Available online at www.sciencedirect.com

SCIENCE DIRECT®

Neural Networks 19 (2006) 900–910

Neural Networks

www.elsevier.com/locate/neunet

2006 Special Issue

Spherical self-organizing map using efficient indexed geodesic data structure

Yingxin Wu[a,*], Masahiro Takatsuka[b]

[a] *School of Information Technologies, The University of Sydney, Australia*
[b] *ViSLab, The University of Sydney, Australia*

Abstract

The two-dimensional (2D) Self-Organizing Map (SOM) has a well-known "border effect". Several spherical SOMs which use lattices of the tessellated icosahedron have been proposed to solve this problem. However, existing data structures for such SOMs are either not space efficient or are time consuming when searching the neighborhood. We introduce a 2D rectangular grid data structure to store the icosahedron-based geodesic dome. Vertices relationships are maintained by their positions in the data structure rather than by immediate neighbor pointers or an adjacency list. Increasing the number of neurons can be done efficiently because the overhead caused by pointer updates is reduced. Experiments show that the spherical SOM using our data structure, called a GeoSOM, runs with comparable speed to the conventional 2D SOM. The GeoSOM also reduces data distortion due to removal of the boundaries. Furthermore, we developed an interface to project the GeoSOM onto the 2D plane using a cartographic approach, which gives users a global view of the spherical data map. Users can change the center of the 2D data map interactively. In the end, we compare the GeoSOM to the other spherical SOMs by space complexity and time complexity.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Spherical SOM; Geodesic dome; Data structure; GeoSOM; Border effect

1. Introduction

The Self-Organizing Map introduced by Kohonen (2001) is a popular tool for high-dimensional data analysis and visualization. Its applications can be found in various research areas such as social science (Lin, White, & Buzydlowski, 2003), web-document mining (Lagus, Kaski, & Kohonen, 2004) and robotics (Ritter, Martinetz, & Schulten, 1992). In a conventional SOM, the neighborhood relationship is defined by a two-dimensional rectangular or hexagonal lattice. During the training phase, all neurons compete with each other for the input signals. The winner and its neighbors update their connection weights. Ideally, at the end of the training, all samples are equally represented by neurons and the neighboring units in the grid tend to model similar regions of the data space. However, the grid units at the boundary of the SOM have fewer neighbors than the units inside the map, so they have fewer chances of being updated. This often leads to the "border effect" — weight vectors of these units "collapse to the center of the input space" (Sarle, 2002). Several approaches have been suggested to solve the problem, such as the heuristic weighting rule method by Kohonen (2001) and local-linear smoothing by Wand and Jones (1995). Aside from these mathematical solutions, a straightforward method is to remove the boundaries from the grid by implementing the SOM on a torus (Ito, Miyoshi, & Masuyama, 2000; Li, Gasteiger, & Aupan, 1993) or a sphere (Boudjemaï, Enberg, & Postaire, 2003; Hirokazu, Altaf-Ul, Ken, Kotaro, & Shigehiko, 2005; Nakatsuka & Oyabu, 2003; Sangole & Knopf, 2003).

Compared to a toroidal SOM, the spherical SOM is more visually effective. Firstly, the area associated with each neuron varies significantly on the surface of a torus: larger around the outer circle and compressed near the inner circle. For a SOM, it is desirable to have all neurons receive equal geometrical treatment. Secondly, the toroidal SOM fails to provide an intuitively readable map. People are usually more familiar with maps generated from a sphere, such as the world maps, than they are with maps based on a torus.

Several spherical SOMs have been implemented and applied to different types of data sets. Using a tessellated platonic polyhedron as a lattice was first suggested by Ritter (1999). He also pointed out that the spherical SOM would be very suitable

* Corresponding author. Tel.: +61 2 83745583; fax: +61 2 83745527.
E-mail addresses: chwu@it.usyd.edu.au (Y. Wu), masa@vislab.net (M. Takatsuka).

0893-6080/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2006.05.021

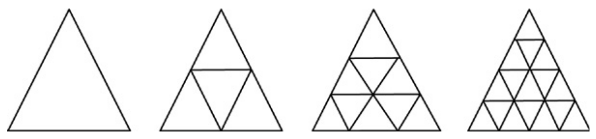


Fig. 1. Left to right: one-frequency, two-frequency, three-frequency and four-frequency tessellation.

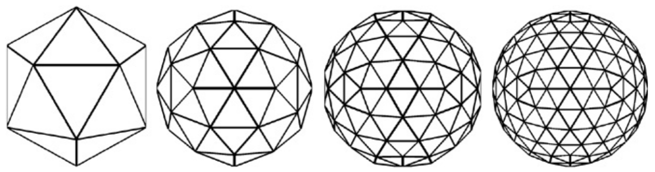


Fig. 2. Left to right: one-frequency, two-frequency, three-frequency and four-frequency icosahedron.

for data with underlying directional structures. In Sangole et al.'s work (Sangole & Knopf, 2003), a spherical SOM was employed in three-dimensional (3D) immersive virtual reality environments for interactive data analysis. Boudjemaï et al. (2003) applied the spherical SOM in 3D object modeling. While these applications have been successful, existing data structures for geodesic domes are either not space efficient or are time consuming when finding the neighbors. In this paper, we present a 2D data structure to store the spherical lattice. It reduces the overhead in maintaining the spherical grid structure. Finding the immediate neighbors of a vertex (neuron) is efficient because it is indexing in a 2D array. It also supports fast dome tessellation (increasing the number of neurons). We call a spherical SOM using this data structure a GeoSOM.

The remainder of this paper is organized as follows. Section 2 reviews existing techniques for spherical SOMs using lattices of geodesic domes. Details of our data structure are presented in Section 3. Section 4 describes the interface that we developed to visualize the GeoSOM. In Section 5, we test our data structure on two data sets. Results are compared with the corresponding 2D SOMs. In Section 6, GeoSOM is compared analytically with the existing spherical SOMs. Conclusions and future work are presented in Section 7.

2. Spherical SOMs based on geodesic domes

2.1. A brief discussion of geodesic domes

For a 2D SOM, a hexagonal lattice is preferable to a rectangular one as it is more uniform: every grid unit has the same number of immediate neighbors and the distances between a unit and its immediate neighbors are the same. However, such uniformity cannot be achieved on the sphere except for the five platonic polyhedra — tetrahedron, cube, octahedron, icosahedron and dodecahedron (Pugh, 1976). These polyhedra can be further tessellated into different frequencies — the geodesic domes. The tessellation method was introduced by Fuller (1975). Each triangular face of the polyhedron is subdivided into several smaller triangles by lines running parallel to the original edges of the triangles. Fig. 1 shows the tessellation of a triangular face. Here, frequency means the number of parts into which the original edges are divided. For the polyhedra whose faces are not triangles, such as a cube and dodecahedron, their faces need to be triangulated first.

The geodesic domes created by such a process will not have uniform characteristics. Rather, the length of the edges will vary and vertices will have different numbers of immediate neighbors. Among the five platonic polyhedra, the icosahedron is most similar to the sphere. After tessellation, it always has the smallest variance in edge length. Most of the vertices have six immediate neighbors. Only the original 12 vertices of the icosahedron have five immediate neighbors. This makes the icosahedron-based geodesic dome more suitable for a spherical SOM. The number of vertices after tessellation can be calculated as $N = f^2 * 10 + 2$, where f is the frequency (Hoffmann, 2002). Fig. 2 shows the icosahedron in different frequencies.

2.2. Existing data structures for the geodesic domes

When an input signal is presented to an SOM, the winning neuron and its neighbors within a certain radius update their weight vectors. This process is carried out for a number of iterations until the SOM converges. Neighbor neurons are located as a result of a neighborhood search, which is a time-consuming process, especially when the number of neurons is large. A conventional SOM places its neurons in a 2D rectangular or hexagonal lattice which is stored in 2D array data structure. Neighbors of the winning neuron can be located very quickly by simple 2D array indexing. The lattice of a geodesic dome is more complex. The data structure for such a lattice should support fast vertex indexing in order to avoid considerable performance loss.

In Sangole et al.'s spherical SOM (Sangole & Knopf, 2003), the distance between two neurons on the geodesic dome was defined to be the length of the shortest path connecting them. Every grid unit maintains a list of its immediate neighbors. To find out all neighborhood units of the winning neuron, the immediate neighbors are read from the winning neuron's neighborhood list. It is followed by the execution of the same procedure on the selected neighbor neurons until the target level is reached. Boudjemaï et al. (2003) used a similar scheme by treating the geodesic dome as a 3D graph and storing it in an adjacency list. The execution speed of this method is slower compared to looking for neighbors in a 2D array. Sangole et al. tackled this problem by finding out all different distance neighbors for every neuron before the training. Although this pre-processing sped up the training, it required an extra $O(N^2)$ space to store the neighbor pointers, where N is the number of vertices. Another disadvantages is that if we further tessellate the geodesic dome to create more neurons, then all the neighbor pointers and the whole adjacency list need to be recursively modified.

In Nakatsuka's work (Nakatsuka & Oyabu, 2003), the distance between two neurons are defined as the angle between

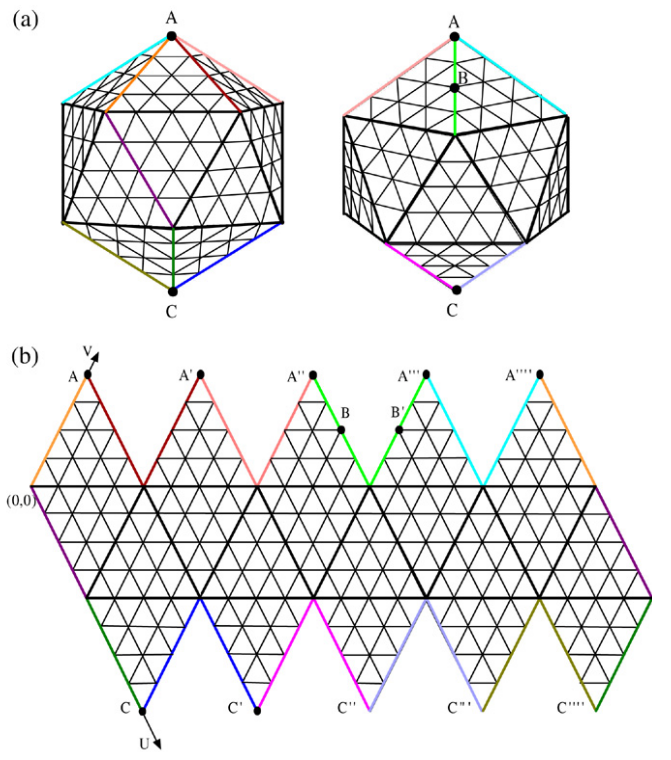


Fig. 3. A four-frequency icosahedron opened onto a 2D plane. (a) Front and back views of the geodesic dome. The colored edges indicate where the dome is cut open. (b) The opened four-frequency geodesic dome.

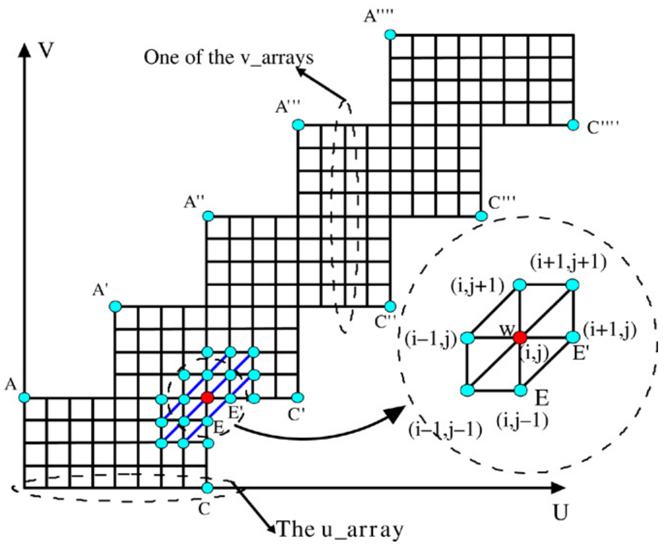


Fig. 4. The 2D data structure for icosahedron-based geodesic domes.

their location vectors on the sphere. The neighborhood size is decided by a neighborhood function. Any neurons with a value of neighborhood function bigger than zero are updated. Using this method, extra floating-point calculations are required to decide whether a neuron needs to be updated or not.

3. Rectilinear grid data structure for the icosahedron-based geodesic dome

3.1. Opening the icosahedron-based geodesic dome

An icosahedron-based geodesic dome can be opened onto the 2D plane. Fig. 3 illustrates the process. Note that, for clarity, we did not project the vertices onto the surface of the sphere. Thicker lines are the original edges of the icosahedron. The 12 vertices of an icosahedron can be grouped into six pairs. Vertices in each pair are opposite to each other on the sphere and can be considered as two poles, such as vertices A and C. The edges marked with colors indicate where the dome is cut open. Fig. 3(b) shows the geodesic dome after opening. Note that in the 2D grid, vertices on the cut edges are duplicated. Most of them are duplicated only once, such as point B and B′. Only the two poles A and C are duplicated four times.

3.2. The 2D data structure

We define two axes U and V on this 2D lattice. Each vertex can then be uniquely denoted by the two-dimensional coordinate pair (u, v). By rotating all the vertices on the lattice such that the two axes U, V are orthogonal to each other, we obtain a 2D matrix to store all the vertices of the geodesic dome (see Fig. 4). In our program, vertices sharing the same u coordinate are stored in a one-dimensional array, called a *v_array*, in ascending order along the v coordinate. Every *v_array* is contained within another one-dimensional array, called the *u_array*, in ascending order along the u coordinate. Note that only the vertices at the boundaries are duplicated. Therefore the number of duplicated vertices will be very small compared to the total number of vertices.

3.3. Searching and updating the neighborhood on the geodesic dome

Like the method proposed by Boudjemaï et al. (2003), we treat the geodesic dome as a 3D graph. The distance between two vertices on the dome is defined to be the minimum number of edges connecting them. Neighbors of a vertex are grouped into levels according to their distances to the vertex. In order to find all neighbors within a specified update radius, an iterative search process needs to occur, starting at the first-level neighbors until the target level is reached.

Each point in the 2D geodesic dome lattice has six first-level neighbors. For point $W(i, j)$ in Fig. 4, the first-level neighbors can be obtained by the indices: $(i, j+1)$, $(i+1, j+1)$, $(i+1, j)$, $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$. As mentioned before, there are duplicated points on the lattice's boundaries and they represent the same weight vector. For example, in Fig. 4, points E and E′ are the same point on the dome. Only one of them should be included as W's immediate neighbor or the same weight vector will be updated twice. In the data structure, duplicated vertices will keep a list of their duplicated points. If one of them is already included, others are marked and will not be included again. In the case where the winning neuron itself has duplicated points, such as point A in Fig. 4, all the neighbors of its duplicated points (from A′ to A′′′′) need to be searched and updated. Algorithm 1 is the pseudo-code for updating a winning neuron's neighborhood on the data structure.

Algorithm 1 Find and update the wining neuron and its neighbors within a given radius r

```
update winning neuron's weight vector;
find the first-level neighbors of the winning neuron;
insert the neighbors into list $A$;
initialize an empty list $A_1$;
for $l = 1$ to $r$ do
    for each neuron $n$ in $A$ do
        if $n$ is visited then
            continue;
        end if
        update the $n$'s weight vector according to $l$;
        mark $n$ and its duplicated points as visited;
        find the first-level neighbors of $n$;
        insert the neighbors into list $A_1$;
    end for
    $A \Leftarrow A_1$;
    $A_1 \Leftarrow \emptyset$;
end for
unmark the visited neurons
```

3.4. Tessellating the geodesic dome

Increasing a dome's frequency involves inserting new vertices into the original edges. Using our data structure, the tessellation process becomes straightforward. Suppose that we have a f_1 frequency geodesic dome, the $f_1 * f_2$ frequency dome can be obtained by inserting $(f_2 - 1)$ new vertices between every pair of adjacent vertices into the data structure.

Fig. 5 illustrates how to tessellate the icosahedron into a three-frequency geodesic dome. Suppose that ABD and BCD are two adjacent triangles on the icosahedron. A, B and C, D are stored in v_*arrays* $u1$ and $u2$, respectively. In order to tessellate the two triangles into three-frequency, we first calculate the vertical trisection point of edges AB and CD and insert them into $u1$ and $u2$ between vertices A, B and C, D. Then we go on to calculate the horizontal trisection points of edges AD and BC. Two new v_*arrays* $u3$ and $u4$ are inserted between $u1$ and $u2$. After that, new vertices between the diagonal adjacent points, such as point E and point F on edge BD, are calculated. All the other spherical triangles in the geodesic dome can be tessellated in the same way. Note that, for clarity, the new generated vertices in Fig. 5 are drawn on the surface of the icosahedron. In our program, all the new vertices' coordinates will be normalized so that they lie on the surface of a unit sphere.

After the above process, some new duplicated points will be created along the boundaries of the map. Therefore, we need to scan along the map boundaries to find all the duplicated points. Algorithm 2 is the pseudo-code for dome tessellation. Note that, in doing the tessellation, we only need to calculate the new vertices and insert them into the right place of the data structure. Since the vertices relationships are maintained by their positions in the data structure, no additional adjacency matrix or neighbor pointers needs to be updated.

Algorithm 2 Increase frequency of the geodesic dome by f

```
// calculate vertical section points
for every v_array $u[i]$ do
    for $j = 0$ to $u[i].length - 2$ do
        calculate $f - 1$ new vertices between $u[i][j]$ and $u[i][j+1]$;
    end for
end for
//calculate horizontal section points
for every two adjacent v_arrays $u[i]$ and $u[i+1]$ do
    $j = 0$;
    while $j < u[i].length$ do
        calculate $f - 1$ new vertices between $u[i][j]$ and $u[i+1][j]$;
        $j = j + 1$;
    end while
end for
//calculate diagonal section points
for every two v_arrays $u[i * f]$ and $u[(i+1) * f]$ do
    $j = 0$;
    while $j < u[i].length - f$ do
        for $k = 1$ to $f - 2$ do
            calculate new vertices between $u[i * f][j + k]$ and $u[(i+1) * f - k][j + f]$;
            calculate new vertices between $u[i * f + k][j]$ and $u[(i+1) * f][j + f - k]$;
        end for
        $j = j + f$;
    end while
end for
scan the boundaries of the new map to find duplicated points;
```

4. Projecting the GeoSOM onto 2D plane

Usually, a spherical SOM is visualized as a 3D spherical object (see Fig. 6). The surface of the sphere is deformed to reflect the variation in weight vectors. Since our flat screens cannot show the front and back parts of a spherical object at the same time, interfaces such as rotation by mouse are provided so that the users can view every part of the SOM. However, it is still difficult for the user to maintain an entire picture of the map. We implemented an interface to project the spherical image onto a 2D plane, so that the user can have a global view of the data map.

Fig. 7 illustrates how the interface works. The user selects two points on the sphere using a mouse. The first point A will be the center of the 2D projection. The second point B, together with A, defines a plane going through the sphere's center O. Suppose that $\vec{OC}$ is the plane's normal, then the sphere's central axis can be obtained from the cross product of $\vec{OC}$ and $\vec{OA}$. This axis intersects the sphere at two points N and S, which become the "North" and "South" poles of the sphere. The geodesic arc going through these two points and opposite to A becomes the split line on which to open the sphere.

Currently, we choose the Wagner III pseudo-cylindrical projection to transform the spherical surface onto a 2D plane. This

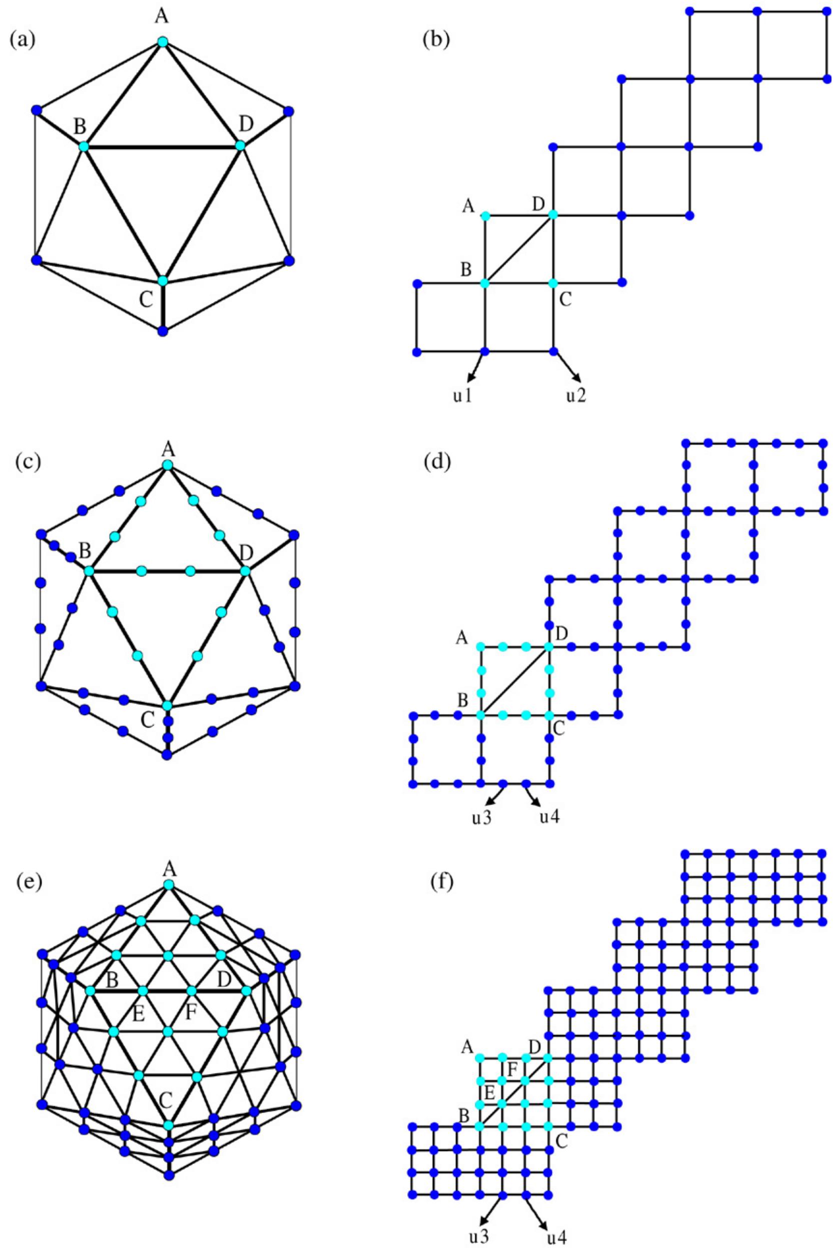


Fig. 5. Tessellate an icosahedron into a three-frequency geodesic dome. First row: two adjacent triangles ABD and BCD on the icosahedron and in the data structure. Second row: calculate and insert the trisection points between the original edges. Third row: calculate and insert the diagonal section points.

projection is neither equal-area nor conformal, but it has less extreme distortions in terms of shape and area. Therefore it can give a more balanced representation of the spherical map's main features (Canters & Decleir, 1989). The equations to convert a point on the sphere (θ, ϕ) to a point on a 2D plane (x, y) are:

$$x = R\frac{\beta\theta}{\sqrt{\alpha\beta}}\cos(\alpha\phi) \tag{1a}$$

$$y = R\frac{\alpha\phi}{\sqrt{\alpha\beta}} \tag{1b}$$

$$\alpha = \frac{2\arccos c}{\pi} \tag{1c}$$

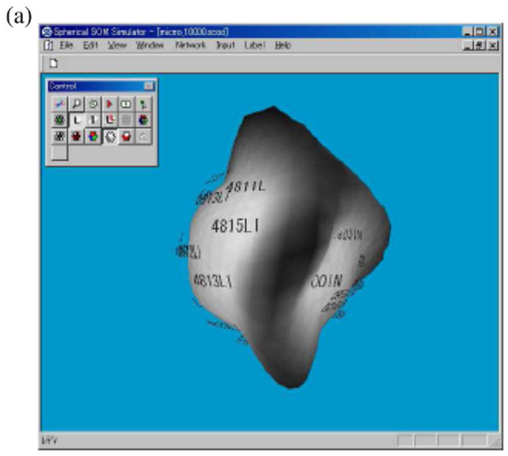
$$\beta = \frac{\alpha}{2p} \tag{1d}$$

where θ is the longitude and ϕ is the latitude. R is the radius of the sphere, which is 1 in our program. $c = 0.5$ and $p = 0.5$ are two constants controlling the shape of the projection.

Using our interface, users can choose any point on the GeoSOM to be the center of the 2D map. They can examine the vicinity of the data that they selected while having a global view of the map. Changing the center or the orientation only requires re-projection. This is not possible if a 2D SOM is used, especially when the data of interest are mapped to the boundaries.

5. Experiments

In this section, we conduct two sets of experiments on GeoSOMs using a synthetic data set and a breast cancer data



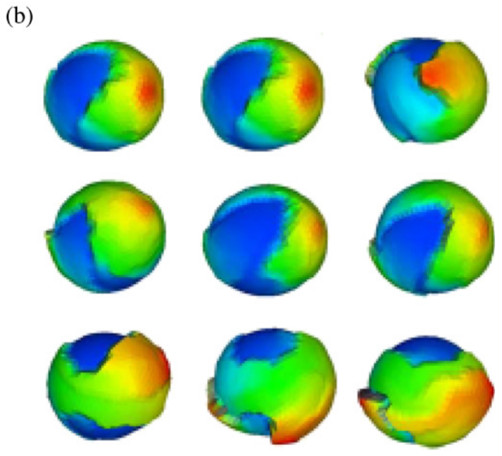


Fig. 6. (a) Spherical SOM which visualizes the microarray data (Nakatsuka & Tokutaka, 2004). (b) Spherical SOM which visualizes the data points generated by Lozi attractor function (Sangole & Knopf, 2003).

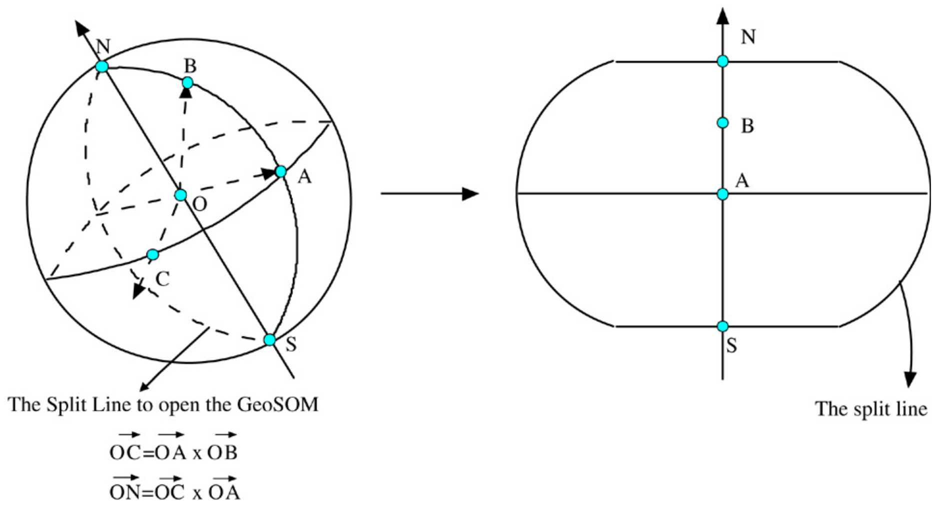

N
B
A
O
C
S
The Split Line to open the GeoSOM
$\overrightarrow{OC}=\overrightarrow{OA} \times \overrightarrow{OB}$
$\overrightarrow{ON}=\overrightarrow{OC} \times \overrightarrow{OA}$
N
B
A
S
The split line

Fig. 7. The user selects two points on the sphere. The first point A will be the center of the projection. The second point B, together with A, defines the central axis and the split line on which to open the sphere.

set. The experiments are carried out on a Pentium IV 3.0 GHz workstation with 1 GB of memory. Results are compared with the corresponding 2D hexagonal SOMs generated by SOM_PAK (Kohonen, Hynninen, Kangas, & Laaksonen, 1996). Parameters in training the two type of SOMs are set to be the same:

- Sizes of the GeoSOM and the 2D SOM are chosen to be as close as possible.
- Before the training, the components of the weight vectors are initialized to random values which are evenly distributed in the area of corresponding data vector components.
- During the training, both SOMs use the same learning rate, update radius and Gaussian neighborhood function. In each of the experiments, the initial learning rate is 0.8. The learning rate and update radius decrease linearly according to the training time.
- The original SOM_PAK updates every neuron whose value of neighborhood function is bigger than zero. We slightly modified it to only update the neurons within the update radius.

We also implemented a program to visualize the GeoSOM and the 2D SOM. In this visualization, the average distances between each neuron and its immediate neighbors are calculated and coded with colors. The colors change linearly from blue to cyan, yellow and orange. Blue denotes the smallest variance between the weight vectors, and orange the largest.

5.1. Modified average distortion measure

Besides running time, we compare the quality of the two SOMs using a modified average distortion measure E_d. The original definition of E_d is:

$$E_d = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{w} h_{b_i,j}\|x_i - w_j\|^2 \tag{2a}$$

$$h_{b_i,j} = \exp\left(-\frac{\text{dist}(b_i, j)^2}{2r^2}\right) \tag{2b}$$

where n is the number of inputs, w is the number of neurons, b_i is the best matching unit (BMU) of input x_i, $h_{b_i,j}$ is the

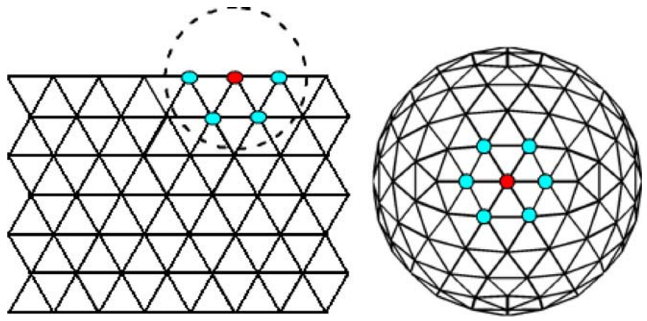


Fig. 8. For the neuron marked with red color, there are 7 neurons included within the circle of radius 1 on the GeoSOM. But only 5 neurons are included on the 2D hexagonal SOM. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

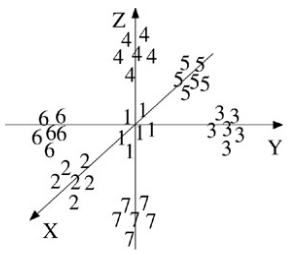


Fig. 9. The 3D synthetic data set.

neighborhood function, $\mathrm{dist}(b_i, j)$ is the distance between the BMU and neuron j on the grid, and r is the radius of the neighborhood. This measure has been proven to be the local energy function for SOM in the case of a discrete data set and fixed neighborhood function (Kohonen, 1991). Vesanto, Sulkava, and Hollmn (2003) showed that it can be decomposed into several components that evaluate the quantization error and topological preservation separately. Therefore, this measure is able to evaluate the overall quality of a SOM.

However, some changes are needed before applying it to compare the two SOMs. In the above equation, each input's distortion is calculated by summing the distances between it and all the other neurons in its BMU's neighborhood. The distances are weighted by the neighborhood function. Neurons close to the BMU play a more important part in the calculation. For input x_i mapped near the boundaries of the 2D SOM, fewer neurons are included within a specified radius of its BMU than on the spherical SOM, as shown in Fig. 8. Hence the original distortion measure would be biased towards the 2D SOM. We redefine an input's distortion to be the average distortion over its BMU's neighborhood. The distortion equation becomes:

$$E_d = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{w} h_{b_i,j}\|x_i - w_j\|^2}{\sum_{j=1}^{w} h_{b_i,j}}. \tag{3}$$

Using this equation, the results will not be affected by the different topologies of the grid. In the experiments, the neighborhood radius r in calculating the distortion is 1. Each experiment is repeated five times. The distortion values shown are averaged over all the trials.

5.2. The synthetic data set

We generated a three-dimensional data set which contains seven clusters, as shown in Fig. 9. Each cluster has 500 normally distributed data points. The standard deviation is 1 in each dimension. The centers of the clusters are: $(0, 0, 0)$, $(10, 0, 0)$, $(0, 10, 0)$, $(0, 0, 10)$, $(-10, 0, 0)$, $(0, -10, 0)$, $(0, 0, -10)$. The GeoSOM uses a nine frequency geodesic dome. It contains 812 neurons. The 2D SOM is trained on a 29×28 hexagonal grid and the number of neurons is also 812. For both SOMs, the initial update radius is 14.

The running time and average distortions at different epochs are shown in Fig. 10. Due to more complex neighborhood searching, the GeoSOM runs slightly more slowly than the 2D SOM. For this simple data set, both of the SOMs converge after 15 epochs of training. Values of the average distortion decrease as the training proceeds. The GeoSOM always achieves much lower distortion — less than two thirds of the 2D SOM's.

Fig. 11 shows the visualization of the GeoSOM and 2D SOM trained for 45 epochs. We evaluate the distortions at each neuron, and they are shown using spheres of different sizes. A larger sphere indicates a larger value. For both SOMs, high distortions occur mainly along the clusters' boundaries. However, the distortions at the borders of the 2D SOM are

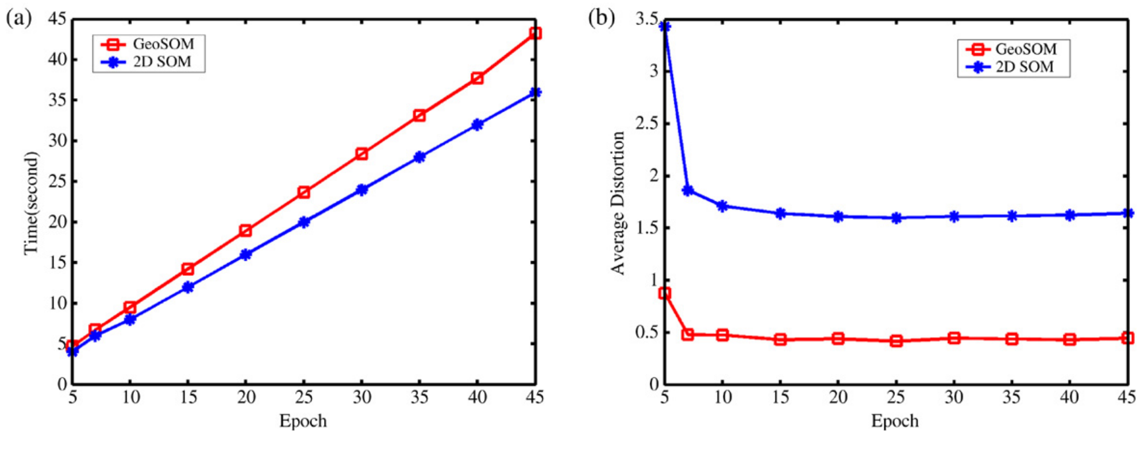


Fig. 10. Experimental results of the synthesis data set: (a) the running time; (b) the average data distortion at different epochs.

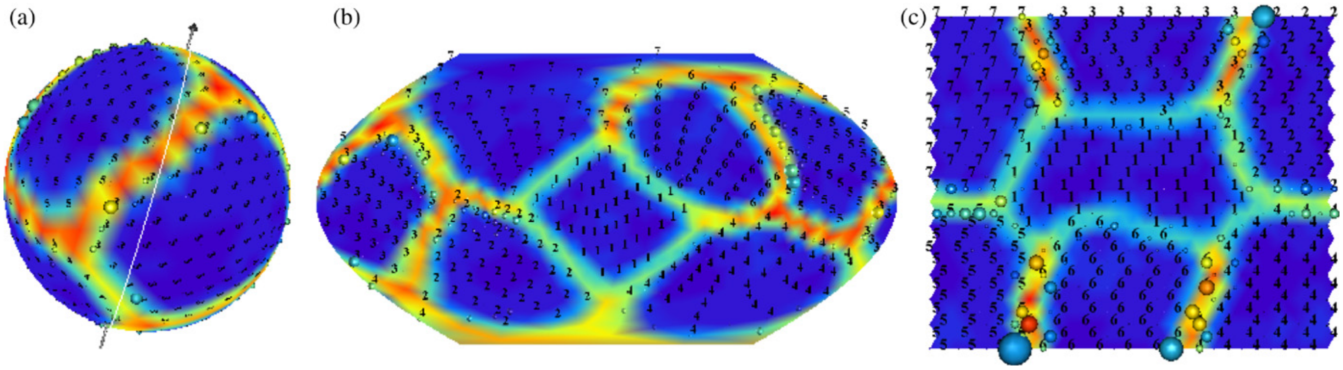


Fig. 11. (a) The GeoSOM trained with the synthetic data set for 45 epochs. The white line shows where it is cut open in projection. (b) Projection of the GeoSOM onto a 2D plane, choosing cluster 1 as the center. (c) The 2D SOM trained with the synthetic data set for 45 epochs.

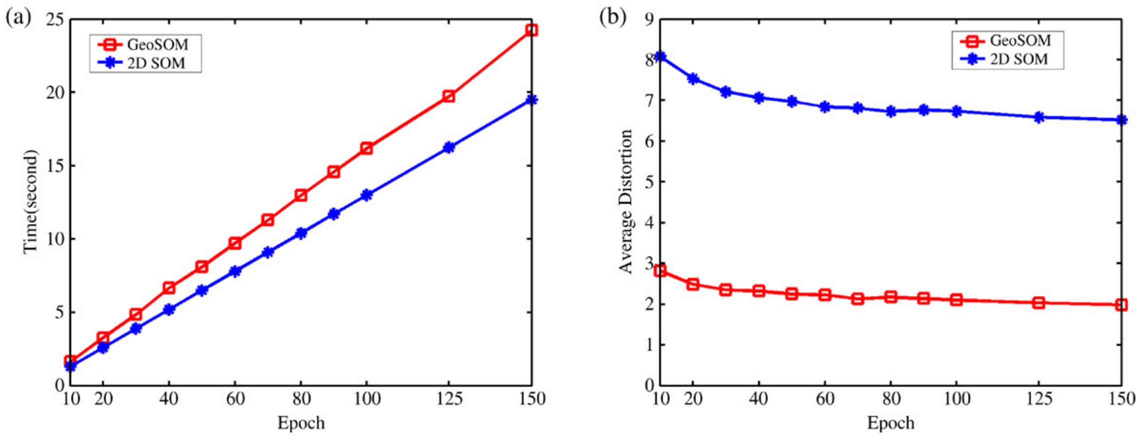


Fig. 12. Experimental results of the breast cancer data set: (a) the running time; (b) the average data distortion at different epochs.

much higher than inside the map. The largest neuron distortion of the 2D SOM is 130.74, while the corresponding value for GeoSOM is 50.65. Compared to the 2D SOM, the GeoSOM reveals more information about how the data is related. For example, cluster 5 is close to clusters 1, 3, 4, 6 and 7 in the original data space. However, on the 2D SOM, cluster 5 is only adjacent to clusters 1, 6, 7 but far away from clusters 3 and 4. These relationships are shown more clearly on the GeoSOM because it has no boundaries.

5.3. *The breast cancer data set*

This data set was downloaded from the machine learning repository of the University of California, Irvine. It contains 699 breast lump samples and each sample is described by nine attributes such as clump thickness, marginal adhesion and single epithelial cell size. The attributes' values are integers that vary from 1 to 10. About 65.5% of the samples are classified as Benign and 34.5% as Malignant. The Benign samples were labeled "2" and the others were labeled "4". 16 samples have one attribute missing. We ignored the missing attributes in finding the best-matching units and also in updating the weight vectors.

In the experiment, the GeoSOM uses an eight-frequency geodesic dome (642 neurons) and the 2D SOM uses a 28 × 23 hexagonal grid (644 neurons). The initial update radius is 11. From Fig. 12, we can see that the experimental results are consistent with the former experiments. GeoSOM runs slightly more slowly than the 2D SOM but it reduces about two thirds of the average distortion. Visualization of the two SOMs trained for 150 epochs are shown in Fig. 13. The sizes of the spheres are more uniform and smaller on the GeoSOM than on the 2D SOM, which indicates that the inputs are better represented by the neurons of the GeoSOM. The border effect can be observed from the 2D SOM, since high distortions mainly occur at the neurons located near the corners and boundaries. The largest distortion value at the neurons of the GeoSOM and the 2D SOM are 29.83 and 69.6, respectively.

6. Comparing GeoSOM to other spherical SOMs

From the above experiments, we can see that the GeoSOM reduces up to two thirds of the data distortion. This reduction is achieved because we use geodesic domes to remove the SOM's border effect. As mentioned earlier, several researchers have also implemented spherical SOMs using icosahedron-based geodesic domes (Boudjemaï et al., 2003; Nakatsuka & Oyabu, 2003; Sangole & Knopf, 2003). It is expected that these SOMs will have the same data distortion as the GeoSOM.

The difference between the GeoSOM and other spherical SOMs using tessellated icosahedrons is the data structure that we used to organize the neurons. The rectilinear grid data structure reduces the overheads required to maintain

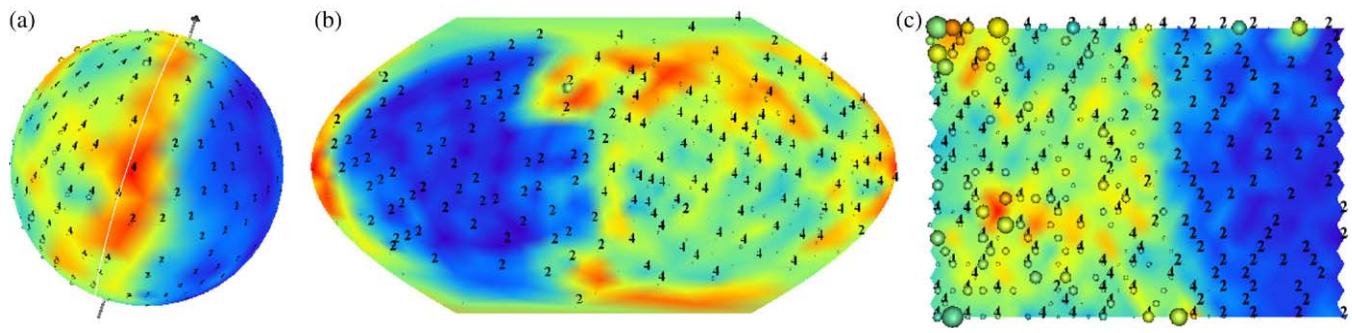


Fig. 13. (a) The GeoSOM trained with the breast cancer data set for 150 epochs. (b) Projection of the GeoSOM onto a 2D plane choosing a point on the cluster boundary as the center of the 2D map. (b) The 2D SOM trained with the breast cancer data set for 150 epochs.

Table 1
Comparison of the GeoSOM, the S-SOM, the 3D-SOM and the N-SOM

	Space complexity (pointers maintenance)	Time complexity (neighborhood searching)	Time complexity (pointer updates in tessellation)
GeoSOM	$O(N^{1/2})$	$O(n)$	$O({N_2}^{1/2})$
S-SOM	$O(N^2)$	$O(1)$	$O({N_2}^2)$
3D-SOM	$O(N)$	$O(n)$	$O(N_2)$
N-SOM	$O(N)$	$O(N)$	$O(N_2)$

In the table, space complexity is the extra overhead caused by using pointers to maintain the neurons' relationships. Time complexity of tessellation is the extra overhead caused by updating the pointers. Notation: N is the number of neurons, n is the number of neighbors within the update radius, and N_2 is the size of the new spherical SOM after tessellation.

the neuron's relationships. In the remainder of this section, we will compare these spherical SOMs by the space and time complexity of these overheads. We will refer to Boudjemaï et al.'s spherical SOM as 3D-SOM, Sangole et al.'s work as S-SOM, and Nakatsuka's work as N-SOM. The analysis results are shown in Table 1.

Space complexity: For the four spherical SOMs with the same number of neurons N, the space required to store the neurons (weight vectors) is the same. Therefore we only compare the extra overhead required to maintain the pointers. The space complexity of storing a single pointer is considered to be $O(1)$:

(1) In the 3D-SOM and the N-SOM, each neuron maintains a list of its immediate neighbor pointers. Most of the vertices on the icosahedron-based geodesic dome have six immediate neighbors. So the space overhead is $6N$, which is $O(N)$ theoretically.
(2) In the S-SOM, every neuron maintains a list of neighbors of each different distance, which requires $O(N^2)$ space.
(3) The GeoSOM organizes its neurons in a 2D rectangular data structure. As mentioned in Section 3, it uses pointers in two places: first, the *u_array* maintains pointers to the *v_arrays*; second, the boundary neurons keep pointers to their duplicated points. The size of the *u_array* is $6 \times f + 1$, where f is the frequency of the geodesic dome. Remembering that $N = 10 \times f^2 + 2$, the space overhead for the *u_array* is $O(N^{1/2})$. The number of boundary neurons is approximately equal to the frequency of the geodesic dome multiplied by the number of boundary edges. In the GeoSOM's data structure, there are 22 boundary edges (see Fig. 3(b)). Most of the boundary neurons have only one duplicated point, so the space overhead for the boundary neurons is $22f$, which is also $O(N^{1/2})$. In total, the GeoSOM requires $O(N^{1/2})$ space to store the pointers.

Time complexity in neighborhood searching: In this discussion, the time spent to visit a single neighbor is considered to be $O(1)$:

(1) During training, the S-SOM pre-computes all neighborhoods, so the time complexity of finding a winning neuron's neighbors is $O(1)$.
(2) Both the 3D-SOM and the GeoSOM search the winning neuron's neighbors level by level. This method is the same as graph traversal using a breadth-first search algorithm. The time complexities for 3D-SOM and GeoSOM are thus both $O(e + n)$, where n is the number of neighbors within the update radius and e is the number of edges on the geodesic dome connecting the neighbors. Because a geodesic dome can be considered as a maximal planar graph, $e = 3n - 6$. Hence the time complexity is actually $O(n)$. Note that the size of the neighborhood shrinks during training, so n decreases as the training proceeds.
(3) In the N-SOM, the neighborhood size is controlled by a neighborhood function. In each update, every neuron needs to be checked to see whether the value of the neighborhood function is bigger than zero. Hence the time complexity is $O(N)$, where N is the total number of neurons.

Time complexity in tessellation: In further tessellating the geodesic dome, all four spherical SOMs need to compute the same number of new neurons. The main difference between GeoSOM and other spherical SOMs lies in updating the pointers to reconstruct the data structure. Therefore we only compare the overhead spent in pointer updates. In the following discussion, the time spent in updating a single pointer is considered to be $O(1)$. The frequency of the original geodesic dome is denoted as f, the frequency of the new geodesic dome is denoted as f_2, and N_2 is the size of the new spherical SOM:

(1) As mentioned in Section 3.4, the GeoSOM needs to expand the *u_array* to accommodate the newly generated *v_arrays* during tessellation. The size of the new *u_array* is $6 \times f_2 + 1$, so the overhead is $O(f_2)$, which is $O({N_2}^{1/2})$ in terms of the new SOM size. The GeoSOM also needs to update the

pointers to the duplicated points, the complexity of which is proportional to the number of boundary neurons. Hence the total time complexity of updating the pointers is $O({N_2}^{1/2})$.

(2) The 3D-SOM, the S-SOM and the N-SOM tessellate the geodesic dome in a recursive manner. The frequency of the geodesic dome is doubled in every iteration until the desired frequency is reached. This procedure can be generally described as:

```
for $i = 0; i < \log_2(f_2/f); i{+}{+}$ do
    for each triangle do
        calculate the middle point of each edge
        update neighbor pointers to construct four smaller
        triangles
    end for
end for
```

Using this method, the number of triangles generated in each iteration is four times the number of triangles in the geodesic dome from the previous iteration. Suppose that the original geodesic dome has T triangles, then the new dome would have $T_2 = 4^{\log_2(f_2/f)}T$ triangles. The number of pointer updates in each iteration is proportional to the number of triangles generated, so the total number of updates is:

$$4cT + 4^2cT + \cdots + 4^{\log_2(f_2/f)}cT = \frac{4}{3}(4^{\log_2(f_2/f)} - 1)cT \tag{4}$$

where c is a constant. Hence the total time complexity is $O(T_2)$. Remembering that the geodesic dome is a maximal planar graph, the number of triangles $T_2 = 2N_2 - 4$. Therefore the complexity in terms of N_2 is $O(N_2)$.

(3) The S-SOM also needs to recompute the neighborhoods for every neuron. Therefore the time complexity for the S_SOM is $O({N_2}^2)$.

From Table 1, we can see that the GeoSOM is the most space efficient among the four spherical SOMs. It also supports fast dome tessellation. Note that, using the recursive tessellation methods of the other spherical SOMs, we can only obtain geodesic domes of frequency 2^n, which is not as flexible as the GeoSOM. These characteristics make the GeoSOM very suitable for visualizing online data sets. At the beginning, we may have a small amount of data, so a small size SOM will be sufficient. As more data accumulates, a larger SOM will be necessary in order to increase the resolution. Using our data structure, we can establish a higher-frequency geodesic dome much more quickly than the other spherical SOMs. Furthermore, weight vectors of the newly generated neurons can be interpolated from the weight vectors of the old neurons. A few epochs of training would be sufficient for the new GeoSOM to converge.

7. Conclusion and future work

It has been demonstrated that a spherical SOM can effectively remove the border effect of the 2D SOM and reveal more information about the high-dimensional data. We introduced a method to organize the neurons of such an SOM using a 2D rectangular grid structure, which reduces the overhead required to maintain the neuron's relationships. Experimental results show that the speed of a GeoSOM is comparable to the speed of a 2D SOM generated by SOM_PAK. Because of the grid topology, GeoSOM reduces the data distortion by two thirds. We also compared the GeoSOM with other proposed spherical SOMs in terms of space complexity and time complexity.

Furthermore, we developed an interface to project the spherical map onto a 2D plane using a cartographic approach. The interface allows users to change the center and the orientation of the 2D projection very easily. We believe that this approach can assist viewers building their mental maps, as they can examine the vicinity of any point while keeping an overview of all the data. It is similar to the focus + context technique (Card, Mackinlay, & Shneiderman, 1999) in information visualization. We intend to carry out user studies to evaluate this approach in the future.

References

Boudjemaï, F., Enberg, P. B., & Postaire, J. -G. (2003). Self organizing spherical map architecture for 3D object modeling. In *Proceedings of workshop on self-organizing maps*.

Canters, F., & Decleir, H. (1989). *The world in perpective: A directory of world map projections*. England: John Wiley & Sons Ltd.

Card, S. K., Mackinlay, J., & Shneiderman, B. (1999). *Readings in information visualization: Using vision to think*. Morgan Kaufmann Publishers.

Fuller, R. B. (1975). *Explorations in the geometry of thinking*. Macmillan Publishing Co. Inc.

Hirokazu, N., Altaf-Ul, A. M., Ken, K., Kotaro, M., & Shigehiko, K. (2005). Spherical SOM with arbitrary number of neurons and measure of suitability. In *Proceedings of the 5th workshop on self-organizing maps*. University Paris 1, Panthon-Sorbonne.

Hoffmann, G. (2002). Sphere tessellation by icosahedron subdivision.

Ito, M., Miyoshi, T., & Masuyama, H. (2000). The characteristics of the torus self organizing map. In *Proceedings 16th fuzzy systerm symposium Akita* (pp. 373–374). Japan Society for Fuzzy and Systems.

Kohonen, T. (1991). Self-organizing maps: Optimization approaches. In T. Kohonen, K. Makisara, O. Simula, & J. Kangas (Eds.), *Proceedings of ICANN'91 International conference on artifial neural networks*: *vol. 2* (pp. 981–990). Amsterdam: North Holland.

Kohonen, T. (2001). *Springer series in information sciences*, *Self-organizing maps* (3rd ed.). Germany: Springer-Verlag.

Kohonen, T., Hynninen, J., Kangas, J., & Laaksonen, J. (1996). SOM_PAK: The self-organizing map program package. Tech. Rep. A31. Laboratory of Computer and Information Science, Helsinki University of Technology.

Lagus, K., Kaski, S., & Kohonen, T. (2004). Mining massive document collections by the WEBSOM method. *Information Sciences*, *163*, 135–156.

Li, X., Gasteiger, J., & Aupan, J. (1993). On the topology distortion in self-organizing feature maps. *Biological Cybernetics*, *70*, 189–198.

Lin, X., White, H. D., & Buzydlowski, J. (2003). Real-time author co-citation mapping for online searching. *Information Processing and Management: an International Journal*, *39*, 689–706.

Nakatsuka, D., & Oyabu, M. (2003). Application of spherical SOM in clustering. In *Proceedings of workshop on self-organizing maps*.

Nakatsuka, D., & Tokutaka, H. (2004). Visualization and cluster analysis using spherical SOM. In *Proceeding of the genome informatics*.

Pugh, A. (1976). *Polyhedra—a visual approach*. University of California Press, Ltd.

Ritter, H. (1999). *Self-organizing maps in non-euclidean spaces* (pp. 97–108). Amsterdam: Elsevier.

Ritter, H., Martinetz, T., & Schulten, K. (1992). *Neural computation and self-organizing maps; An introduction*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Sangole, A., & Knopf, G. K. (2003). Visualization of randomly ordered numeric data sets using spherical self-organizing feature maps. *Computers & Graphics*, *27*(6), 963–976.

Sarle, W.S. (2002). SOM FAQ.

Vesanto, J., Sulkava, M., & Hollmn, J. (2003). On the decomposition of the self-organizing map distortion measure. In *Proceedings of the workshop on self-organizing maps* (pp. 11–16).

Wand, M., & Jones, M. (1995). *Kernel smoothing*. London: Chapman & Hall.