Self-Organizing Graphs – A Neural Network Perspective of Graph Layout

Bernd Meyer

University of Munich
Oettingenstr. 67, D-80538 Munich, Germany
bernd.meyer@acm.org

Abstract. The paper presents self-organizing graphs, a novel approach to graph layout based on a competitive learning algorithm. This method is an extension of self-organization strategies known from unsupervised neural networks, namely from Kohonen's self-organizing map. Its main advantage is that it is very flexibly adaptable to arbitrary types of visualization spaces, for it is explicitly parameterized by a metric model of the layout space. Yet the method consumes comparatively little computational resources and does not need any heavy-duty preprocessing. Unlike with other stochastic layout algorithms, not even the costly repeated evaluation of an objective function is required. To our knowledge this is the first connectionist approach to graph layout. The paper presents applications to 2D-layout as well as to 3D-layout and to layout in arbitrary metric spaces, such as networks on spherical surfaces.

1 Introduction

Despite the fact that large efforts have been devoted to the construction of graph layout tools, their usability in practical applications is still relatively limited for large graphs and in the case of non-standard layout requirements. Two important issues that have to be addressed are flexibility and speed. The dilemma is that fast algorithms, such as Sugiyama layout [24], are usually highly specialized and tailored for a particular domain. On the other hand, more flexible declarative layout methods, in particular simulated annealing [5] and other general stochastic optimization methods, are computationally very expensive [6]. The situation becomes worse if the evaluation of the cost function is expensive, such as checking the (potentially quadratic) number of edge crossings, because it has to be evaluated on every iteration. Especially genetic algorithms can exhibit a very problematic performance [21]. In real-world tasks, such as re-engineering, graph sizes can easily reach more than 100000 nodes. In such cases speed is of prime importance even if a sub-optimal layout has to be accepted.

This paper introduces a flexible new layout method called ISOM layout that in comparison with other stochastic techniques consumes only little computational resources. No heavy-duty preprocessing and no costly repeated evaluation of an objective function are required. One of the method's major advantages is its extreme versatility in regard to the visualization space used. The algorithm

S.H. Whitesides (Ed.): GD'98, LNCS 1547, pp. 246–262, 1998.

is explicitly parameterized with a metric of the layout space and there is no limitation on the metric that can be used. It is therefore directly useable for 2D and 3D-graph layout as well as for non-standard layouts, for example in non-rectangular viewing areas. Even specialized layout tasks like embedding a network into a spherical surface can directly be solved as we will demonstrate. The method presented is based on a competitive learning algorithm which is derived from well-known self-organization strategies of unsupervised neural networks, namely from Kohonen's self-organizing maps [18, 19, 20]. To our knowledge this is the first connectionist approach to graph layout.

At a first glance, a number of arguments are apparently speaking against the application of NN to problems such as graph layout: It is difficult to handle symbolic relations with NN and structures of potentially unlimited size are not easily accommodated in a NN. Some of these problems have recently been addressed by the neural folding architecture [11] and by adaptive structure processing [9]. However, our approach uses an entirely different way to overcome these limitations: We will not use an external network structure "to learn the graph", instead the graph itself will be turned into a learning network.

The central problem in graph layout is that it requires to solve computationally hard global optimization problems. As several excellent solutions for other computationally hard optimization tasks prove, optimization is one of the particular strengths of NN. Prominent examples are the travelling salesman problem or graph-theoretic problems like optimal bipartitioning [16]. It therefore seems promising to study NN for graph layout.

The main advantage of using a NN method for optimization problems is that we do not have to construct a suitable heuristics by hand. Instead, the network discovers the search heuristics automatically or—putting it less mysteriously—a meta-heuristics is built into the learning algorithm of the network. As a consequence, the implementation of the method is very simple.

2 Kohonen's Self-Organizing Maps

The model of self-organizing graphs which we are going to present is an extension of a well-established neural network type, namely Kohonen's self-organizing maps (SOM), which are a kind of unsupervised competitive network. We will therefore have to briefly review the basic idea of competitive learning before we can introduce self-organizing graphs. A more general introduction to neural networks is beyond the scope of this paper. The interested reader is referred to [16] and [1] as well as Kohonen's books [19, 20].

2.1 Competitive Learning

There are two different types of learning for NN: Supervised and unsupervised learning. Supervised learning requires an a-priori defined learning objective and a "teacher" external to the network. In the learning phase this teacher (or supervision procedure) judges how close the network's response is to the intended solution and makes appropriate adjustments to the network's connection strengths

(weights) in order to tune it towards the correct response. Once the learning phase is finished, this enables the network to perform according to the predefined objective function.

In unsupervised learning there is no teacher and no a-priori objective function: The net has to discover the optimization criteria itself. This, of course, means that a particular network type can only perform well for a particular kind of task. Typical application areas of unsupervised learning are: clustering, autoassociation, content-based retrieval, encoding, compression, and feature mapping. The best-known NN models of unsupervised learning are Hebbian learning [16] and the models of competitive learning: The adaptive resonance theory [15], and the self-organizing map or Kohonen network which will be explained in the following. Some discussion of the usage of unsupervised learning for visualization tasks can be found in [23]. In the case of graph layout, using an unsupervised learning method means that we will not teach the layout aesthetics to the network. Instead we will let the net discover the appropriate criteria itself.

The basic idea of competitive learning is that a number of output units compete for being the "winner" for a given input signal. This winner is the unit to be adapted such that it responds even better to this signal. In hard competitive a.k.a. "winner-take-all" learning only a single unit is adapted. In contrast, soft learning adapts several units at once. In a NN typically the unit with the highest response is selected as the winner.

The learning process therefore requires to elect a winner and to apply a selective weight adjustment in every learning cycle. This clearly sounds as if some kind of supervision procedure was needed and as if we were discussing a supervised learning scheme. Later we will indeed simplify (and accelerate) the learning method by using an external supervision procedure, but the same result can also be achieved with unsupervised learning. The key to managing the selection of the winner as well as a selective update without supervision is to use lateral and recursive network connections.

Most unsupervised competitive networks have a rather simple structure and since we are aiming at Kohonen networks, we will only discuss networks that consist of a single input layer and a single output layer (the so-called competitive layer) here. The competitive net is wired such that each unit in the competitive layer is connected to every input unit. Additionally each competitive unit is connected to itself via an excitatory (positive) connection and it is connected to all other units in the competitive layer via inhibitory (negative) connections (see Figure 1). As usual, the response of a unit is calculated as a (usually sigmoidal) output function σ of the net input to this unit, i.e. as a function of the sum of all signals received from the inputs $x_1, \ldots, x_k$ and via the lateral connections weighted by the respective connection strengths: $r_j = \sigma(\sum_{i=1}^{k} w_{j,i} x_i + \sum_{i=0}^{m} \hat{w}_{j,i} r_i)$. If this update is made repeatedly the network will exhibit a behaviour called *competitive dynamics*: Because of the lateral connections the node which initially has the greatest response to the input increases its own activation via self-excitation at the same time strongly inhibiting the other units via the inhibitory lateral connections. With a suitable choice of

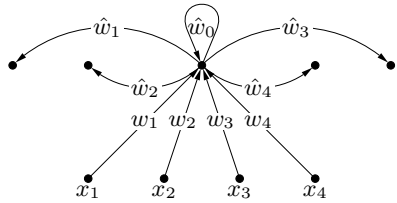

Fig. 1. Competitive Layer Schematics

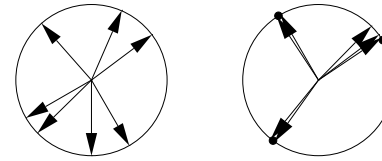

Fig. 2. Updates in Weight-Space

the activation function σ and the lateral weights, the network will for any given input eventually settle into an equilibrium where only the single winner unit u_j is active and all other nodes have their activation reduced to zero [13, 14].

It remains to be shown how a selective weight update can be performed without supervision. But since in the equilibrium state of an ideal competitive net only the single winner unit u_j is active (i.e. has a response $r_j > 0$), the update can just be enforced for every unit across the entire network provided it is weighted by the corresponding unit's response. In this way only the weights belonging to the winner unit will be updated. The entire competitive learning procedure can thus be formulated in the following way: (1) present an input vector $\boldsymbol{i}$ to the net and let the net settle into equilibrium, (2) for every node u_j enforce a weight correction $\Delta \boldsymbol{w}_j = \alpha\, r_j\, (\boldsymbol{i} - \boldsymbol{w}_j)$ where α is suitable learning factor.

So far we have regarded the units' responses as the desired output, but for our purposes it is much more interesting to switch to a different perspective and to look at the weights of the competitive units instead. In fact, from now on we will ignore the actual responses of the units altogether. Eventually we will transform our layout task into a problem in weight-space and solve it entirely there. Each weight set for a competitive unit u_j can be represented by a vector $\boldsymbol{w}_j = (w_{j,1}, \ldots, w_{j,n})$. If we assume that input vectors and weight vectors are normalized according to $\| \boldsymbol{w}_j \| = \sum_i w_{j,i}^2 = 1$ then these vectors can be represented by arrows to the surface of an n-dimensional unit hypersphere. In weight-space a learning step can now be interpreted as turning the weight vector of the winning unit towards the current input vector. Starting from an initial random distribution of the weight vectors, the network will therefore attempt to align its weight vectors with the input vectors it is seeing. In this way it obviously solves an instance of a clustering task: The weight vectors are clustered with the input vectors. Figure 2 shows an example: Starting from the initial random configuration on the left side the network moves towards the configuration on the right side of the same figure.

2.2 The Kohonen Network

If we switch from hard competitive learning to soft learning, several units may be adapted at once. The question is, which units shall be chosen for an update? If the network has some known spatial arrangement, one of the possibilities is

to update the winner together with its neighboring nodes. This is the learning algorithm used by Kohonen's self-organizing maps. In real biological neural networks the spatial arrangement is in fact important and not only the strength but also the location of a neural excitation conveys information. In order to understand the importance of the spatial organization we need to have some suitable metric for the sensory signals such that we can judge the proximity of two input signals. If we know the network's geometric arrangement (i.e. the spatial locations of the individual units) such an input metric enables us to analyze the relationship between proximity of input signals and spatial proximity of the resulting network excitation. Surprisingly, in the mammal brain it is often the case that spatially close regions of cells respond to input stimuli that are in proximity. Such mappings of metric regions of the input space to spatial regions of the brain (or, more abstractly, metric regions of the output space) are called *topology preserving feature maps* or *topographic maps*. Striking examples of such topographic maps in the mammal brain are the retinotopic map, where close regions of the retina are mapped to close regions of the cortex, and the somatosensory map, where close regions of the body surface are mapped to close regions of the somatosensory cortex. These are both spatial metrics, but examples of more abstract metrics can also be found. The tonotopic map from the ear to the auditory cortex, for example, works such that spatially close cells correspond to hearing similar frequencies. In fact, the seminal study in the field [17] established this kind of abstract mapping for the orientation receptor cells which react to specific orientations of visual stimuli (such as grids). Their spatial arrangement is such that cells located in proximity correspond to similar angles of stimuli. Since topographic maps are a common phenomenon, some self-organization mechanism that automatically performs the corresponding neural "wiring" is likely to exist. Von der Mahlsburg [26] succeeded first in showing that competitive learning can achieve this. Kohonen later extended and simplified the model [18, 19, 20], casting it into the computationally more adequate form of the so-called self-organizing map (SOM).

Kohonen's networks use two relatively simple spatial configurations: They are either rectangular or hexagonal grids implying an 8-neighborhood or a 6-neighborhood, respectively. The network structure again is a single layer of output units without lateral connections and a layer of n input units in which each output unit is connected to each input unit.

Since Kohonen networks are a computational method we can sacrifice the biological justification and simplify the situation by using an external supervision process to select the winner and to update its weights. In this case we do not need any lateral connections and we do not need to wait for the network to settle into a stable state so that the network response can be computed much faster, namely in constant time. In this way Kohonen's learning procedure can be formulated as:

1. present a stimulus vector $\boldsymbol{v}$ to the network,
2. find the unit u_j with the largest response r_j,

3. adapt the weights of u_j and all nodes in a neighborhood of a certain radius r, according to the function $\Delta \boldsymbol{w}_i = \eta(t)\, \Lambda(u_j, u_i)\, (\boldsymbol{v} - \boldsymbol{w}_i)$.
4. After every k-th stimulus decrease the radius r.

$\eta(t)$ is a time-dependent adaption factor and $\Lambda(u_j, u_i)$ is a neighborhood function the value of which decreases with increasing distance between u_i and u_j. Thus the winner is adapted strongly whereas the influence of the input diminishes with increasing distance from the winning unit. This process is iterated until the learning rate $\eta(t)$ falls below a certain threshold.

For the selection of the winner unit it is, in fact, not at all necessary to compute the units' responses. As Kohonen shows, the winner unit u_j can as well be taken to be the one with the smallest distance $\parallel \boldsymbol{v} - \boldsymbol{w}_j \parallel$ to the stimulus vector, i.e. $j = argmin_{i \in \{1,\ldots,m\}} \parallel \boldsymbol{v} - \boldsymbol{w}_i \parallel$. Both criteria turn out to be identical for normalized vectors.

We can think of the adaption as being determined by a "cooling" parameter η, which decreases the adaption with increasing training time, a "decay" parameter Λ, which decreases the adaption with increasing distance from the winning unit, and a "narrowing" parameter k which decreases the spatial extent of adaption over time. Kohonen demonstrates impressively that for a suitable choice of the learning parameters the output network organizes itself as a topographic map of the input. Various forms are possible for the parameter functions, but negative exponential functions produce the best results, the intuition being that a coarse organization of the network is quickly achieved in early phases, whereas a localized fine organization is performed more slowly in later phases. Therefore common choices are: $\Lambda(u_i, u_j) = e^{-d(u_i,u_j)^2/2\sigma(t)^2}$ and $\eta(t) = b\alpha t^{-\alpha}$, where $d(u_i, u_j)$ is the topological distance of u_i and u_j and σ is a time-dependent width factor of the form $\sigma(t) = a\alpha t^{-\alpha}$.

3 From Self-Organizing Maps to Self-Organizing Graphs

We have mentioned above that the first key to solving the layout task is to look at the network's behaviour in weight-space instead of at its responses. If we visualize the behaviour of the SOM in weight-space, the connection to graph layout will immediately become clear. Restricting the input to two dimensions, each weight vector can naturally be interpreted as a position in 2D-space.

Figure 3 illustrates the learning process of a SOM with 9 units in a rectangular grid. The 4-neighborhood is depicted by straight lines. The initial random weight distribution (left) eventually settles into an organized topographic map (right). Each unit has moved to one of the stimulus positions marked by black dots.

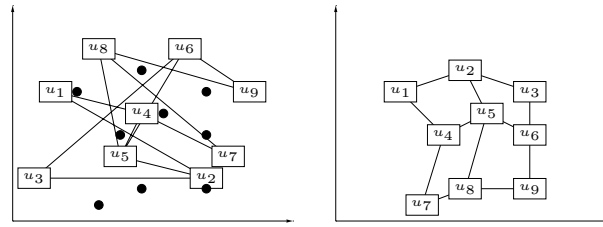

Fig. 3. Learning of a Topographic Map

If we interpret the network's weight-space in Figure 3 as the embedding of a grid-like graph, the connection of SOM learning to graph layout immediately becomes obvious: In its self-organization process the SOM has obviously created a reasonable, if not perfect, layout for its network graph. This is not incidental, since the optimization criteria for learning a topographic mapping and for performing a graph layout are quite similar under a suitable transformation of the layout problem.

As mentioned, the first key is to look at the weight-space instead of at the output response and to interpret the weight-space as a spatial embedding of the graph. Abstractly speaking, the SOM constructs a metric-preserving mapping from the m-dimensional weight-space onto the n-dimensional input space. Let us inspect this more closely and have a look at various ways to intuitively interpret the way the SOM works: To cite [16] "... we can think of a sort of elastic net in input space that wants to come as close as possible to the inputs; the net has the topology of the output array (i.e. a line or a plane) and the points of the net have the weights as coordinates." In other words, the learning process "stretches" the network such that its nodes cluster with the input positions while at the same time matching the topology of the network with the metric of the input space.

Clearly, this is quite similar to the task that we have to solve for graph layout: There we have to find an embedding of the nodes such that the Euclidean distance of a pair of nodes matches their graph-theoretic distance. The main difference is that in the case of the SOM we have only dealt with very simple, fixed 2D-network topologies, namely rectangular or hexagonal grids, whereas in graph layout we must handle arbitrary topologies which are different for every new task. The second key therefore is to realize that there are no restrictions on the topology that we can give to the SOM's network. The learning process will always attempt to construct a metric preserving mapping between the input space and the network topology. In fact, recent models of competitive neural networks for other problem domains, such as the growing neural gas [10], are also using different topologies or even topologies that evolve during the training process. *The idea therefore is to train a competitive network that has the same topology as the graph to be laid out.* Now each unit of the network can be identified with a node of the graph and each unit's weight vector can be interpreted as the spatial embedding of this node. *Analogously to the SOM, we expect such a network to settle into a configuration where the nodes are clustered with the input positions and where their Euclidean distances match their graph-theoretic distances.*

Note that a hidden shift of perspective was the third key to the solution: Instead of training the network to compute a certain input-output relation we are regarding the training phase as the intended computation. The network is never actually used once it is trained.

One question remains to be answered: Since the network clusters its weight vectors with the input stimuli it is seeing, the set of input vectors clearly determines the final layout. As can been seen in Figure 3, each node of the graph will move towards some stimulus position. So how can a reasonable set of input

stimuli be obtained? The solution to this problem is surprisingly simple: *A set of points that is distributed uniformly in the input area is used as the set of input stimuli.* Using a uniform distribution has the important property that it causes the net to stretch such that it uniformly fills the available input space (up to a small border zone that remains unused). Of course, it also makes the set of input stimuli independent from the actual graph to be laid out.

3.1 The ISOM Layout Algorithm

We are now set to detail the layout algorithm outlined above. The main differences to the original SOM are not so much to be sought in the actual process of computation as in the interpretation of input and output. First, the problem input given to our method is the network topology and not the set of stimuli. The stimuli themselves are no longer part of the problem description but a fixed part of the algorithm. Secondly, we are interpreting the weight-space as the output parameter. The actual network output is discarded completely. As a consequence, there is no activation function σ. Because of this inverted perspective the method is termed the *inverted self-organizing map (ISOM)*. Apart from these changes, we are using slightly different cooling and decay parameters which have proven useful in experiments.

Algorithm ISOM

input: a graph $G = (V, E)$, output: a spatial embedding of G
epoch $t := 1$;
radius $r := r_{max}$; /* initial radius */
cooling factor c;

forall $v \in V$ do $v.pos := random\_vector()$;
while $(t \leq t_{max})$ do {
    adaption α:= $max(min\_adaption, e^{-c(t/t_{max})} \cdot max\_adaption)$
    $\boldsymbol{i} := random\_vector()$; /* uniformly distributed in input area */
    $w := v \in V$ such that $\| v.pos - \boldsymbol{i} \|$ is minimal
    for w and all successors w_k of w with $d(w, w_k) \leq r$ do
        $w_k.pos := w_k.pos - 2^{-d(w,w_k)}\, \alpha\, (w_k.pos - \boldsymbol{i})$;
    $t := t + 1$;
    if t *mod interval* $= 0$ and $r > min\_radius$ do $r := r - 1$;
} end.

Note that the node positions $w_k.pos$ which take the role of the weights in the SOM are given by vectors so that the corresponding operations are vector operations. Also note the presence of a few extra parameters such as the minimal and maximal adaption, the minimal and initial radius, the cooling factor, and the maximum number of iterations. Suitable values have to be found experimentally.

3.2 Comparison to Force-Directed Layout

We will now show that the objective layout criteria implicitly used by the ISOM are closely related to those that are used in the well-established group of force-directed methods [8, 3].

Force-directed layout is minimizing the energy in the edges which are understood as springs attached to the nodes. These springs are used to model attraction forces and repellent forces between neighboring nodes. Computing a force-directed layout means to find a configuration where these forces are in balance. The ISOM, on the other hand, is optimizing the embedding for the following two objectives: (1) the graph-theoretic distance of all node pairs is matched with their metric distance and (2) a uniform space filling distribution of nodes is generated. We can understand (1) as the analogue of attractional forces, since neighboring nodes move towards the same stimulus positions, and (2) as the analogue of repellent forces, since the nodes are trying to drift apart in order to fill the space.

Intuitively the objectives of the ISOM and of basic force-directed layout are closely related and with both methods symmetries are automatically exhibited. One of the main differences is that the ISOM achieves the approximation of the implied computationally hard optimization problem as the byproduct of a stochastic self-organization process. This eliminates the need to find a good heuristic procedure for computing the equilibrium of forces and, in fact, the ISOM procedure is potentially faster.

The computation performed in a single iteration is inexpensive. The winner can be found in linear time (or in logarithmic time, if suitable spatial index structures are used). If the graph is not dense, only a constant number of successor nodes will be updated due to the limited radius. Each of these nodes can be accessed in constant time from its predecessor and the correction factor can also be computed in constant time. Thus, for a graph of bounded degree if the number of epochs and the initial radius is considered as fixed, the entire layout computation can be done in linear time. Even if we include the required number of iterations as a dependent variable in the complexity calculation, the overall complexity appears to be at most quadratic, since the experimental results indicate that it is sufficient to use a number of iterations that is linear in the size of the graph.

In contrast to force directed-models, the ISOM is easily adapted to any kind of layout space, since it can be explicitly parameterized with the metric of this space (see Section 5). On the other hand, some additional layout objectives, such as orthogonal drawings, are more easily introduced in force-directed models, since these can be modelled explicitly by extra forces [25].

4 Experimental Evaluation

Let us now give experimental results and some small examples, space not permitting more. We have implemented a Java applet which is available for further

exploration at `http://www.bounce.to/BerndMeyer`. The experiments confirm our theoretical expectations and show that the ISOM converges unexpectedly fast towards reasonable layouts. For medium-sized graphs (of up to approximately 25 nodes) typically not more than 500 epochs are required to produce a nice layout. The basic structural organization of the graph happens very fast in the early phases of the computation while later phases with small adaptions and radiuses mainly serve to refine the layout.

The choice of parameters can be important, but the ISOM seems fairly robust against small parameter changes and usually quickly settles into one of a few stable configurations. As a rule of thumb for medium-sized graphs, 500 epochs with a cooling factor $c = 0.4$ yield good results. The initial radius obviously depends on the size and connectivity of the graph. r_{max}=3 with an initial adaption of 0.8 was used for the examples. The interval for radius decrease has to be chosen such that most of the adaption happens while $r = 1$ (with adaptions of, say, $0.4 \ldots 0.15$). The final phase with $r = 0$ should only use very small adaption factors (approximately below 0.15) and can often be dropped altogether. A long phase for $r = 0$ with too high adaption factors is not advisable, since the symmetry of the layout may be destroyed. This is not surprising, since we know that a one-dimensional SOM will eventually be arranged as a space-filling peano curve, if phase $r = 0$ is active for too long. In fact, it is fairly obvious that the SOM algorithm for $r = 0$ reduces to mere vector quantization, since the topological structure of the network is no longer taken into regard.

As a first small extension to the basic model we can use different layout areas, such as non-rectangular regions. This is simply done by choosing a distribution of input stimuli that is uniformly distributed in this region. The only restriction on the shape of this area is that it must be convex, because otherwise edges might shortcut across regions where the layout area is caving in and even nodes might move outside of the layout area. We observe that using different layout areas leads to different layouts which are reasonably adapted to the available area. For example the layout of the complete graph K_6 in Figure 6 was generated with a triangular distribution, while Figure 7 was generated with a rectangular distribution.

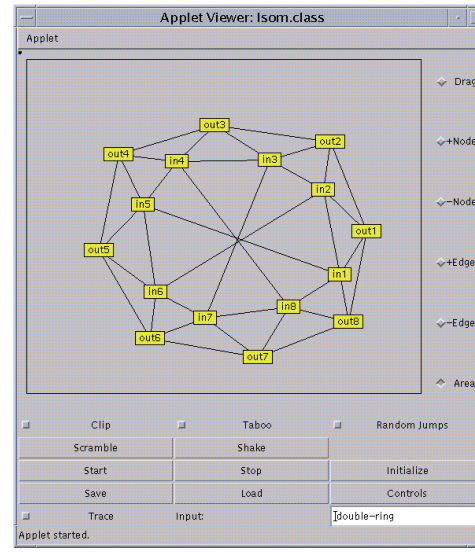

Fig. 4. The ISOM Applet at Work

Free trees can be handled as Figure 8 illustrates, but a drawback is that there is no straightforward possibility to draw a rooted tree according to the usual layout conventions with the basic algorithm.

5 Self-Organizing Graphs in 3D and on Spheres

An obvious extension that comes to mind is to use the same method for 3D-layout. The changes are straightforward: All that is required is to use 3D-vectors for input stimuli and weights. Apart form this, the algorithm remains unchanged. In most cases a reasonable 3D-structure is obtained (Figure 10). It is interesting to compare a 3D-layout with a layout of the same graph in 2D-space.

For a cube, for example, the 2D-ISOM generates exactly a 2D-projection of the layout generated by the 3D-ISOM (Figure 5, right) instead of the planar 2D-layout which would also be possible (Figure 5, left). This is because the non-planar layout conforms better to the criterion of uniform edge lengths.

Fig. 5. 3D-Structures in 2D-Space

Despite the fact that many layout methods give preference to planar layouts, this example may serve to illustrate that a planar layout is not always superior to a non-planar one. The important property of a good layout is to make the structure of the graph plainly recognizable. Depending on what the meaning of the graph is, this may sometimes even better be done with a non-planar layout.

Though the 3D-structure of a graph may already become apparent in a 2D-layout, a real 3D-layout is often required for more complex structures. We have implemented a prototype simulator for self-organizing 3D-graphs in Mathematica [27] which allows to look at the 3D-graph from different viewpoints or to generate movies that show a 3D-layout rotating in 3D-space. Experience shows that such possibilities are often required to fully recognize the 3D-structure on a 2D-display. While the above suggests that the ISOM works well in 3D-space, a critical assessment is in place. It is not really clear whether the optimization of layout objectives derived for 2D-space always leads to good 3D-drawings. This has also been observed by other authors [4]. In fact, the aesthetic criteria governing 3D-layout are not well-understood, since we are dealing with a different type of visual perception in 3D. In particular, we tend to interpret closed paths as surfaces. If the structure of the graph is already known, automatically generated 3D-layouts can deviate quite far from the expectation. On the positive side they often reveal additional structural properties (in particular symmetries) that would not be recognized in the preconceived layout. This can be illustrated with the layout in Figure 11 which could also be drawn as a cube inside of another cube with corresponding corners of the inner and outer cube connected.

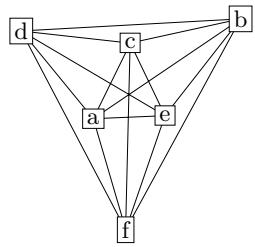

Fig. 6. The Complete Graph K_6

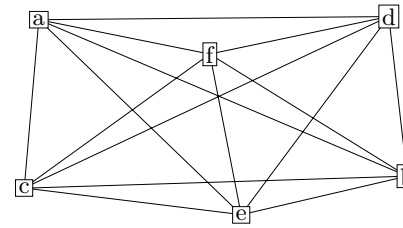

Fig. 7. A Rectangular Layout of K_6

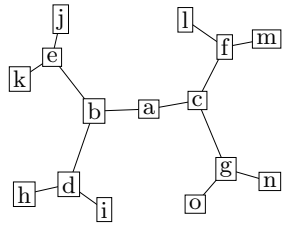

Fig. 8. A Layout of a Free Tree

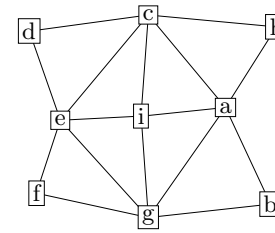

Fig. 9. Another Example

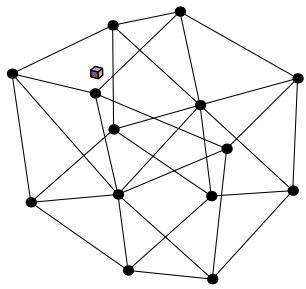

Fig. 10. Hexagonal Cylinder Layout

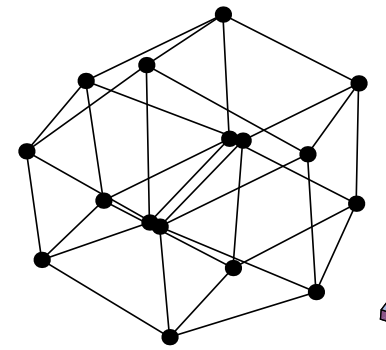

Fig. 11. A Cube Inside a Cube?

The ISOM is easily extensible to other special layout spaces. In particular, there are several interesting applications of layout on arbitrary spherical surfaces: (1) It can serve to utilize the display space more efficiently. A generalized fisheye view, for example, is but the projection of a uniform layout on some spherical surface. Real spherical 3D-layout opens even more interesting possibilities: A graph can, for example, be displayed on the surface of a globe that can be interactively rotated. This combines the space utilization of fisheye views with a novel interaction mode for graph exploration. (2) A layout on a spherical surface often provides a good alternative to non-planar straight line drawings without burdening the user with the entire complexity of understanding a 3D-layout on a 2D-display device (see Fig. 12). (3) Finally, a spherical layout may simply be part of the defined problem, such as a visualization of a world-wide network on a globe.

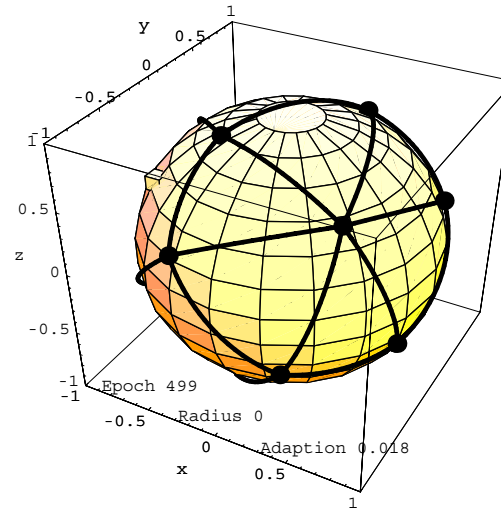

Fig. 12. Layout on a Unit Hypersphere

The so far implicit parameterization of the algorithm with the metric of the layout space is easily made explicit. The distance between u_i and u_j must now be defined as the length of the shortest curve on the surface that connects u_i and u_j and the adaption must be modified such that the node to be adapted moves along this curve by the appropriate amount. In most cases it is easy to define the curve connecting u_i and u_j in parametric form: $\boldsymbol{r}(t) = \{x(t), y(t), z(t)\}$ for $0 \leq t \leq 1$. Let $\boldsymbol{w}_i = \boldsymbol{r}(t_1)$ and $\boldsymbol{w}_j = \boldsymbol{r}(t_2)$. The length of the curve connecting u_i and u_j is then simply given by

$$s(\boldsymbol{w}_i, \boldsymbol{w}_j) = \left| \int_{t_1}^{t_2} \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2}\, dt \right|$$

Whenever this integral can be expressed in closed form (or numerically aproximated), the generalized ISOM below can be used for the particular surface. For a unit hypersphere the formulas become particularly simple. Since all shortest connections between two points u_i and u_j now lie on greater circles and the length of the connecting arc on a unit hypersphere is equal to the cosine of the associated central angle, the distance formula is reduced to: $s(\boldsymbol{w}_i, \boldsymbol{w}_j) = cos\,(\angle(\boldsymbol{w}_i, \boldsymbol{w}_j)) = \boldsymbol{w}_i \cdot \boldsymbol{w}_j$ The updated position thus is simply given by the normalized vector $\frac{\boldsymbol{r}}{\|\boldsymbol{r}\|}$ with $\boldsymbol{r} := w_k.pos - 2^{-d(w,w_k)}\,\alpha\,(w_k.pos - \boldsymbol{i})$.

Algorithm ISOM-generalized

input: a graph $G = (V, E)$, output: a spatial embedding of G
epoch $t := 1$;
radius $r := r_{max}$; /* initial radius */
cooling factor c;

forall $v \in V$ do $v.pos :=$ *random_vector*();
while $(t \leq t_{max})$ do {
  adaption α:= *max*(*min_adaption*, $e^{-c(t/t_{max})} \cdot$ *max_adaption*)
  $\boldsymbol{i} :=$ *random_vector*(); /* uniformly distributed on layout surface */
  $w := v \in V$ such that $s(v.pos, \boldsymbol{i})$ is minimal
  for w and all successors w_k of w with $d(w, w_k) \leq r$ do
    Let $r(t_1) = w_k.pos$;
      $w_k.pos := r(t_2)$ such that $\frac{s(r(t_2),\boldsymbol{i})}{s(r(t_1),\boldsymbol{i})} = 1 - 2^{-d(w,w_k)}\,\alpha$;
  $t := t + 1$;
  if t *mod interval* $= 0$ and $r >$ *min_radius* do $r := r - 1$;
} end.

A prototype version of self-organizing spherical graphs has also been implemented in Mathematica. Figure 12 shows one viewpoint of the spherical layout of a hexagonal cylinder as an example.

6 Extensions

A problem that can occur in some layouts are "collapses" or "clashes" in which two (connected or unconnected) nodes are moving towards the same position. Theoretically three types of clashes could occur: edge-edge clashes, node-edge clashes, or node-node clashes. Node-node clashes can be avoided by (1) choosing a larger layout area and (2) choosing different cooling factors or more epochs such that the final phase with $r = 0$ is extended and the node distribution becomes more "space filling". An alternative way to deal with node-clashes is a post-process which zooms in on the clashing cluster and generates a new local layout by applying ISOM layout only to the nodes in the zoomed area. Nodes outside of this area are ignored and remain unchanged during this post-process. Both solutions are not completely satisfying, primarily because they require intervention from the user. Also some structures are notorious for letting unconnected or unrelated nodes move towards the same position, and an alternative layout in which these nodes are located in entirely different places may be preferable depending on the context (think back to the example in Figure 11). Choosing different parameters does separate clashing nodes. It does not, however, find an entirely different structure. We are currently investigating the use of different types of decay functions Λ in order to achieve the desired effect. Particularly promising seems the usage of the so-called Mexican hat distribution (the solid curve in Figure 13) instead of the standard Gaussian distribution (the dashed

curve). Because the Mexican hat function falls below zero above a certain distance, it can be used to simulate a force that pushes unrelated nodes apart (i.e. nodes with a large topological distance). Of course, the usage of the clipping radius must be modified accordingly.

As for edge-clashes, since every edge-edge clash implies a node-edge clash, it is sufficient to eliminate the latter type. A simple way to achieve this is to use a post-process that substitutes each edge running through a node to which it is not adjacent by a curved edge that avoids the node.

There are a number of extensions that come to mind which we did not yet have a chance to experiment with sufficiently. A relatively straightforward extension concerns more richly structured graphs. If there are different edge types, some types may be interpreted as relating the adjacent nodes more closely than other types. In this case we would want nodes connected by such edges to be in a closer proximity. This could be enforced by giving different edge types different weights and using the induced weighted topological distance as the parameter of the decay function.

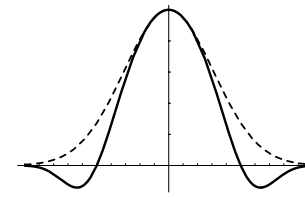

Fig. 13. Mexican Hat

The combination of the ISOM with additional layout constraints is worth further exploration. The basic idea is to supply a set of arithmetic constraints on the node positions against which updates must be verified. An update causing inconsitency must be rejected. As an alternative strategy the next best winner could be chosen. Instead of just checking consistency it may also be reasonable to accept—with a certain percentage of exceptions—only updates which reduce the degree of constraint violation. A constraint-enhanced ISOM would make it possible to explicitly avoid node clustering. More importantly, it would offer the possibility to take problem-specific layout constraints into regard. A layout of rooted trees, for example, should be fairly straightforward. It is clear, however, that this extension is computationally costly.

From a theoretical perspective a statistical analysis of the network behaviour is desirable. So far we have found suitable values for the layout parameters by experimentation. It would be a great improvement if we could develop a statistically justified heuristics for estimating suitable parameter values. For this the notion of the network's energy state would be useful. Though it is a customary and fruitful method to look at neural network learning methods from the point of view of energy minimization, this is rarely done for Kohonen networks. However, some steps towards this are reported in [22, 12]. Such a notion could be the key to a thorough theoretical analysis of the ISOM and might even reveal formal connections to force-directed layout.

7 Conclusions

The paper has introduced self-organizing graphs and the ISOM graph layout method which is based upon an extension of the competitive learning algorithm used in the self-organization of Kohonen networks. To our knowledge this is the first connectionist approach to automatic layout. We have presented an experimental evaluation and extensions of the basic model to 3D-layout and to generalized layout spaces such as spherical surfaces.

The advantages of the ISOM layout method, which has an extremely simple implementation, are its adaptability to different types, shapes, or dimensions of layout areas and even to different metric spaces. On top of this it consumes only little computational resources which renders it comparatively fast.

Since graph layout is a computationally hard optimization problem, it is interesting to note that many good solutions for such problems, such as simulated annealing, force-directed models, genetic algorithms, neural networks, and more recently ant colonies [7] are inspired by natural metaphors. So is the ISOM.

We are hoping that in the future we will be able to support our intuitive understanding of the ISOM's function by a more formal analysis based on the notion of an energy state of the network.

References

[1] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing.* MIT Press, Cambridge/MA, 1988.

[2] F.J. Brandenburg, editor. *Graph Drawing (GD'95).* Springer, Passau, Germany, September 1995.

[3] F.J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In [2], pages 76–87.

[4] I.F. Cruz and J.P. Twarog. 3D graph drawing with simulated annealing. In [2], pages 162–165.

[5] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, October 1996.

[6] G. DiBattista, P. Eades, R. Tamassia, and G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Journal of Computational Geometry Theory and Applications*, 4:235–282, 1994.

[7] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1):29–41, 1996.

[8] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[9] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. Technical Report 15/97, Universita di Firenze, Florence, 1997.

[10] B. Fritzke. Some competitive learning methods. Unpublished manuscript. www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/, April 1997.

[11] C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *International Conference on Neural Networks (ICNN-96)*, 1996.
[12] G.J. Goodhill and T.J. Sejnowski. A unifying objective function for topographic mappings. *Neural Computation*, 9:1291–1303, 1997.
[13] S. Grossberg. Adaptive pattern classification and universal recoding: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:121–134, 1976.
[14] S. Grossberg. How does the brain build a cognitive code? *Psychological Review*, 87:1–51, 1980.
[15] S. Grossberg. Competitive learning: from interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63, 1987.
[16] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City/CA, 1991.
[17] D. Hubel and T. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:173–181, 1962.
[18] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
[19] T. Kohonen. *Self-Organization and Associative Memory*. Springer, New York, 1989.
[20] T. Kohonen. *Self-Organizing Maps*. Springer, New York, 1997.
[21] C. Kosak, J. Marks, and S. Shieber. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(2):440–454, 1994.
[22] S.P. Luttrell. A bayesian analysis of self-organizing maps. *Neural Computation*, 6:767–794, 1994.
[23] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge/MA, 1996.
[24] K. Sugiyama. A cognitive approach for graph drawing. *Cybernetics and Systems: An International Journal*, 18:447–488, 1987.
[25] R. Tamassia. Constraints in graph drawing algorithms. *Constraints, An International Journal*, 3:87–120, 1998.
[26] C. von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85–100, 1973.
[27] S. Wolfram. *The Mathematica Book, Third Edition*. Cambridge University Press, Cambridge/MA, 1996.