Self-Organizing Maps for Drawing Large Graphs

Eric Bonabeau
Florian Hénaux

SFI WORKING PAPER: 1998-07-066

SFI Working Papers contain accounts of scientific work of the author(s) and do not necessarily represent the views of the Santa Fe Institute. We accept papers intended for publication in peer-reviewed journals or proceedings volumes, but not papers that have already appeared in print. Except for papers by our external faculty, papers must be based on work done at SFI, inspired by an invited visit to or collaboration at SFI, or funded by an SFI grant.

©NOTICE: This working paper is included by permission of the contributing author(s) as a means to ensure timely distribution of the scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the author(s). It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author’s copyright. These works may be reposted only with the explicit permission of the copyright holder.

www.santafe.edu
Self-organizing maps for drawing large graphs

Eric Bonabeau¹, Florian Hénaux²

(1) Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA and ATR Adaptive Communications Research, Seika-cho Soka-ku, Kyoto 619-02, Japan, email: bonabeau@santeafe.edu
(2) ENST, 46 rue Barrault, 75634 Paris Cédex 13, France

Abstract

Self-organizing maps (SOM) are unsupervised, competitive neural networks used to project high-dimensional data onto a low-dimensional space. In this article we show how SOM can be used to draw graphs in the plane. The SOM-based approach to graph drawing, which belongs to the general class of force-directed algorithms, allows the drawing of arbitrary weighted graphs. It is particularly efficient to draw large graphs and can be used as a preprocessing step before application of a more sophisticated method.

Keywords: neural networks, self-organizing maps, graph drawing, force-directed algorithm
Self-organizing map for drawing large graphs

Eric Bonabeau1,3, Florian Henaux2,4

(1) Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA
(2) ENST, 46 rue Barrault, 75634 Paris Cédex 13, France
(3) email: bonabeau@santafe.edu
(4) email: henaux@email.enst.fr

Keywords: neural networks, self-organizing maps, graph drawing

1. Introduction

Complex networks of relationships, that arise in numerous disciplines, can often be represented by graphs [7,11]. Such graphs can be weighted, possibly with negative weights, directed or undirected, etc. It is usually convenient to be able to visualize graphs in the form of diagrams, preferably embedded in a two-dimensional space, where interpretation by a human observer is facilitated. Automated procedures for drawing graphs are extremely useful, especially for large graphs. Many graph drawing algorithms have been designed in the past three decades [12,13], but, as emphasized by Kamada and Kawai [8], few of those algorithms deal with arbitrary graphs. We propose in this letter such an algorithm, based on self-organizing maps (SOM) [9,10], which are unsupervised, competitive neural networks that allow the projection of high-dimensional data onto low-dimensional spaces, while preserving characteristic features of the data. We show that self-organizing maps provide a natural framework for graph drawing, if the local topology is variable instead of being fixed, that is, if nodes in the network
have a variable number of neighbors. For example, when presented with a uniform distribution of points in the plane, a regular two-dimensional grid (a special type of graph) undergoes an ordering process and unfolds to approximate the distribution: the stationary state is a nice two-dimensional representation of the grid, provided there is no topological defect, generally a twist [6]. It seems that this result could be generalized to graphs which are not regular grids: for example, a random graph comprised of densely connected clusters should unfold to approximate the uniform planar distribution and at the same time should be constrained by its connectivity structure. If a given node drags the nodes which it is connected to, the approximation of the uniform distribution is frustrated because the graph is not homogeneous; but this frustration is precisely what allows the clusters to be clearly visualized.

This approach is close in spirit to the spring model for graph drawing [1,3,8,13], where graph vertices are connected by springs: drawing the graph then amounts to letting the dynamical system converge towards an equilibrium state of minimal energy. In this process, connection length between two vertices is optimal when it corresponds to the spring’s characteristic length. The main difficulty is to define the characteristic length of a spring, which should correspond to an appropriate distance in the graph [8]. While our approach is also based on a dynamical system of vertices that drag their neighbors along when they move, the appropriate graph distance is already implicitly built-in. Moreover, our algorithm is extremely easy to implement, works with negative connection weights or directed graphs, applies to very large graphs and accommodates "constrained graph drawing", where vertices are forced to lie on a specific subset of the plane. On the other hand, this algorithm does not perform as well on small graphs, uses straight lines only, and, because it is stochastic, reaches a dynamic, rather than static, equilibrium. This type of approach
has been used for VLSI placement [2,5,14], but has never been studied as a general heuristic for arbitrary graph representation.

2. Model

Let \( G = (V, E) \) be a non-directed graph. \( V = \{v_i\}_{i=1}^{n} \) is the set of \( n \) vertices and \( E \), a subset of \( V \times V \), the set of edges, of cardinal \( |E| \). \( E \) can be represented as a matrix \( [e_{ij}] \) of edge weights, where \( e_{ij} \neq 0 \) if \( \{v_i, v_j\} \in E \) and \( e_{ij} = 0 \) if \( \{v_i, v_j\} \notin E \). \( e_{ij} \) is the weight of edge \( \{v_i, v_j\} \) (\( e_{ii} = 1 \) by convention). The first-order neighborhood of \( v_i \) is defined by

\[
N^1(v_i) = \{v_j \in V \setminus N^0(v_i) | e_{ij} = 0 \},
\]

where, by convention, \( N^0(v_i) = \{v_i\} \). The \( \gamma \)-th order neighborhood of \( v_i \) is defined by recurrence by

\[
N^\gamma(v_i) = \{v_j \in V \setminus \bigcup_{k=0}^{\gamma-1} N^k(v_i) | \exists v_k \in N^{k+1}(v_i), e_{ij} = 0 \}.
\]  

Let \( w_i \in \mathbb{R}^2 \) be the positional vector of vertex \( v_i \), and \( x_m \in \mathbb{R}^2 \) be the \( m \)th training vector, where training vectors are drawn according to a uniform spatial distribution over, for example, a square area of linear size 1. At each time step, a training vector is presented. A vertex \( v_{i(m)} \) is selected. \( i(m) \) is defined by

\[
i(m) = \arg \min_{\forall v_j} \|x_m - w_j\|,
\]

where \( \| \cdot \| \) is the \( L_1 \) norm in the plane. Let \( T \) and \( m \geq 1 \) be the total simulation time and current time, respectively, expressed in number of training vectors. The coordinates of \( v_{i(m)} \) and of all vertices belonging to \( \bigcup_{v_j \in N^\gamma(v_{i(m)})} N^\gamma(v_{i(m)}) \), where \( f(m) \) is an integer function of \( m \), are updated:
\[ w_{i(m)}(m+1) \leftarrow w_{i(m)}(m) + \alpha(m) (x_m - w_{i(m)}(m)) \]  
\[ w_{i}(m+1) \leftarrow w_{i}(m) + \gamma^{-1} e_{i(m)} \beta(m) (w_{i(m)}(m+1) - w_{i}(m)) \] 

for \( \forall j \in N^3(v_{i(m)}), \gamma \leq \xi(m). \)

\( \alpha(m) \) and \( \beta(m) \) are two decaying functions: \( \alpha(m) = \alpha_t T/(\eta m + T) \) and \( \beta(m) = \beta_t T/(\eta m + T) \), where \( \eta, \alpha_t \) and \( \beta_t \) are tunable parameters. \( f(m) \) is used to tune the range of influence of the winning node. In the examples given below, we use \( f(m)=1 \) only for geometric graphs, where it is necessary because connections are by definition local in the plane, and therefore the range of influence of a node is too localized if \( f(m)=1 \); otherwise, this procedure is computationally expensive, and \( f(m)=1 \) is sufficient. Notice that in equation (4) vertices are updated so as to get closer to \( v_{i(m)} \) and not to \( x_m \) as is usual in self-organizing maps [6,9,10]. Although this procedure affects how well the distribution of training vectors is approximated, the algorithm is not aimed at accurately learning the distribution but at clearly visualizing clusters of vertices: the distribution of training vectors is a tool and not a goal. The \( e_{i(m)} \) term in equation (4) allows to treat the case or arbitrary connection weights: for example, if \( e_{i(m)}>0 \), \( v_j \) gets closer to \( v_{i(m)} \), while if \( e_{i(m)}<0 \), \( v_j \) is moved further away from \( v_{i(m)}. \)

3. Results

We have tested the algorithm on random graphs \( G(n, c, p_c, p_e) \) [4], where \( n \) is the number of vertices per cluster, \( c \) is the number of clusters, \( p_c \) is the probability that two vertices within a cluster are connected, and \( p_e \) is the probability that two vertices belonging to different clusters are connected. We set for simplicity \( e_i=1 \) if
\((v_i,v_j) \in E\) and \(e_i=0\) if \((v_i,v_j) \notin E\). Training vectors are presented according to a uniform distribution over the portion of plane considered \([(0,1) \times (0,1)]\). Initially, all vertices are randomly placed in this area. In order to avoid some vertices oscillating back and forth on the plane because they are subject to conflicting constraints, \(\beta(m) = \exp[-\frac{1}{\tau}d_i-w_i\sum_{j=1}^{n} d_j]\) for \(v_i \in N_k(v_{(m)})\), where \(d_i\) is a characteristic distance beyond which it is pointless to drag nodes. Fig. 1 illustrates the fact that the algorithm is able to find "natural" clusters in a graph \(G(25,4,0.8,0.01)\): Fig. 1a shows the initial random distribution of nodes on the plane; Fig. 1b shows that, after \(T=2000\) training vectors, vertices distribute themselves in space in such a way that exactly \(c=4\) clusters of vertices appear, that correspond to the four clusters of the graph.

We can evaluate the algorithm considered as a mapping from graph space onto the plane: we expect nodes which are far apart (respectively close) in the graph to lie far away from (respectively close to) each other on the plane, and vice versa. Let us introduce the \(L_2\) distance \(d(v_i,v_j)\) between two nodes in a graph:

\[
d(v_i,v_j) = n^{-1}\sum_{k=1}^{n}|e_{i_k} - e_{j_k}|
\]

In the present case, where \(e_i=0\) or 1, we have \(d(v_i,v_j) = n^{-1}|D(p(v_i),p(v_j))|\), where \(p(v_i) = \{v_j \in V : e_{i,j} \neq 0\}\) is the set of edges adjacent to \(v_i\), including \(v_i\), \(D\) is the symmetric difference \((D(A,B)=(A \cup B)-(A \cap B))\), and \(|.|\) denotes the number of elements of a set. The use of \(D\) expresses the fact that vertices that possess many common adjacent edges are close in the graph, whereas those with only a few or no adjacent edges in common are far apart in the graph. In a random graph of the type studied here, we expect to have two characteristic distances in the graph: the
distance between vertices that belong to the same cluster, and the distance between vertices that do not belong to the same cluster. Fig. 1a shows \[ \| w_i - w_j \| \] as a function of \( d(v_i, v_j) \) for the same graph and the same distributions of vertex positions in the plane as in Fig. 1a: the two expected clouds of points are observed, and there is, of course, clearly no correlation between \( \| w_i - w_j \| \) and \( d(v_i, v_j) \). Fig. 2b shows the same relationship after \( T=2000 \) training vectors (vertex positions are those of Fig. 1b): the distance between the positions of vertices in the plane representation is an increasing function of their distance in the graph, with two clouds of points corresponding to the two characteristic distances in the graph. The algorithm is able to generate a good mapping of the graph onto the plane.

The vertices can be constrained to lie on specific portions of the plane by modifying the distribution of training vectors: if, for example, training vectors are drawn according to a distribution which is uniform over the union of disconnected regions, vertices will be located in these regions (which must not be too far apart, however, otherwise some vertices may lie between regions). Fig. 3 illustrates this idea: a graph \( G(25,4,0.8,0.01) \) is represented with the distribution of training vectors uniform over a set of 4 disconnected square zones of equal size: all vertices are assigned to the different zones in such a way that vertices belonging to the same cluster in the graph tend to be located in the same region. Because the algorithm is stochastic, it is impossible to ensure that all nodes lie in the desired regions, but most of them do. Extension to more complicated regions, such as concentric circles or parallel stripes, is straightforward. The algorithm works very rapidly: 500 to 5000 training vectors were sufficient to converge in the examples we studied (in the present case, convergence means that all nodes have been assigned to the appropriate regions). An evaluation of the algorithm's convergence speed indicates that time complexity increases sublinearly with graph size, up to 16000
vertices, for this class of graphs [4]. Notice that, because the dynamics of the graph representation is driven by the distribution of training vectors, the algorithm does not perform well on small graphs: there must be a sufficient number of vertices for the training vector distribution to be reasonably well approximated without too much frustration.

Fig. 4b shows the representation of a randomly generated geometric graph with 200 vertices, shown on Fig. 4a, found by the algorithm. In geometric graphs, which are used in the context of numerical modeling, vertices, characterized by Euclidean coordinates, are located on a plane and are locally connected to other vertices that lie within a radius of connectivity R (here, R=0.18). Vertices are initially randomly distributed in a small area located in the center of the size 1 square. We set \( f(m) = 3 \) for \( 1 \leq m \leq T = 2000 \) to increase the range of influence of nodes. The unfolding of the graph to approximate the training vector distribution leads to a remarkable representation where vertices are located on the plane in such a way that connections are local.

Fig. 5 illustrates how the algorithm works when there are negative weights. A graph \( G(25, 4, 0.8, 0.01) \) has been generated with \( e_{ij} = 1 \) if vertices \( i \) and \( j \) belong to the same cluster, and \( e_{ij} = -1 \) if vertices \( i \) and \( j \) do not belong to the same cluster. The effect of the repulsive interactions among vertices from different clusters can be clearly seen on Fig. 5, where negative connections are represented by dashed lines.

Finally, the algorithm also works on directed graphs, if mother-vertices (with outgoing edges) drag daughter-vertices (with incoming edges) when they move. To illustrate how the algorithm can represent directed graphs, a 81-vertex tree has been constructed, where each vertex is connected to three daughter-vertices, down
to leaves which are not connected to any vertex. It is assumed that connections point towards daughter-vertices. Fig. 6a shows a representation of the tree obtained when the directed nature of the graph is not taken into account, that is, when connections are considered to be symmetric. Although the ternary structure of the tree is visible in some places, especially terminal branches, the tree structure is better represented when the algorithm works on non-symmetric connections (Fig. 6b).

4. Discussion

We have presented an algorithm for drawing large arbitrary graphs, based on self-organizing maps with variable local topology. Although the algorithm does not explicitly include sophisticated aesthetic criteria, it unfolds arbitrary graphs in a process that results in usually clear and useful representations. In particular, it is able to clearly detect natural clusters in graphs. Because this algorithm is very easy to implement and appears to be computationally efficient, we suggest that it could be used as a preprocessing step before application of more refined methods. This is especially true for large graphs.

Acknowledgements

E. B. is supported by the Interval Research Fellowship at the Santa Fe Institute. E. B. and F. H. acknowledge support from the GIS (Groupe d'Intérêt Scientifique) Sciences de la Cognition.

8
References


Figure captions

Fig. 1: (a) Initial distribution of vertices on the portion of plane \([0,1] \times [0,1]\), for a random graph \(G(25,4,0.8,0.01)\). (b) The algorithm is able to find "natural" clusters in this graph. Training vectors are presented according to a uniform distribution over \([0,1] \times [0,1]\). Vertices distribute themselves in space in such a way that exactly \(c=4\) clusters of vertices appear, that correspond to the 4 clusters of the graph. \(T=2000, f(m)=1\). For \(m \leq T/3\): \(\alpha(m) = \alpha T/(\eta m + T)\) and \(\beta(m) = \beta T/(\eta m + T), \eta=50, \alpha_i=0.6, \beta_i=0.4\). For \(m > T/3\), \(\beta(m) = \exp\left(-\frac{1}{\eta m + T}d_v\right)\) for \(v_i \in N_i(v_{1,m})\), \(d_v=0.17\).

Fig. 2: (a) \(|w_i - w_j|_1\) vs \(d(v_i, v_j)\) for the graph representation of Fig. 1a. \(|\cdot|_1\) is the \(L_1\) norm in the plane, and \(d(\cdot, \cdot)\) is the \(L_1\) norm for graphs. (b) \(|w_i - w_j|_1\) vs \(d(v_i, v_j)\) for the graph representation of Fig. 1b.

Fig. 3: Same as Fig. 1b, but the distribution of training vectors is uniform over a set of 4 disconnected square zones of equal size, represented by dotted lines. All vertices are assigned to the different zones in such a way that all vertices belonging to the same cluster in the graph are located in the same region.

Fig. 4: (a) Geometric graph of 200 vertices in a square area of linear size 1, where the radius of connectivity is \(R=0.18\) (\(L_1\) norm in the plane). (b) Representation of the graph found by the algorithm at \(T=2000\). Same parameters as Fig. 1b except that \(f(m) = 3\).

Fig. 5: Same as Fig. 1b, but \(\varepsilon_i=1\) if vertices \(i\) and \(j\) belong to the same cluster, and \(\varepsilon_i=-1\) if vertices \(i\) and \(j\) do not belong to the same cluster. Negative connections are represented by dashed lines. The effect of the repulsive interactions can be clearly seen.

Fig. 6: (a) Representation of a directed graph (a 81-vertex tree with each vertex being connected to three daughter-vertices) obtained with a non-directed algorithm. (b) Representation of the same directed graph obtained with a directed algorithm.
Fig. 3
Fig. 6b