Fast Spherical Self Organizing Map
- Use of Indexed Geodesic Data Structure -

Yingxin Wu [(1)], **Masahiro Takatsuka** [(2)]
[(1)] School of Information Technologies, The University of Sydney
Sydney. Australia
[(2)] ViSLAB, The University of Sydney
Sydney. Australia
chwu@it.usyd.edu.au, masa@vislab.net

Abstract - *In order to remove the "border effect", several spherical Self-Organizing Maps (SOM) based on the geodesic dome have been proposed. However, existing neighborhood searching methods on the geodesic dome are much more time-consuming than searching on the normal rectangular/hexagonal grid. In this paper, we present detailed descriptions of the algorithms used in training the Geodesic SOM (GeoSOM), which we previously proposed. Experimental results show that the GeoSOM runs almost at the same speed of the conventional 2D SOM and it also represents the data more accurately. In visualizing the GeoSOM, a 2D data map can be created by projecting the spherical SOM onto 2D the plane. The user is able to select any point of interest to be the center of the 2D map. No retraining is required when changing the center point.*

Key words - data structure, geodesic dome, Self-Organizing Map, visualization

1 Introduction

1.1 Border effect and the spherical SOM

In a conventional SOM, the neighborhood relationship is defined by a 2D 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, which often leads to the notorious "border effect" — the weight vectors of these units "collapse to the center of the input space"[11].

Several approaches have been suggested to eliminate the "border effect", such as the heuristic weighting rule method by Kohonen [7] and the local-linear smoothing by Wand and Jones [12]. Besides the mathematical solutions, the problem can also be solved by removing the edges from the SOM: implement the SOM on a torus [6, 14] or on a sphere. There are several disadvantages with the torus-based SOM. First, when it is used for the purpose

of visualization, it fails to provide an intuitively readable map. We are familiar with a map generated from a sphere but not those based on the torus. Second, the surface areas associated with each neuron varies significantly on a surface of the torus. The surface area allocated to a neuron is much smaller near the inner circle and much larger around the outer circle. For SOM, it's desirable to have all neurons receive equal geometrical treatment. Ritter [9] suggested the use of tessellated platonic polyhedra as the underlying lattices. He also pointed out that the spherical SOM would be very suitable for data with underlying directional structures. Since then, several spherical SOM based on the geodesic dome have been implemented and applied to different types of data sets [3, 4, 10].

1.2 A brief discussion of the geodesic dome

The five platonic polyhedra (tetrahedron, octahedron, cube, dodecahedron and icosahedron) are all regular tessellations of the sphere. Each of them has equal edge length and their vertices have the same number of immediate neighbors. However, the geodesic domes created by further tessellating the polyhedra will not have such regular characteristics. Figure 1 illustrate the tessellation process. Detailed descriptions of the tessellation process can be found in [8]. Because the icosahedron is most similar to the sphere, so after tessellation, it always has the smallest variances in edge length compared to the geodesic domes obtain from other platonic polyhedra. This makes the icosahedron-based geodesic dome more suitable for a spherical SOM [13].

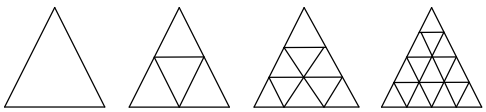


Figure 1: Left to right: one-frequency, two frequency, three frequency and four frequency tessellation

1.3 Existing data structures for the geodesic dome

During the training of a SOM, a winning neuron and its neighbors, defined by a neighborhood function, update their weight vectors for each input. This process will go for a number of iterations in order for the SOM to converge. Neighbor neurons are located as a result of a neighborhood search. Hence, it could be a very time-consuming process, especially when a data set is large. A conventional SOM stores its neurons in a 2D rectangular or hexagonal lattice, more specifically a 2D array data structure. This means that the neighbors of the winning neuron can be located very fast by simple 2D array indexing. Storing a geodesic lattice structure is usually much more complex. Hence, the data structure to store the geodesic lattice should support fast vertex indexing in order to avoid considerable performance loss.

In Sangole's spherical SOFM [10], 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. Farid's SOM [4] used a similar scheme by treating the geodesic dome as a

3D adjacency graph. The main problems of using this approach is that we need to carefully maintain a large number of pointers, and the execution speed is much slower compared to looking for neighbors in a 2D array. Furthermore, if we need to increase the number of units in a SOM [1], all the immediate neighbor pointers need to be recursively updated. Nakatsuka [3] calculates the distance between two neurons as the angle between their location vectors. This method is also slow because every time a neighbor is updated, we need to perform extra floating point calculations to obtain the distance.

2 Data structure for the icosahedron-based geodesic dome

2.1 The 2D data struture

The icosahedron-based geodesic dome can be opened up on a 2D plane. Figure 2(a) shows a four frequency icosahedron after opening. The lines marked with the same color indicate where the dome are split and they represent the same line on the dome. Vertices along these lines need to be duplicated. Most of the boundary vertices would be duplicated only once, such as point B and B'. However, the two poles A and C would be duplicated for four times.

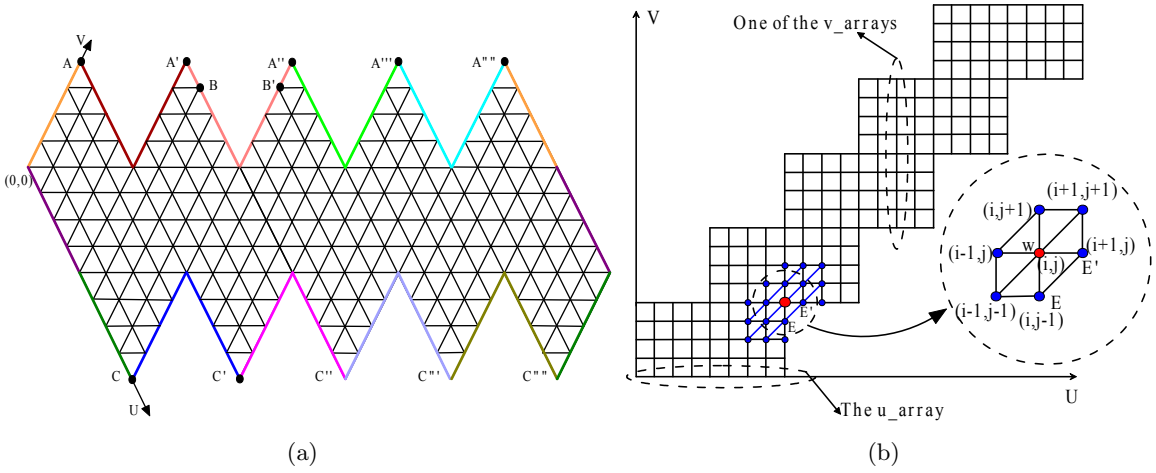


Figure 2: (a) Unwrapped four frequency Icosahedron and (b) the corresponding data Structure

We defined two axes U and V on this lattice. Each vertex on the lattice 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. In our program, vertices sharing the same u coordinate are stored in a one-dimensional array, called a *v_array*, in order of ascending v coordinate. Every *v_array* is contained within another one-dimensional array, called the *u_array*, in order of ascending u coordinate. It should be noted that only the vertices at the boundary may be duplicated; therefore the number of duplicated vertices will be very small compared to the total number of vertices. Using this data structure, only $O(n)$ space is needed to store the geodesic dome, where n is the number of vertices on the dome.

2.2 Searching and updating the neighborhood

Similar to Sangole's method [10], the neighbors of a vertex are grouped into levels. For the point W in Figure 2(b), its first level neighbors are the vertices that connect to it by only one edge on the geodesic dome. The second level neighbors are the vertices that connect to W by two edges and so on. In order to find all neighbors within the 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 $W(i, j)$, the first level neighbors can be indexed as easy as: $(i, j+1)$, $(i+1, j+1)$, $(i+1, j)$, $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$. As mentioned before, there are duplicated points on the lattice's boundaries and they represent the same weight vector. For example, in Figure 2(b), point 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 our program, each vertex maintains a list of its duplicated points. If one of them is already included, others are marked and wouldn't be included again. In the case when the winning neuron itself has duplicated points, such as point A (in Figure 2(a)), all the neighbors of its duplicated points (from A' to A'''') need to be searched. Following is the pseudo code for searching and updating a winning neuron's neighborhood on the data structure.

Algorithm 1 Find and update neighbors within radius r of the winning neuron

```
1: Find the first level neighbors of the winning neuron
2: Insert the neighbors into list A
3: Initialize an empty list A_1
4: for l = 2 to r do
5:    for each neuron in A do
6:       if visited then
7:          Continue;
8:       end if
9:       Update the neuron's weight vector according to l
10:      Marked the neuron and its duplicated points as visited
11:      Find the first level neighbors of the neuron
12:      Insert the neighbors into list A_1
13:   end for
14:   A ⇐ A_1
15:   A_1 ⇐ ∅
16: end for
```

2.3 Increasing the number of neurons

The number of vertices of the icosahedron-based geodesic dome is given by: $v = 10 * f^2 + 2$, where f is the frequency number [5]. In our data structure, every pair of adjacent vertices is connected by one edge on the geodesic dome. Hence, increasing a geodesic dome's frequency involves inserting new vertices into every adjacent pair of vertices. In fact, it is quite straightforward to obtain an $m * n$ frequency geodesic dome from an m frequency dome. We only

need to calculate and insert $(n-1)$ new vertices between the adjacent vertices pairs. Here for simplicity, Figure 3 only illustrates how to tessellate the icosahedron into a two frequency geodesic dome using our data structure. Some new duplicated points will be created along the boundaries of the map. Therefore, after inserting the new vertices, we need to scan along the map boundaries to find all the duplicated points. Algorithm 2 is the pseudo code for increasing a geodesic dome's frequency.

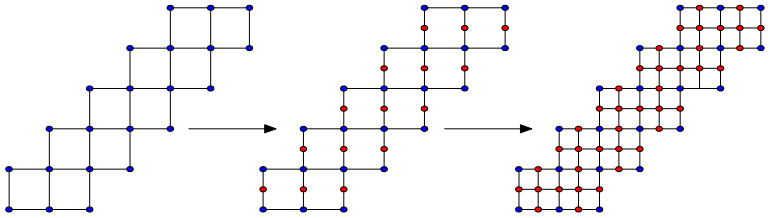


Figure 3: Increasing the frequency of the geodesic dome

Algorithm 2 Increase frequency of the geodesic dome by f

1: **for** every v_array **do**
2: calculate the $f-1$ new vertices between every two vertical adjacent vertices
3: **end for**
4: **for** every two adjacent v_arrays **do**
5: calculate the $f-1$ new vertices between every two horizontal adjacent vertices
6: calculate the $f-1$ new vertices between every two diagonal adjacent vertices
7: **end for**
8: Scan the boundary of the new map to find the duplicated points

Note that in increasing the frequency, we only need to calculate the new vertices and insert them into the right place of the data structure. No additional adjacency matrix or neighbor pointers need to be updated.

3 Experimental Evaluation

In this section, we compare our GeoSOM with the conventional 2D SOM. The 2D SOM is trained by SOM_PAK [7]. Training times and quantization errors are measured. We also implement a program to visualize the training results.

3.1 Experiment setup

We generate 3D synthetic data for this experiment. The data set contains seven clusters and each cluster has 500 normally distributed data points. The standard deviation is 1 in each dimension. The means of the clusters are: $(0,0,0)$, $(10,0,0)$, $(0,10,0)$, $(0,0,10)$, $(-10,0,0)$, $(0,-10,0)$, $(0,0,-10)$. Figure 4 shows the how the clusters are located in 3D space. The GeoSOM uses a 10 frequency geodesic dome as the lattice which contains 1002 neurons. The 2D SOM is trained on a 37×27 hexagonal grid (999 neurons). For both SOMs, the initial learning rate is 0.8 and the initial update radius is 20, both of which linearly

decrease according to the training time. A Gaussian neighborhood is used. The experiment is conducted on a Pentium IV 3.0GHz workstation with 1 GB memory. The operating system is Windows XP Professional Edition. Note that the GeoSOM was implemented in Java and SOM_PAK was written in C.

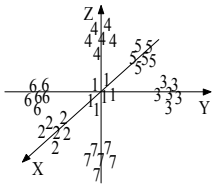


Figure 4: The 3D synthetic data set

3.2 Results

For this relatively simple data set, both of the SOMs converge very fast: the GeoSOM begins to converge after 10 epochs of training while the 2D SOM begins to converge after 15 epochs. The GeoSOM always achieves a lower quantization error than the 2D SOM. For example, when trained for 40 epochs, the error of the GeoSOM is 0.4996441 and the error of the 2D SOM is 0.6632137, which means the GeoSOM reduces about 25% of the quantization error. As to the training speed, the GeoSOM is a little slower than the 2D SOM due to more complex neighborhood searching. However, considering the GeoSOM was implemented in Java, we can conclude our proposed data structure indeed supports fast neighborhood searching on the geodesic dome.

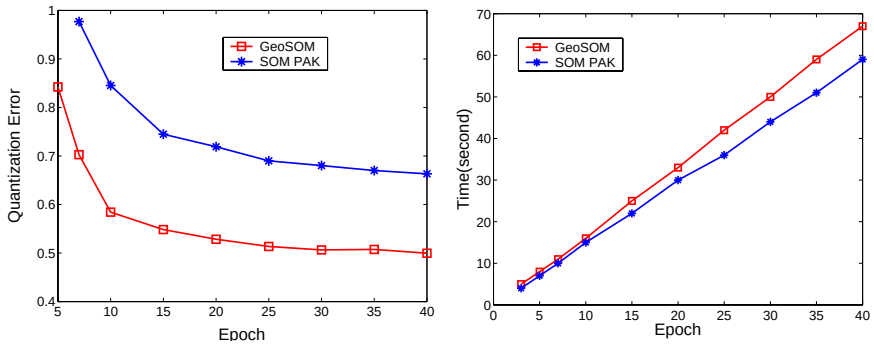
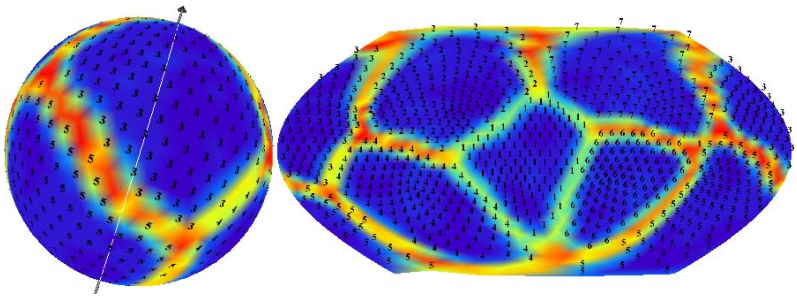


Figure 5: The quantization error and training time

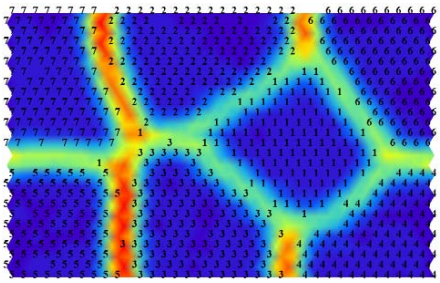
3.3 Visualize the experiment results

In our visualization program, the neurons are colored according to the average distances between each neuron and its direct neighbors. The colors are linearly changing from blue, cyan, yellow to orange. Blue denotes the smallest variance between the weight vectors, and orange the largest.

Compared to a 2D data map, the spherical map is not convenient for the user to have a global view of the entire data set. Hence, we implement an interface to project the GeoSOM on to a 2D plane. Currently, the Wagner III pseudocylindrical projection is used. This projection is neither equal-area or conformal. However, it has less extreme distortions and therefore can give a more balanced representation of the sphere's main features [2]. Using our program, the user can choose any point on the sphere to be the center of the 2D map. He/she can also choose the direction of the sphere's central axis. Using the conventional 2D SOM, this can only be done by fixing some data on the SOM during training. If the user wants to change the point of interest, he/she needs to redo the training. Figure 6 shows the visualization results. The white line on the GeoSOM shows where the sphere is split and the arrow indicates the direction of the central axis.



(a) GeoSOM



(b) 2D SOM

Figure 6: Visualization of SOMs after trained for 100 epochs.

From the visualization of both SOMs, we can see that there are seven clusters in the data set and adjacent clusters on the map are close to each other in the original data space. Compared to the 2D SOM, the GeoSOM can give more information about how the data is related. For example, cluster 5 is close to clusters 1, 3, 4, 6 and 7. However, on the 2D SOM, 5 is only adjacent to 3 and 7 but far away from clusters 1, 4, 6. These relationships are shown more clearly on the GeoSOM because the it has no boundaries. Of course, some information is lost because of the dimension reduction: cluster 1 is closed to every other cluster in the original data space; but on both of the SOMs, two of the clusters are not adjacent to cluster 1.

4 Conclusions

Several spherical SOMs using a geodesic dome as an underlying grid structure have been implemented and applied to different data sets. However, existing neighborhood searching algo-

rithms on the geodesic dome are inefficient, which makes the time-consuming training process even longer. We proposed a 2D rectangular grid data structure to store the icosahedron-based geodesic dome. This data structure supports efficient tessellation of icosahedron as well as fast neighborhood searching. Experiment results show that our GeoSOM not only runs in comparable speed with SOM_PAK, but also cut down the quantization error by 25%. We also developed an interface to project the spherical SOM on to a 2D plane. The user can choose any point of interest to be the center of the 2D data map for close examination.

References

[1] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan. Dynamic self-organizaing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000.

[2] F. Canters and H. Decleir. *The World In Perpective: A Directory of World Map Projections*. John Wiley & Sons Ltd., England, 1989.

[3] N. Daisuke and O. Matashige. Application of spherical som in clustering. In *Proceedings of Workshop on Self-Organizing Maps, WSOM03*, Kitakyushu, Japan, September 2003.

[4] B. Farid, E. P. Biela, and P. Jack-Gérard. Self organizing spherical map architecture for 3d object modeling. In *Proceedings of Workshop on Self-Organizing Maps, WSOM03*, Kitakyushu, Japan, September 2003.

[5] G. Hoffmann. Sphere tessellation by icosahedron subdivision. 2002.

[6] M. Ito, T. Miyoshi, and H. Masuyama. The characteristics of the torus self organizing map. In *Proceedings 16th Fuzzy Systerm Symposium Akita*, pages 373–374. Japan Society for Fuzzy and Systems, 2000.

[7] T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer-Verlag, Germany, 3rd edition, 2001.

[8] A. Pugh. *Polyhedra—a Visual Approach*. University of California Press, Ltd, 1976.

[9] H. Ritter. *Non-Euclidean Self-Organizing Maps*, pages 97–109. ELSEVIER, Amsterdam, 1999.

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

[11] W. S. Sarle. SOM FAQ. 2002.

[12] M. Wand and M. Jones. *Kernel Smoothing*. Chapman & Hall, London, 1995.

[13] Y. Wu and M. Takatsuka. Geodesic self-organizing map. In *Proceedings of Conference on Visualization and Data Analysis 2005*. IS&T / SPIE, January 2005.

[14] L. J. Xinzhi. Visualization of high-dimensional data with relational perspective map. *Information Visualization*, 3:49–59, 2004.