Using the SOM Visualization Software

Obtaining the Software

The SOM visualization software can be downloaded from the Internet at http://jsomap.sourceforge.net/ee547/somviz.zip. Once somviz.zip is downloaded, simply unzip it to some directory. In that directory, a new directory called somviz is created that contains all of the relevant files.

The SOM visualization software is based on three packages:

- JSOMap – self-organizing maps – http://jsomap.sourceforge.net
- Orca – data visualization– http://software.biostat.washington.edu/orca
- Jama – matrix operations used by Orca – http://math.nist.gov/javanumerics/jama/

The somviz.zip file contains a .jar file for each package. Additionally, it contains a data directory where data files are stored.

Since the SOM visualization software is 100% Java, you must have a Java virtual machine installed on the computer you are using. The software should work with any jvm version 1.2 or higher (only version 1.3 has been fully tested but 1.2 should work). See http://java.sun.com for details on obtaining a Java virtual machine if one is not already installed on your computer.

Data files

The data directory contains these five data files:

- 3dclusters.txt – three normally distributed clusters in three dimensions
- flea2.txt – three clusters in six dimensions
- uniform2.txt – uniformly distributed in three dimensions
- curve.txt – slight curve in three dimensions
- cross4d.txt – four dimensional cross

You can also use your own data files, as long as they are plain text, with data in matrix form; that is, rows are data points and columns are variables. There also must be a string label for each column on the first line of the file.

Starting the Program

To start the SOM visualization software, simply execute the following on a command line while in the somviz directory

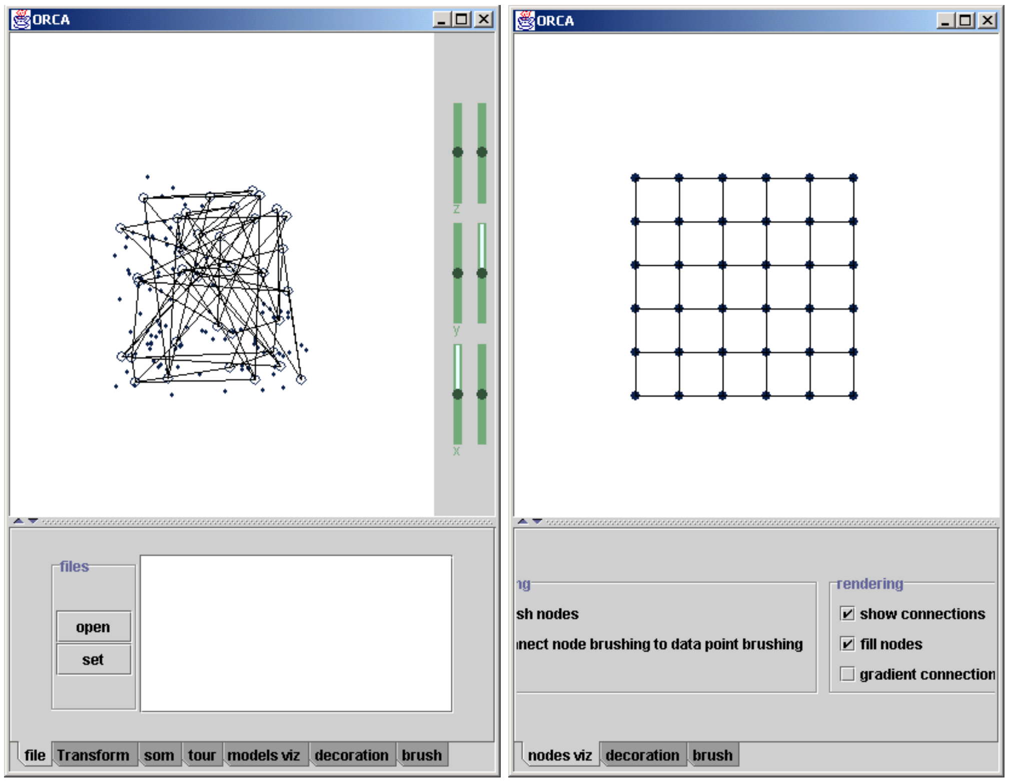
```
java –cp .;jsomap.jar;orca.jar;jama.jar examples.SOMExample data/DATAFILE.txt
```

where DATAFILE.txt is one of the data files in the data directory.

For those interested in what this command actually does, it basically puts the current directory and the three .jar files on the classpath, instructs the Java virtual machine to call the main method in the examples.SOMExample class, and uses the specified data file.

Running the SOM Algorithms

When the program starts, you should see the following two windows:

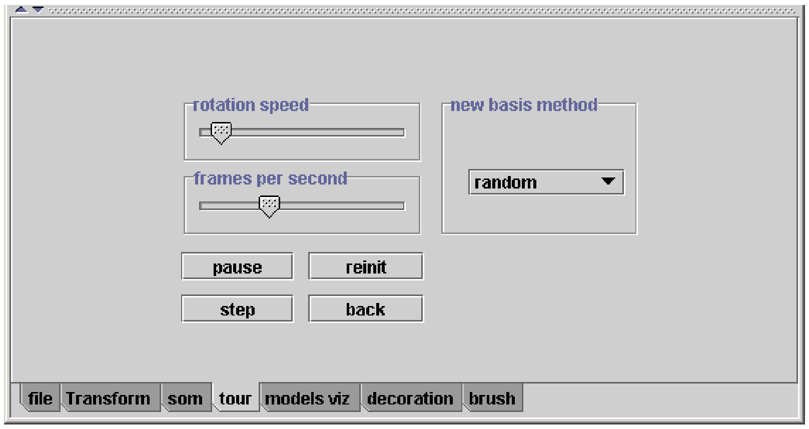


☑ show connections
☑ fill nodes
☐ gradient connection

Figure 1. The two windows shown when the program starts. The grand tour view is on the left and the map view is on the right.

We'll focus on the tour view (the window on the left of Figure 1) first. At this point, it might be a good idea to drag the bottom-right of the window out a bit, to make it bigger (some of the tabs are too big and won't fit in the window at this size).

Next, select the tour tab to access the grand tour controls, shown in Figure 2. The main thing we'll use on this tab is the rotation speed slider, which controls the speed of the tour. The tour increases speed as the slider moves to the right. Initially the tour is paused, so drag the slider to about the middle of the range to get it going at a decent speed. Also, if the animation looks choppy, try moving the frames per second slider to the right to increase the number of frames shown per second.

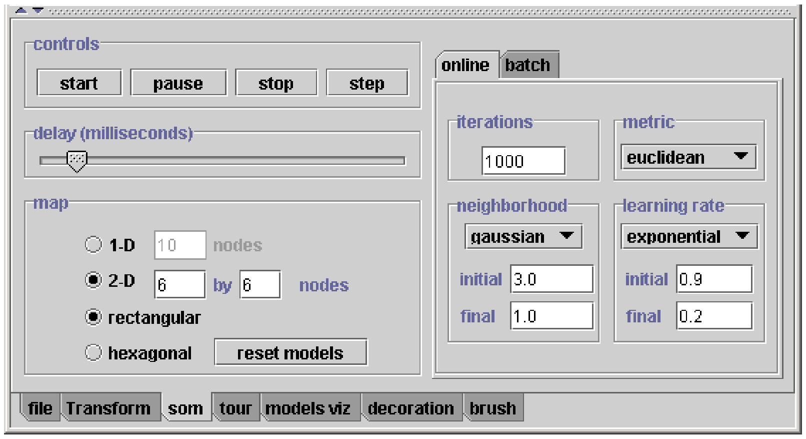


new basis method: random

Figure 2. The tour tab. This tab shows the controls for the grand tour.

Now that the tour is running, you can see both the data points (the small dots) and the map (the larger open circles are the models and the lines between them show the map topology). The models of the map are randomly initialized in the input space, so the map really looks like a mess.

The next step, and probably the most exciting, is to run the SOM algorithm. Click on the som tab to show the SOM algorithm controls. You should see a user interface like the one in Figure 3. It may look a little confusing at first, but don't worry, we'll cover everything in a minute. Click the start button (at the top left) to start the algorithm, and watch the map organize itself!



map: 1-D 10 nodes; [✓ 2-D] 6 by 6 nodes; [✓ rectangular]; hexagonal

iterations: 1000

metric: euclidean

neighborhood: gaussian; initial 3.0; final 1.0

learning rate: exponential; initial 0.9; final 0.2

Figure 3. The som tab. This tab shows the controls for the SOM algorithms.

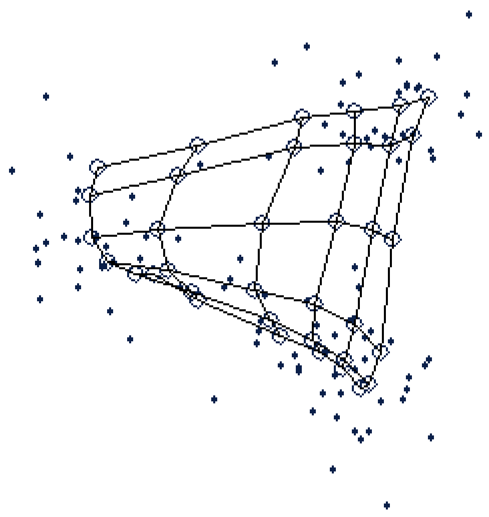


Figure 4. An example of a trained map.

After pressing the start button, you should see the map starting to bend and warp around the data, becoming more like an organized surface. To speed up or slow down the algorithm, move the delay slider to the right or the left, respectively. Eventually the algorithm will finish and the map will be fully trained on the data. Figure 4 shows what the map should look like after the SOM algorithm is done training it. If you're not interested in what all of the other controls on the som tab do, feel free to skip to the next section to learn about brushing.

The control panel

We've already seen what the start button does; the other buttons in the control panel do useful things too. While the algorithm is running, you can pause it by pressing the pause button and stop it by pressing the stop button (pretty straightforward, huh?). When the algorithm is paused, pressing the step button performs the next iteration of the algorithm. Thus, you can step through iterations one at a time.

The delay slider

As mentioned previously, the delay slider controls the iteration speed of the algorithm. More accurately, the position of the slider determines the amount of delay between iterations of the algorithm. When the slider is all the way to the left there is no delay between iterations and when it's all the way to the right there is a 200-millisecond delay.

The map panel

The map panel controls the topology of the map. You can choose either a 1-D map (a line of nodes) or a 2-D map (a grid of nodes), and specify how many nodes should be in the map. In 2-D mode, you can also select either a rectangular or hexagonal map topology. Pressing the reset models button (surprise!) randomly distributes the models through the input space.

The online tab

There are currently two algorithms available – the online version and the batch version. When the online tab is selected, the online algorithm is used. Recall that the online algorithm performs the following steps iteratively:

1. Select the next pattern $\mathbf{x}^{(k)}$
2. Find the winning node $\mathbf{r}_c$ with the most similar model to $\mathbf{x}^{(k)}$
   $$\mathbf{r}_c = \arg\min_i \left[\mathrm{d}\left(\mathbf{x}^{(k)}, \mathbf{m}_i^{(k)}\right)\right]$$
3. Move the model of the winning node, and its neighbors, towards $\mathbf{x}^{(k)}$
   $$\mathbf{m}_i^{(k+1)} = \mathbf{m}_i^{(k)} + \mathrm{n}(\mathbf{r}_c, \mathbf{r}_i, k) * \left[\mathbf{x}^{(k)} - \mathbf{m}_i^{(k)}\right]$$

The neighborhood function, $\mathrm{n}(\mathbf{r}_c, \mathbf{r}_i, k)$, controls how far each model $\mathbf{m}_i$ moves, based on the winning node $\mathbf{r}_c$, the model's node $\mathbf{r}_i$, and the current iteration *k*. The neighborhood function can also be written as

$$\mathrm{n}(\mathbf{r}_c, \mathbf{r}_i, k) = \alpha(k) * \mathrm{K}(\mathbf{r}_c, \mathbf{r}_i, k)$$

where $\alpha(k)$ is some monotonically decreasing function of *k* called the learning rate and $\mathrm{K}(\mathbf{r}_c, \mathbf{r}_i, k)$ is some kernel function whose width decreases with *k*.

The online tab allows the user to configure several parts of the online algorithm, including

- Number of iterations
- Type of distance metric
- Type of neighborhood function
- Initial and final values of the neighborhood function
- Type of learning rate function
- Initial and final values of the learning rate function

The batch tab

The batch algorithm is a bit different than the online algorithm. Instead of considering a single pattern at a time, all of the patterns are considered during each iteration. The batch algorithm performs the following steps iteratively:

1. Assign each pattern **x** to the node $\mathbf{r}_j$ with the most similar model $\mathbf{m}_j$. Each node $\mathbf{r}_j$ has a set R_j of patterns that were assigned to it.
2. Update each model based on the patterns assigned to its node and to its neighbors' nodes.
   $$\mathbf{m}_i^{(k+1)} = \frac{\sum_j \left(\sum_{p \in R_j} \mathbf{x}_p\right) * \mathrm{K}(\mathbf{r}_i, \mathbf{r}_j, k)}{\sum_j \mathrm{K}(\mathbf{r}_i, \mathbf{r}_j, k) * |R_j|}$$

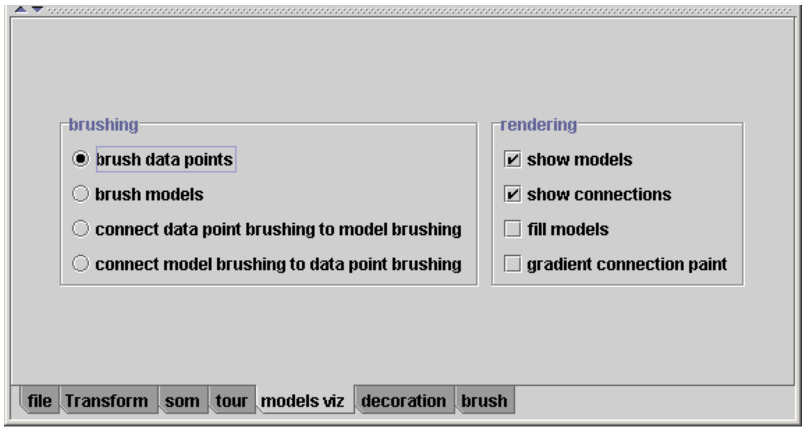
Again, $K(\mathbf{r}_i, \mathbf{r}_j, k)$ is the neighborhood function, and computes how much the patterns assigned to node $\mathbf{r}_j$ affect the new model $m_i^{(k+1)}$ of node $\mathbf{r}_i$. Note that there is no learning rate used in the batch algorithm.

The batch tab allows the user to configure several parts of the batch algorithm, including

- Number of iterations
- Type of distance metric
- Type of neighborhood function
- Initial and final values of the neighborhood function

Interacting with the Grand Tour View

Running the SOM algorithm and visualizing the map in the same space as the data is very interesting, but there is more that we can do. Select the models viz tab to display the visualization controls for the grand tour view. This tab is shown in Figure 5 below.



brushing
- [✓ brush data points]
- brush models
- connect data point brushing to model brushing
- connect model brushing to data point brushing

rendering
- ☑ show models
- ☑ show connections
- ☐ fill models
- ☐ gradient connection paint

Figure 5. The "models viz" tab. This tab shows the visualization controls for the grand tour view.

There are two sets of controls on this tab: the brushing controls and the rendering controls. The rendering controls are simple, so we'll discuss them first. There are four options for rendering, and you can turn the options on or off by selecting or deselecting the check boxes. You should try checking and unchecking the boxes to see what they do (they do exactly what they say).

The last rendering option controls the color of the connection lines between the models. If it's unchecked, the lines are simply colored black. If it's checked, the ends of the connection line are colored to match the color of the corresponding model, and the color follows a gradient between the two ends. Note that selecting this option generally makes the animation choppy, as calculating the color gradient is quite computationally expensive.

Linked brushing is an important data visualization technique. If multiple plots of the same data are linked, changing the appearance of the data in one plot also changes the appearance of the data in the other plots. The grand tour view and the map view are linked, and there are four different types of brushing we can do. To perform brushing, just select the brush tab, click on a color, and then drag a rectangle around the data you want to brush. It's that easy!

Brushing data points

Since the map view does not show the data points, selecting this option and brushing the data points in the tour view only colors the points in the tour view. If there were another type of plot linked to the tour view, like a scatter plot matrix, this would be a useful form of brushing.

Brushing models

Each node in the map view has a corresponding model in the tour view. If this option is selected, brushing on the models in the tour view also brushes their corresponding nodes in the map view. This type of brushing lets you find out exactly where a certain model is in the map. Figure 6 shows an example of brushing models.

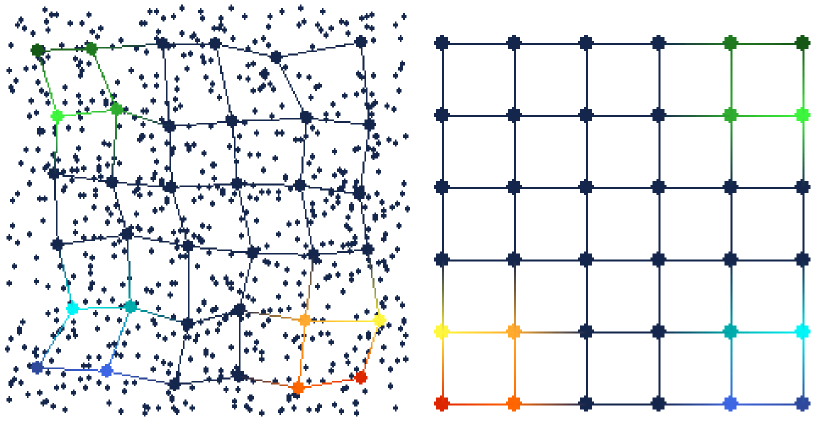


Figure 6. Example of brushing models in the tour view.

Connecting data point brushing to model brushing

One use for a SOM is projection. Basically, once the SOM is trained, a pattern can be projected onto a node of the map by finding the most similar model and assigning the pattern to that model's corresponding node. You can interactively perform this projection by brushing a data point in the tour view. The model and node that the data point is projected to are then also brushed to the same color. Figure 7 shows an example of connecting data point brushing to model brushing.

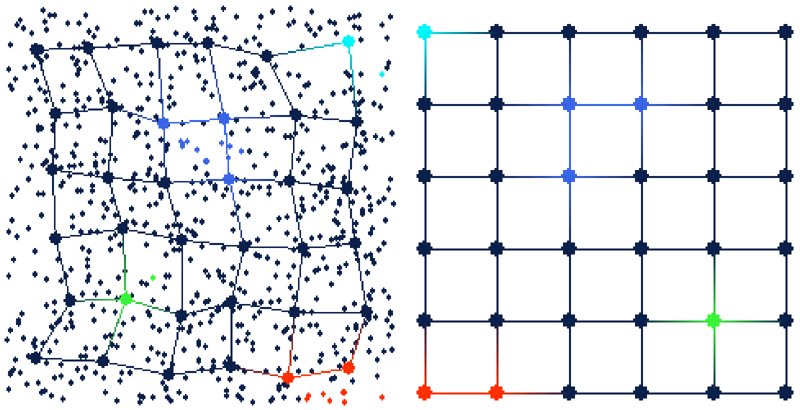


Figure 7. Example of connecting data point brushing to model brushing.

Connecting model brushing to data point brushing

You can also do the reverse of the previous type of brushing. Selecting this option, you can brush a model in the tour view, and all of the data points that project to that model will also be brushed. This type of brushing is useful for seeing how the map partitions the data. Figure 8 shows an example of connecting model brushing to data point brushing.

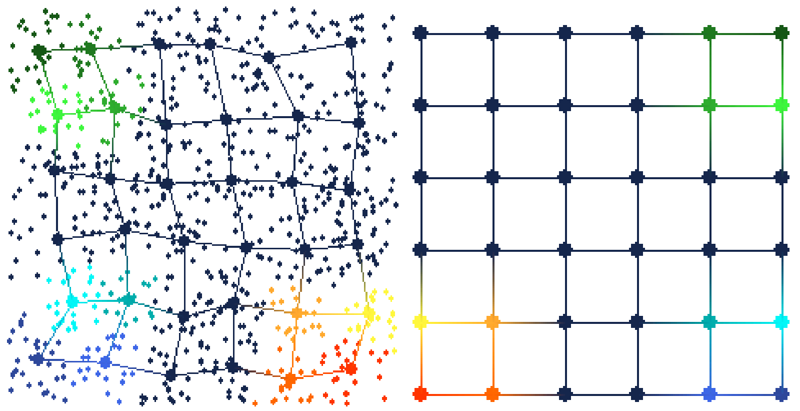


Figure 8. Example of connecting model brushing to data point brushing.

Interacting with the Map View

Now let's turn out attention to the map view. The map view just shows the map in feature space (either 1-D or 2-D). This probably isn't that interesting until we can start interacting with the map view. Select the nodes viz tab, and we'll explain how to do this interaction.

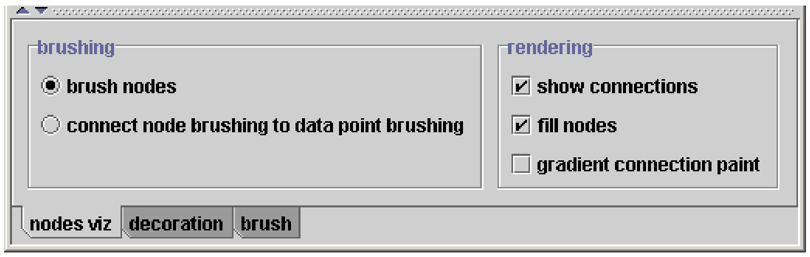

brushing
[✓ brush nodes]
connect node brushing to data point brushing
rendering
[✓ show connections]
[✓ fill nodes]
gradient connection paint

Figure 9. The "nodes viz" tab. This tab shows the visualization controls for the map view.

The nodes viz tab, shown in Figure 9, is very similar to the models viz tab that we looked at earlier. It has basically the same rendering controls, but only two brushing options. There are no data points in the map view, so the two data point brushing options that were available for the tour view are not applicable to the map view. Let's discuss the two options that are available.

Brushing nodes

This type of brushing is similar to brushing models in the tour view. Brushing a node in the map view also brushes its model in the tour view.

Connecting node brushing to data point brushing

Using this brushing option, you can project data points to the nodes in map, much like the second type of model brushing in the tour view. Brushing a node in the map view brushes all of the data points in the tour view that project to that node.