two files as input data, one is the original raster data file, another records boundary points. The map function takes as input a $<i, recordLine>$ pair in which i is the line number of the original matrix. Every *recordLine* deposit a row of grid cells. The map function must read record file for getting the location of generators in the original matrix. Finally, it outputs the $<i, newLine>$ pair as intermediate output. The output value deposits a row of grid's character.

In reduce function, the intermediate results are merged, sorted, and summed to output the final matrix. It takes as input a $<i, recordLine>$ pair. The output of the function is the same as the input. Because the MapReduce model has ranked the record lines following keys in map function, so reduce function only need to output the final data.

Table 1 Algorithm: VoronoiMapper (key, value)

```
Input: $<key, value>$ pair, where key is the line number, and value is the content of a row
of original matrix.
Output: $<key', value'>$pair, where key is the line number, and value is the content of a row
of grid's character.
Map(Text Key , Text value){
4   Read the original dataset, where the point is expressed by a triple group-
    $p[i](i, j, value)$ with row, column and value.
5   Get the record file of boundary points, every point is represented by $obj[i](ID, i, j, w)$,
    which has four attributes: ID, row, column, and weight.
6   for i=0 to row number
7     **min_distance**=$d_\omega(p[m], obj[0])$;
8     for j=0 to column number
9     calculate the weight distance $d_\omega(p[i], obj[j])$;
10    if( $d_\omega(p[i], obj[j])$<**min_distance**) then
11        do **min_distance**= $d_\omega(p[i], obj[j])$;
12        $p[i].value = obj[j].ID$;
13        record the row $obj[j].i$ and column $obj[j].j$ ;
14    else ( $d_\omega(p[i], obj[j])$==**min_distance**)
15      then $p[i].value = obj[j'].ID$ ;($obj[j']$ is expressed as the generator which has
    smaller coordinate)
16  end if
17  end for
18  output ($key'$ , $value'$);}
```

Table 2 Algorithm: VoronoiReducer(key, value)

```
Input: $<key, value>$ pair, where key is the line number, and value is a string of the
intermediate results.
Output: $<key', value'>$pair, where key is a null string, and value is a string of the intermediate
results.
Reduce(Writable key , Iterator<Text> value){
19  Initiate string $key'$ as a null string.
20  output ($key'$ , $value'$);}
```

3.3 Analysis

In practice, the MapReduce implementation of the method runs on multiple machines in parallel. When discussing the method complexity, we consider the fact that it runs in parallel and discuss the parallel method complexity. In the MapReduce job, the computational complexity of the associated reduction is as follows:

$$(m \times n - g_k) \times g_k \times O/\text{nodes}$$

where $m \times n$ is the total number of grid cells, k is the number of generators, g_k is the total number of boundary points after edge points extraction of polygon, nodes is the number of the Hadoop nodes, and O is the computational complexity of weight distance computing.

4 Results and Discussion

In this section, we evaluate the performance impact of algorithm implementation and not its accuracy on our cluster system. The core idea of Apache Hadoop [19] is the MapReduce programming model. Our experimental hardware consists of nine nodes cluster: one namenode and eight datanode. Each node in the cluster is equipped with four quad-core 3.10 GHZ Intel Core(TM) i5-2400 processors, 4 GB of memory and 500 G of disk, runs Fedora15, and is connected with fast Ethernet. In this chapter, all experiments described are obtained using Hadoop version 0.20.2 and Java 1.7.0.04, while the data are stored with two replicas per block in HDFS.

4.1 Single Machine Environment Versus Hadoop Pseudo-Distributed Environment

When there are same data scales and same hardware configuration environments, we compare the time of generating WVDs under single machine environment and Hadoop pseudo-distributed environment. From the experimental results, the algorithm running in the single machine environment needs less time when the data scale is small. But when the scale of data increases to a certain extent, it reports out of memory and cannot complete calculation tasks, where the tasks can be treated successfully under Hadoop pseudo-distributed environment.

In our analysis, the control between nodes and task schedule take most part of resources when there is a small scale of data. So the time of calculation tasks is longer. When the scale of data increases, the single machine environment cannot meet the demand of computing because of many reasons such as the growth of memory resource consumption. However, the Hadoop platform can easily handle large datasets (Table 3).

4.2 Experiment Analysis for Cluster System

In the following experiments, we choose three gigabit-scale datasets as original datum: DS1, DS2, and DS3. The datasets are described in detail in Table 4.

Table 3 Contrast of execution time between single and parallel systems

Times	File size/MB	Record lines	Serial τ_1/s	Hadoop τ_2/s
1	2	1,750	1.716	22
2	6	5,000	4.805	26
3	24	20,000	18.548	43
4	45	37,450	Out of memory	61
5	90	75,000	Out of memory	73

Table 4 Experimental datasets

Datasets	Size of original data/MB	Record lines/10^5	Polygons	Numbers of boundary points	Data blocks
DS1	1,116.4	9.3	3	5,786	18
DS2	2,233.6	18.6	3	5,786	35
DS3	4,468.4	37.2	3	5,786	70

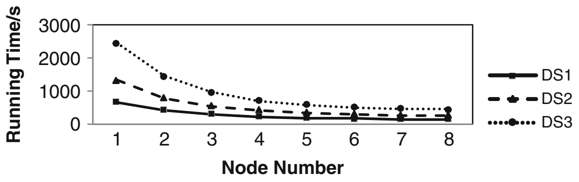

Fig. 3 Running time

In light of three datasets, kill the certain quantity of datanodes every time and the running time of each experiment is shown in Fig. 3. We can see that the time decreased with the growth of node numbers. Increasing the number of nodes can significantly improve the processing ability of the cluster when the data scale is the same. The running speed is similar to linear growth with the increase of data nodes. It shows that the speed of generating WVDs for polygons is increased markedly on Hadoop distributed environment.

5 Case Study

In the former section, we verify the performance efficiency of parallel algorithm. Now we make a trial to use our approach in some practical application. Because of the limit of experiment condition, we only choose some part of maps as original map data. Our original datasets are the raster data are obtained by map rasterization using ArcGIS software.

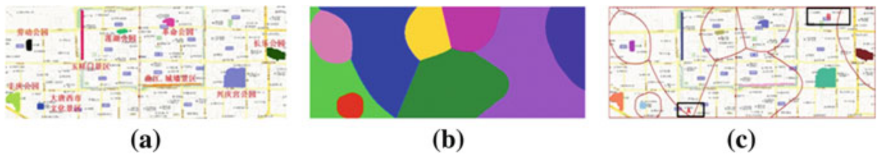

Fig. 4 WVDs for polygons of greenbelts

5.1 *Case One: Green Space Planning*

According to the green space planning in Xi'an city, the city will vigorously develop the green space system to become a national garden city. The city greenbelt element can appear as green space blocks in various shapes and size. Park, shelter-forest, and water conservation district are strip or area features. The affection of greenbelts on the environment depends on varied factors such as area, size, tree species, and purification ability. Now we abstract greenbelts into polygons and assign generators' weight. Then generate the WVDs for greenbelts for the decision of green construction.

Our chapter takes greenbelt in Xi'an as an example and analyzes the sphere of influence of greenbelts. Fig. 4a shows the original map data from part area of Xi'an city. It chooses nine greenbelts as generators. Set weight according to the size of greenbelts, we can get the raster WVD in Fig. 4b. Figure 4c is the final vector diagram. It marks off the sphere of influence for every greenbelt. Regions A and B are located at the interface of several greenbelts and in a position of the edge. They are weakly influenced by the greenbelts. So we suggest increasing greenbelt in these regions.

5.2 *Case Two: Optimal Path Planning Problem*

Given some obstacles in space or in a plane, the retraction method for motion planning uses the Voronoi diagram to determine whether there exists an optimal path from an initial posit onto a final position. If obstacle can be approximately regarded as particle, the safest path can follow Voronoi edges. If obstacle cannot be approximately expressed by particle, the expansion of Voronoi diagram can be employed (Generators are lines, polygons, or polyhedrons). On the other hand, for different obstacles, they may have different criticality. So their weights are different. For example, when obstacles are contaminated zones or danger, the path need to be away from them. So, their weights should be smaller than safe obstacles. Our approximate fast algorithm computes the WVDs for polygons to get the Voronoi edges, which are the final optimal path. Figure 5a gives distribution of some obstacles with different weights. Then, we will get the optimal path by following the Voronoi edges (Fig. 5b).

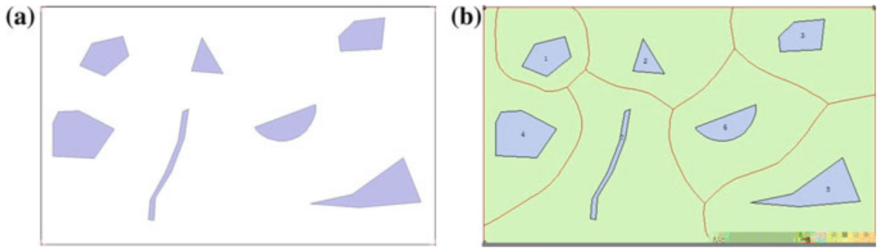

Fig. 5 Optimal path generation of obstacles

6 Conclusion and Future Work

The chapter presents a parallel approach for generating weighted raster-based Voronoi diagrams on Hadoop platform. The presented approach is based on traditional distance transformation and MapReduce model. Considering type and weight of generators, distance computation is simplified by extracting edge points of polygon. The experiments show that the approach significantly improves the performance with a linear scale-up in response to the increase in nodes. Application cases show that the approach is successfully applied in urban green space planning. Some further work can also be expected with the presented method: selecting reasonable weights for different cases and applying the algorithm to applications such as data collection in sensor networks, emergency modeling, and Voronoi-based geospatial query.

Acknowledgments The authors would like to thank the National Natural Science Foundation of China (No. 41271387), the tourism constructing study soft science project of Xian city (No. SF1228-3), and the academician innovation project of Shaanxi Normal University (No. 999521) for the support given to the study.

References

1. Lauro, C.G., Antonio, G.N., Novaes, J.E., Souza, D.C., João, C.S.: A multiplicatively-weighted Voronoi diagram approach to logistics districting. Comput. Oper. Res. **33**, 93–114 (2006)
2. Hu, P., You, L., Hu, H., et al.: Voronoi diagram. In: An Introduction to Map Algebra. Surveying and Mapping Press, China (2008)
3. Wang, X.S., Li, Q., Guo, Q.S., Wu, H.H., Fu, F.Y.: The generalization and construction of Voronoi diagram and its application on delimitating city's affected coverage. J Central China Normal Univ. (Nat. Sci.), **36**(1), 107–111 (2002)
4. Zhang, Y.H.: Voronoi diagram of weighted segments. Chin. J. Comput. **18**(11), 822−829 (1995)
5. Dong, R.: The discrete construction algorithm of line segment weighted Voronoi diagram. Hebei Normal University, Hebei (2006)

6. Dong, R., Zhang, Y.H., Liu, S.J., Gong, X.Y., Wang, D.D.: On discrete construction algorithm of line segment weighted Voronoi diagram and its realization. Comput. Appl. Softw. **26**(7), 245−247 (2009)
7. Wu, X.J., Luo, X.F.: The algorithm for creating weighted Voronoi diagrams based on cellular automata. In: Proceedings of the 6th World Congress on Intelligent Control Automation, EI (2006)
8. Dong, P.L.: Generating and updating multiplicatively weighted Voronoi diagrams for point, line and polygon features in GIS. Comput. Geosci. 3**4**, 411–421 (2008)
9. Fan, X.W.: The vector construction algorithm and implementation of weighted Voronoi diagram. Northwest University, China (2011)
10. Gong, Y.X., Li, G.C., Tian, Y., et al.: A vector-based algorithm to generate and update multiplicatively weighted Voronoi diagrams for points, polylines, and polygons. Comput. Geosci. **42**, 118–125 (2012)
11. Rong, G., Liu, Y., Wang, W., Yin, X., Gu, X., Guo, X.: GPU-assisted computation of centroidal Voronoi tessellation. IEEE Trans. Vis. Comput. Graph. **17**(3), 345–356 (2011)
12. Xu, Z.H., Kong, D.H., Xiao, X.F.: GPU-based weighted Voronoi diagram computing. J. Syst. Simul. (20), 29−32 (2008)
13. Afsin, A., Ugur, D., Farnoush, B.K., Cyrus, S.: Voronoi-based geospatial query processing with MapReduce. In: Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science, 9–16 (2010)
14. Kei, K., Kokichi, S.: Crystal Voronoi diagram and its applications. Future Gener. Comput. Syst. **18**, 681–692 (2002)
15. Barry, B., South, R.: Modeling retail trade areas using higher-order, multiplicatively weighted Voronoi diagrams. J. Retail. **73**(4), 519–536 (1997)
16. Atsuyuki, O., Atsuo, S.: Locational optimization problems solved through Voronoi diagrams. Eur. J. Oper. Res. **98**, 445–456 (1997)
17. Ickjai, L., Christopher, T.B., Kyungmi, L.: Map segmentation for geospatial data mining through generalized higher-order Voronoi diagrams with sequential scan algorithms. Expert Syst. Appl. **39**, 11135–11148 (2012)
18. He, Q., Shang, T.F., Zhuang, F.Z., Shi, Z.Z.: Parallel extreme learning machine for regression based on MapReduce. Neurocomputing **102**, 52–58 (2013)
19. Zhang, J.B., Li, T.R., Ruan, D., Gao, Z.Z., Zhao, C.B.: A parallel method for computing rough set approximations. Inf. Sci. **194**, 209–223 (2012)