A grid layout algorithm for automatic drawing of biochemical networks

Weijiang Li and Hiroyuki Kurata*

Department of Bioscience and Bioinformatics, Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

Received on November 11, 2004; revised on January 16, 2005; accepted on January 24, 2005

Advance Access publication January 27, 2005

ABSTRACT

Motivation: Visualization is indispensable in the research of complex biochemical networks. Available graph layout algorithms are not adequate for satisfactorily drawing such networks. New methods are required to visualize automatically the topological architectures and facilitate the understanding of the functions of the networks.

Results: We propose a novel layout algorithm to draw complex biochemical networks. A network is modeled as a system of interacting nodes on squared grids. A discrete cost function between each node pair is designed based on the topological relation and the geometric positions of the two nodes. The layouts are produced by minimizing the total cost. We design a fast algorithm to minimize the discrete cost function, by which candidate layouts can be produced efficiently. A simulated annealing procedure is used to choose better candidates. Our algorithm demonstrates its ability to exhibit cluster structures clearly in relatively compact layout areas without any prior knowledge. We developed Windows software to implement the algorithm for CADLIVE.


Contact: kurata@bio.kyutech.ac.jp


1 INTRODUCTION

Rapid advances in molecular biology have elucidated a comprehensive and concrete map for gene regulatory networks and signal transduction pathways, which are large, complicated and hard to understand meaningfully without computer-aided analysis tools. Comprehensive visual representations of the networks are necessary to help researchers gain insight into the complex network data. This stimulated the interest in developing computer analysis tools that support visualization of biochemical networks. Graphical user interfaces (GUIs) are needed to handle large-scale biological network data. Available software includes BioCharon (Kirkwood et al., 2003), CADLIVE (Computer-Aided Design of LiVing systems) (Kurata et al., 2003), CellDesigner (Funahashi et al., 2003; Kitano et al., 2003), Cellerator (Shapiro et al., 2003), Cellware (Dhar et al., 2004), Cytoscape (Shannon et al., 2003), Gepasi (Mendes, 1997; Mendes et al., 2003; Mendes and Kell, 2001), Jarnac and JDesigner (Sauro, 2003), NetBuilder (Brown et al., 2002) and Simpathica (Antoniotti et al., 2003) (http://sbml.org/index.jsp).

Out of these, CADLIVE is an effort that determines the graphical notation for biochemical networks (including modified molecules and complexes) and builds large-scale biochemical maps (Kurata et al., 2003). CADLIVE describes a biochemical network with a set of regulator-reaction equations in an extended Systems Biology Markup Language (SBML) level 2 format (Hucka et al., 2003).

An important feature of the GUI software is the ability to display networks that aid users in understanding network architectures. Using this software, gene regulatory networks and signal transduction pathways can be visualized intuitively through the comprehensive graphical notations. Obviously, it is tedious and labor-intensive to make intuitive and heuristic layouts for complex networks. This is a common problem that needs to be solved. However, automatic drawing methods are absent; the network components must be placed manually.

A biochemical network can be well modeled as a graph. A scientific goal of drawing a graph is to facilitate human perception of its topological structure. This can be achieved by optimizing a specific cost function of the graph to fulfill certain criteria. There are a large number of standard graph layout algorithms using this approach (Vismara et al., 2000), but few direct applications of these algorithms have been reported that make satisfactory drawings of the concrete biochemical networks.

Several layout algorithms for visualizing metabolic pathways have been published in recent years (Karp and Paley, 1994; Sala-monsen et al., 1999; Becker and Rojas, 2001; Schreiber, 2002; Goesmann, 2002; Breitkreutz, 2003; Hoflord et al., 2004; Hu, 2004), since standard graph drawing techniques cannot make conventional textbook-like drawings. These algorithms focus on how to place the objects and route their connections to render layouts in traditionally accepted styles.

In a metabolic pathway, metabolites are the nodes, and enzymes are treated as the edge labels. Usually, a few linear or circular main paths form the ‘backbones’ of the layout. Drawing backbones is the focus of these algorithms. Once the backbone nodes are properly positioned, arrangement of other components is relatively easy. By contrast, in a biochemical network that consists of various reactions (transcription, translation, complex formation, modification and transportation), various kinds of molecules (enzymes, modified molecules and complexes) are regarded as nodes, which makes the topological structure more complex. For gene regulatory networks or signal transduction pathways, it is difficult to find any primary paths acting as backbones. In such cases, the focus is to show the functional modules, which consist of closely related nodes.

*To whom correspondence should be addressed.
Therefore, the visualization techniques for metabolic pathways are generally not applicable to other networks.

Force-directed layout algorithms are a widely used graph layout method. The basic idea is to model a graph as a mechanical system where the nodes are repulsive particles and the edges are attractive interactions. There is an additional attractive force between two nodes only when they are connected by an edge. All the repulsive and attractive interactions are represented by an energy function (cost function). A layout is determined when the forces drive the system to a steady state (a local minimum of energy). Force-directed layout algorithms have been used for visualizing large-scale maps of coarse-grained networks, such as biological similarity relationship networks (Enright and Ouzounis, 2001) and protein–protein interaction networks (Ju and Han, 2003; Han and Ju, 2003; Ju and Han, 2003).

Although an abundance of data is contained in this type of drawing, little information about the network topological structure can be gained without further transformation.

When applied to biochemical networks, two problems with force-directed layout methods become evident. The first involves identification of cluster structures. A cluster consists of a set of closely related nodes, which usually form a functional module. Therefore, all nodes within a cluster are expected to be placed in adjacent geometrical positions in a layout. The major shortcoming of force-directed layouts is that the cluster structures of networks cannot be displayed clearly because the nodes that are not directly connected repulse each other; thus appropriate cluster structures are difficult to form. The second problem involves the overall geometric distribution of nodes. In a force-directed layout, distances between nodes are not well proportioned. Dominant repulsive interactions make the distribution of nodes scattered, while the minimal distance between nodes is often too short compared with the drawing area size. This is disadvantageous for making compact layouts.

To overcome these drawbacks, we propose a new algorithm, called grid layout algorithm, where a network is treated as a system of interacting particles (nodes) that are placed on a square grid. The nodes interact according to a specified cost function, which is designed based on the topological structure of the network. In such a system, closely related nodes attract each other and remotely related nodes repulse each other. This facilitates the forming of correct cluster structures in the layouts. Since all nodes are placed on grid points, the problem of nodes being placed too close together does not exist.

2 GRID LAYOUT AS AN OPTIMIZATION PROBLEM

We express our network layout method as an optimization algorithm that aims at optimizing the solutions of a certain objective cost that is a function of layouts of the input network. The network is modeled as a system of interacting particles which are placed on a 2D square grid. The network is confined within an area, \( L \). The particles (nodes) interact according to a predefined energy function based on the network topological structure. A configuration of the particles represents a layout of the network, where all edges are straight lines. The energy of a configuration is the cost score of the corresponding layout. A stable configuration has low energy; equivalently, an acceptable layout has a low cost score.

2.1 Cost function

Denote a layout of the given network by the vector of coordinates \( \mathbf{R} = (r_1, r_2, \ldots, r_n) \), where \( n \) is the number of nodes in the network and \( r_i = (x_i, y_i) \) the coordinate of a node \( i \). In a grid layout, all nodes are placed on grid points, i.e., all \( x_i \)s and \( y_i \)s are integers. The layout of a network is evaluated by the cost function, which is defined as the sum of the costs between all pairs of the nodes:

\[
f(\mathbf{R}) = \sum_{i<j} f_{ij}(\mathbf{R}).
\]

The cost between nodes \( i \) and \( j \) is assumed to be

\[
f_{ij}(\mathbf{R}) = w_{ij}d(r_i, r_j),
\]

where \( w_{ij} \) is the cost weight between nodes \( i \) and \( j \) and \( d(r_i, r_j) \) represents some ‘distance’. The distance is not necessarily Euclidean. In this paper, it is provided by

\[
d(r_i, r_j) = |x_i - x_j| + |y_i - y_j|,
\]

which has the advantage of less computation.

2.2 Weight matrix

Obviously, in such a framework of the cost function, the weight matrix is an essential part. The simplest specification is to set the weight matrix to the adjacency matrix: \( w_{ij} = 1 \) if nodes \( i \) and \( j \) are connected; \( w_{ij} = 0 \), otherwise. In this scheme, all connected nodes attract each other with equal weights. As a result, nodes in a densely connected part of the network will be located close to each other in the refined layouts. On the other hand, all nodes of a connected network (there is at least one path connecting every pair of nodes) tend to congregate in a compact area. Consequently, the cluster structures of networks cannot be shown clearly. To avoid this pitfall, more complex weight matrices are required.

In general, nodes with strong relations are expected to be placed close together and weakly related nodes are expected to be placed far apart. To achieve this, the weights of closely related nodes should be positive and the weights of weakly related nodes should be negative.

The question is then, ‘How to classify a pair of nodes as closely or weakly related?’ A convenient criterion is based on the number of short paths connecting them. Closely related nodes should have short paths between them. Two nodes are weakly related when they cannot be connected by any short path. Suppose

\[
M^{(k)} = (A + I)^k,
\]

where \( I \) is the unit matrix and \( A \) the adjacency matrix of the network:

\[
A_{\alpha\beta} = \begin{cases} 1, & \text{if nodes } \alpha \text{ and } \beta \text{ are connected;} \\ 0, & \text{otherwise.} \end{cases}
\]

The quantity \( M^{(k)}_{\alpha\beta} \) is the number of \( k \)-paths between nodes \( \alpha \) and \( \beta \). (A \( k \)-path is a path whose length is not > \( k \)). The weight matrix based on \( M^{(k)} \) is as follows:

\[
w_{ij} = \begin{cases} 3, & \text{if } M^{(1)}_{ij} > 0; \\ 1, & \text{if } M^{(2)}_{ij} = 0 \text{ and } M^{(1)}_{ij} > 0; \\ 0, & \text{if } M^{(3)}_{ij} = 0 \text{ and } M^{(2)}_{ij} > 0; \\ -1, & \text{if } M^{(3)}_{ij} = 0 \text{ and } M^{(1)}_{ij} > 0; \\ -2, & \text{otherwise.} \end{cases}
\]

In this scheme, two nodes will attract each other if there are short paths between them. The shorter the paths are, the stronger the attraction will be. Nodes that are not connected by any short paths (the
shortest path length > 4) will repulse each other. To prevent the tendency that repulsive nodes depart infinitely, an upper limit for the repulsive term should be set. The cost between nodes \( i \) and \( j \) is modified to
\[
f_{ij}(\mathbf{R}) = \psi_{ij}(\mathbf{r}_i, \mathbf{r}_j),
\]
where
\[
\psi_{ij}(\mathbf{r}_i, \mathbf{r}_j) = \begin{cases}
    w_{ij}d(\mathbf{r}_i, \mathbf{r}_j), & \text{if } w_{ij} \geq 0; \\
    w_{ij} \min(d(\mathbf{r}_i, \mathbf{r}_j), d_{\text{max}}), & \text{otherwise}.
\end{cases}
\]
\(d_{\text{max}}\) is called the saturated repulsive distance.

### 2.3 Least changes and local minima of the discrete cost function

We call an arbitrary layout a raw layout. There is a huge population of raw layouts for each actual network. The probability that a raw layout happens to be satisfactory is so small that it is almost impossible to choose an acceptable layout from randomly generated layouts. One possible approach to solving the problem is to choose acceptable layouts from some refined layouts instead of raw layouts. In this approach, an efficient method is necessary to significantly reduce the cost scores of raw layouts.

The cost score of an acceptable layout should be low, but no a priori criterion is available to determine whether a cost score is sufficiently low. In the absence of an absolute criterion, a reasonable method is to compare the cost scores in a relative sense. An acceptable layout should be stable, which means that the layout cannot be improved by simple changes to the layout. Thus, an acceptable layout should be a local minimum of the cost function. Ideally, the optimal layout is a global minimum, where no change of the layout can reduce the cost.

It is necessary to give an operational definition of ‘minimum’ for the discrete cost function used here. For this purpose, we introduce the term, least changes. A least change on a layout is defined as moving a single node from the current position to another (vacant) position. Least changes are the simplest operations on a layout. If no least change can reduce the cost of a layout, the layout is said to be a minimum of the cost function. Obviously, such a minimum is only a local minimum, since it does not guarantee that no changes can reduce the cost score. Nonetheless, a local minimum layout is still a good candidate for an acceptable layout. Thus, we regard a layout at a local minimum of the cost function as a candidate layout. Layout optimization will be substantially improved based on candidate layouts rather than raw layouts that have much higher cost scores. The fundamental problem is how to generate candidate layouts efficiently.

### 3 ALGORITHM

When the cost function is defined, the layout problem reduces to a question of how to minimize the cost function. In terms of the criterion of the cost function, the lower the cost score, the better the layout. The goal of the optimization algorithm is to find layouts with sufficiently low costs.

#### 3.1 Grid layout algorithm

We use a simulated annealing heuristic for global optimization of the cost function. The grid layout algorithm is implemented as GridLayout\((T_{\text{max}}, T_{\text{min}}, \mathbf{R}_{\text{min}}, f_{\text{min}}, n_c, r_c, p)\). At the beginning, a random layout is chosen as the initial state. The annealing process begins with temperature \(T_{\text{max}}\). At temperature \(T\), the layout \(\mathbf{R}\) is perturbed to a neighboring layout \(\mathbf{R}'\) by subroutine Neighbor\((\mathbf{R}, p)\), where \(p\) is the perturbation rate. \(\mathbf{R}'\) is then refined to a candidate layout by the local minization subroutine LocalMin\((\mathbf{R}')\). If \(f' < f\), then the new layout is accepted; otherwise, the new layout is conditionally accepted with probability \(\exp((f - f')/T)\). These steps repeat for \(n_c\) times. Then the system cools, i.e. the temperature \(T\) is decremented by a cooling factor \(r_c\). This procedure continues until the system is frozen, i.e. \(T < T_{\text{min}}\). When the algorithm is terminated, \(\mathbf{R}_{\text{min}}\) is the lowest minimum found during the whole process and \(f_{\text{min}}\) is the corresponding cost score.

**Algorithm.** GridLayout\((T_{\text{max}}, T_{\text{min}}, \mathbf{R}_{\text{min}}, f_{\text{min}}, n_c, r_c, p)\)

1. \(T \leftarrow T_{\text{max}}\), \(\mathbf{R} \leftarrow \) a random layout
2. \(f_{\text{min}} \leftarrow f \leftarrow \text{LocalMin}(\mathbf{R})\), \(\mathbf{R}_{\text{min}} \leftarrow \mathbf{R}\)
3. \(\text{do while } T > T_{\text{min}}\)
   4. \(\text{repeat}\)
   5. \(\mathbf{R}' \leftarrow \text{Neighbor}(\mathbf{R}, p)\)
   6. \(f' \leftarrow \text{LocalMin}(\mathbf{R}')\)
   7. \(\xi \leftarrow \text{a (0,1)-random number}\)
   8. \(\text{if } \xi < \exp((f - f')/T)\) then
   9. \(f \leftarrow f', \mathbf{R} \leftarrow \mathbf{R}'\)
   10. \(\text{if } f < f_{\text{min}}\) then
   11. \(f_{\text{min}} \leftarrow f, \mathbf{R}_{\text{min}} \leftarrow \mathbf{R}\)
   12. \(\text{endif}\)
   13. \(\text{endif}\)
   14. \(T \leftarrow r_cT\)
   15. \(\text{until } n_c\) times
   16. \(\text{enddo}\)

Roughly speaking, the local minimization algorithm is used to produce the candidate layouts, and the simulated annealing procedure is used to choose better candidates.

A basic operation in the grid layout algorithm is to move the system to a ‘neighboring’ state. For a layout \(\mathbf{R}\), its neighboring layouts are generated by the following algorithm:

**Algorithm.** Neighbor\((\mathbf{R}, p)\)

1. \(\text{for each node } \alpha\)
2. \(\xi \leftarrow \text{a (0,1)-random number}\)
3. \(\text{if } \xi > p\) then
4. \(\mathbf{r}_\alpha' \leftarrow \mathbf{r}_\alpha\)
5. \(\text{else}\)
6. \(\mathbf{r}_\alpha' \leftarrow \text{a random vacant point in } L\)
7. \(\text{endif}\)
8. \(\text{endfor}\)
9. \(\text{return } \mathbf{R}' = (\mathbf{r}_1', \mathbf{r}_2', \ldots, \mathbf{r}_n')\)

In Neighbor\((\mathbf{R}, p)\), a node changes its position with a given probability \(p\), called the perturbation rate. The output layout \(\mathbf{R}'\) is similar to the input layout \(\mathbf{R}\). The similarity between \(\mathbf{R}'\) and \(\mathbf{R}\) is controlled by the perturbation rate \(p\). When \(p = 0\), \(\mathbf{R}'\) is exactly \(\mathbf{R}\); when \(p = 1\), \(\mathbf{R}'\) is totally dissimilar to \(\mathbf{R}\). In the whole procedure, the input layout \(\mathbf{R}\) remains unchanged.
3.2 An ordinary local minimization algorithm

For the sake of convenience, we introduce a least change operator $T_{ap}$ that moves a node $\alpha$ to a vacant point $p$ (a least change). Given a layout $R$, $T_{ap}R$ is a new layout that is the same as $R$ except that node $\alpha$ is at point $p$ in the new layout. We use a steepest descent method to search for local minima.

Let $L$ be the set of grid points where the nodes will be placed. A vacant point means that the point is not occupied by any node. We begin with the original algorithm LocalMin$(R)$ that implements a simple discrete steepest descent search.

Algorithm. LocalMin$(R)$

\begin{enumerate}
  \item $f_0 \leftarrow f(R)$
  \item repeat
  \item $f_{\text{min}} \leftarrow f_0$
  \item for each node $\alpha$ and vacant point $p \in L$
  \item $f_{\text{trial}} \leftarrow f(T_{ap}R)$
  \item if $f_{\text{trial}} < f_{\text{min}}$ then
  \item $f_{\text{min}} \leftarrow f_{\text{trial}}, \beta \leftarrow \alpha, q \leftarrow p$
  \item endif
  \item endfor
  \item if $f_{\text{min}} \geq f_0$ then return $f_{\text{min}}$
  \item $R \leftarrow T_{pq}R$
  \item endrepeat
\end{enumerate}

In each phase, the algorithm checks all the possible least changes for the given layout to find a best least change. There is usually more than one best least change; only the first one found is recorded. At the end of the phase, the layout is updated by applying the recorded best least change. Then the algorithm turns into the next phase. The process proceeds until the layout cannot be improved by any least change. The layout is now a local minimum of the cost function. The decrease of the cost score in each phase is the greatest possible value caused by least changes. Thus, the algorithm is a steepest decent search. Due to its high computation complexity, further improvements are required.

3.3 Fast local minimization algorithm

Denote the cost difference caused by a least change as

$$\Delta_{ap}(R) = f(T_{ap}R) - f(R).$$

(9)

All $\Delta_{ap}(R)$ constitute a matrix, called $\Delta$-matrix. A minimal element of the $\Delta$-matrix corresponds to a best least change. Algorithm LocalMin$(R)$ actually uses formula (9) to calculate the $\Delta$-matrix to find a best least change. This is extremely time-consuming, since the time complexity of formula (9) is $O(n^r)$, and $n(m-n)$ least changes need to be checked (for all $n$ nodes to move to all $m-n$ vacant points in the layout region).

Since a least change moves only one node, the major part of the cost function remains unchanged. Therefore, when a layout $R$ is altered to $T_{ap}R$ by a least change, the $\Delta$-matrix of the new layout can be updated from the old $\Delta$-matrix at a dramatically less amount of computation.

Now we deduce the update formulas for the $\Delta$-matrix. Suppose a node $\beta$ moving to a point $q$ is the best least change of layout $R$, i.e.

$$\Delta_{pq}(R) = \min(\Delta_{ap}(R), \forall \alpha \text{ and vacant } p \in L).$$

(10)

When node $\beta$ moves to point $q$, the layout changes from $R$ into $T_{pq}R$. It can be proved that for all $\alpha \neq \beta$ and $p \neq q$,

$$\Delta_{pq}(T_{pq}R) = \begin{cases} 
\Delta_{pq}(T_{pq}R) - \Delta_{pq}(R), & \text{if } \alpha = \beta; \\
\Delta_{ap}(T_{pq}R) - F_{q}(T_{pq}R) - F_{p}(T_{pq}R), & \text{otherwise};
\end{cases}$$

(11)

where TERM is an abbreviation of

$$f_{\text{opt}}(T_{pq}R) - f_{\text{opt}}(T_{pq}R) - f_{\text{opt}}(T_{pq}R) + f_{\text{opt}}(R)$$

(12)

and

$$F_{q}(R) = \sum_{j} f_{\text{opt}}(j).$$

(13)

Using formula (9) and algorithm (11), almost all elements of the new $\Delta$-matrix can be updated from the old $\Delta$-matrix at $O(1)$ computation cost. Only in $n$ special cases ($p = r_q$), should the $\Delta$ elements be calculated through formula (13), the computation cost being $O(n)$.

Based on the $\Delta$-matrix, a new local minimization algorithm, LocalMin$(R)$, which is much faster than LocalMin$(R)$ at the expense of a large extra memory demand for storing the $\Delta$-matrices, is given below:

Algorithm. LocalMin$(R)$

\begin{enumerate}
  \item $f_0 \leftarrow f(R), D_{\min} \leftarrow 0$
  \item for each node $\alpha$ and vacant point $p \in L$
  \item $D_{ap} \leftarrow F_{\alpha}(T_{ap}R) - F_{\alpha}(R)$
  \item if $D_{ap} < D_{\min}$ then $D_{\min} \leftarrow D_{ap}, \beta \leftarrow \alpha, q \leftarrow p$
  \item endfor
  \item do while $D_{\min} < 0$
  \item $D'_{\min} \leftarrow 0$
  \item for each node $\alpha \neq \beta$ and vacant point $p \in L, p \neq q$
  \item $D_{\beta p} \leftarrow \Delta_{ap}(T_{pq}R)$ using formula (11)
  \item if $D_{\beta p} < D'_{\min}$ then
  \item $D'_{\min} \leftarrow D_{\beta p}, \beta' \leftarrow \alpha, q' \leftarrow p$
  \item endif
  \item endfor
  \item $D_{\beta p} \leftarrow D'_{\beta p}$
  \item $D_{\min} \leftarrow D'_{\min}$
  \item enddo
  \item return $f_0 + D'_{\min}$
\end{enumerate}

3.4 Illustration of the grid layout algorithm

As an example, the layout process of the Escherichia coli heat shock response regulatory network is shown in Figure 1. For clarity, node
Fig. 1. A process flow of grid layout of the *E.coli* heat shock response network. (a) An initial random layout; (b) a candidate layout obtained through local minimizing the random layout a; (c) a perturbed layout of b; (d) a candidate layout obtained through local minimizing the perturbed layout c; and (e) an output layout.

labels are not shown. At the beginning, the layout is set to a random layout (Fig. 1a). By applying the local minimization algorithm LocalMin to this raw layout, a candidate layout (Fig. 1b) is obtained. Notice that many closely related nodes are placed together in the candidate layout. To find a better layout, simulated annealing is started. During the process, the candidate layout is perturbed (Fig. 1c) and a new candidate layout (Fig. 1d) is obtained after local minimization. If the new candidate fulfills the simulated annealing acceptance condition, then it replaces the old candidate and acts as the current layout. Otherwise, the current layout remains unchanged. This process repeats until simulated annealing ends. The output layout (Fig. 1e) is the best layout (whose cost score is the lowest) found during the whole search process.

3.5 Choice of parameters

When generating a candidate layout, LocalMin tests all vacant points in the layout region for each node. The computation time depends heavily on the size of the layout region. Smaller layout regions are obviously preferred for computation speed, but they limit the exploration for good layouts. Currently, there is no a priori principle to determine whether a layout region is appropriate for a given network. As an empirical rule, a rectangular layout region $[0, x_{\text{max}}] \times [0, y_{\text{max}}]$ with grid points amounting to about 3 to 6 times the number of nodes is suitable for most networks. In this work, we use $x_{\text{max}} = y_{\text{max}} = 2\sqrt{n}$. The layout perturbation rate is set to be 0.55, that is to say, for each perturbation of a layout, 55% of the nodes change their positions. Of course, this is an empirical choice.

Fig. 2. A grid layout of the yeast cell cycle regulatory network visualized by CADLIVE. The rounded rectangles are proteins, the rectangles metabolites, the parallelograms mRNAs and the ovals represent events. The solid black circles are complexes or modified molecules. The arrows indicate various reactions (binding, modification, conversion, translation and transportation) or regulations (by an enzyme, activator or inhibitor). Functional modules are marked by color blocks.

labels are not shown. At the beginning, the layout is set to a random layout (Fig. 1a). By applying the local minimization algorithm LocalMin to this raw layout, a candidate layout (Fig. 1b) is obtained. Notice that many closely related nodes are placed together in the candidate layout. To find a better layout, simulated annealing is started. During the process, the candidate layout is perturbed (Fig. 1c) and a new candidate layout (Fig. 1d) is obtained after local minimization. If the new candidate fulfills the simulated annealing acceptance condition, then it replaces the old candidate and acts as the current layout. Otherwise, the current layout remains unchanged. This process repeats until simulated annealing ends. The output layout (Fig. 1e) is the best layout (whose cost score is the lowest) found during the whole search process.

3.5 Choice of parameters

When generating a candidate layout, LocalMin tests all vacant points in the layout region for each node. The computation time depends heavily on the size of the layout region. Smaller layout regions are obviously preferred for computation speed, but they limit the exploration for good layouts. Currently, there is no a priori principle to determine whether a layout region is appropriate for a given network. As an empirical rule, a rectangular layout region $[0, x_{\text{max}}] \times [0, y_{\text{max}}]$ with grid points amounting to about 3 to 6 times the number of nodes is suitable for most networks. In this work, we use $x_{\text{max}} = y_{\text{max}} = 2\sqrt{n}$. The layout perturbation rate is set to be 0.55, that is to say, for each perturbation of a layout, 55% of the nodes change their positions. Of course, this is an empirical choice.

Fig. 2. A grid layout of the yeast cell cycle regulatory network visualized by CADLIVE. The rounded rectangles are proteins, the rectangles metabolites, the parallelograms mRNAs and the ovals represent events. The solid black circles are complexes or modified molecules. The arrows indicate various reactions (binding, modification, conversion, translation and transportation) or regulations (by an enzyme, activator or inhibitor). Functional modules are marked by color blocks.

The saturated repulsive distance, $d_{\text{max}}$, plays the role of avoiding over-separation of weakly related nodes. Generally, it is appropriate for $d_{\text{max}}$ to take a value near the average distance between clusters. In this work, $d_{\text{max}}$ is chosen to be 5, which is suitable for the networks that we have tested.

Parameters of simulated annealing are chosen as follows: The initial and terminating temperatures $T_{\text{min}} = 0.1$ and $T_{\text{max}} = (f(10)_{\text{max}} - f(10)_{\text{min}})/2$, where $f(10)_{\text{max}}$ and $f(10)_{\text{min}}$ are the maximum and minimum cost scores from 10 random layouts. The number of repetitive trials, $n_e$, at the same temperature is 10. Similarly the cooling rate $r_c = 0.8$.

3.6 Implementation

We developed Windows programs to implement the grid layout algorithm for graphs. To make grid layouts for CADLIVE networks, an auxiliary program was built to convert CADLIVE networks into graphs and communicate data between the grid layout program and CADLIVE GUI. It should be noticed that a CADLIVE network cannot be fully described by a usual graph because of the existence of modifiers of reactions. Thus, conventions are required to convert CADLIVE networks into graphs. Please refer to http://kurata21.bse.kyutech.ac.jp/grid/grid_layout.htm for details.

4 RESULTS AND DISCUSSION

4.1 Grid layout of the yeast cell cycle regulatory network

To demonstrate the feasibility of the grid layout algorithm, we applied it to the yeast cell cycle (Kurata *et al.*, 2003), a complicated regulatory network with 200 nodes. The result is shown using CADLIVE software in Figure 2. In the figure, the topological structure is shown...
clearly in a rather compact layout area. Because nodes in the same clusters are placed closely and arranged reasonably, the number of edge crossovers is quite low, although there is no special penalty term for edge crossovers.

Figure 2 shows the obvious cluster structure of the network. When we mark the functional modules of the network using color blocks, the biological meanings of the clusters are clearly shown. They correspond to the nine functional modules of the cell cycle network: global regulation (1), SCF (2), Sic1 synthesis (3), DNA replication (4), DNA checkpoint (5), spindle formation (6), spindle checkpoint (7), APC complex (8) and chromatin separation (9).

4.2 Continuous force-directed layout of the yeast cell cycle regulatory network

For comparison, a continuous force-directed layout for the same network is shown in Figure 3. The layout is computed by GraphViz, an open source software developed at AT&T Labs (http://www.graphviz.org). The nodes in the continuous force-directed layout distribute rather non-uniformly; some nodes locate too closely. In Figure 3, the graphical symbols of many nodes overlap each other. Here, we use an area smaller than needed to fit the layout to an appropriate size.

Figure 3 shows the nine modules marked by color blocks. It can be seen that most of the clusters are roughly properly positioned, but the cluster structures are not clearly exhibited. The modules, global regulation (1), SCF (2), Sic1 synthesis (3) and APC complex (8) in particular were not separated as clearly as when the network was drawn using the grid layout algorithm. Strong influences between different clusters interfere with the formation of correct cluster structures. Because of the dominant repulsive interactions, most nodes are ‘blown’ outward and the layout exhibits a dispersed radial structure.

For drawing complex biochemical networks, it is very important to exhibit clearly the cluster structures. A clearly visible cluster structure has two features: (1) the nodes within the cluster are placed together in a relatively small region; and (2) the whole cluster is far from other clusters. To measure the visual clarity of two node groups in a layout, we define the separateness of two predefined clusters A and B as

\[ S_{AB} = \frac{d_{AB}}{(r_A + r_B)}, \]

where \( d_{AB} \) is the distance between the geometric centers of the two clusters, and \( r_A \) and \( r_B \) the distribution radii. The distribution radius of a cluster is defined as the average distance between the nodes within the cluster. A large \( S_{AB} \) implies the two clusters locate far from each other; thus they are clearly visible. Contrarily, \( S_{AB} < 1 \) means the two clusters locate too closely to distinguish.

To assess the overall visibility of the clusters, we calculated the average separateness on all pairs of the nine clusters. For the grid layout shown in Figure 2, the average separateness is 3.6, while the value for layout shown in Figure 3 is 1.3. Obviously, the grid layout shows the clusters much more clearly.

In the layout shown in Figure 3, the shortest distance between nodes are so small that the layout cannot be drawn in a compact manner, because each node needs a certain space to accommodate its graphical symbol. We see in Figure 3 that there are many overlaps of node symbols. To totally avoid the overlaps, a much larger area is needed. If the graphical symbols are drawn as large as in Figure 2, the required region will be at least five times larger than in Figure 2 in area.

4.3 Comparison with continuous force-directed layout algorithms

Our layout algorithm is basically a discrete force-directed layout algorithm. The discrete and continuous algorithms are similar in the aspect that the layouts are determined through minimizing cost functions consisting of pair-wise node interactions. However, the common feature is limited to this conceptual level. Three essential differences make the grid layout algorithm distinct from continuous force-directed layout algorithms.

The first difference involves the cost functions. In continuous force-directed layouts, all nodes repulse each other and the only attractive forces are spring forces between connected nodes. Driven by the dominant repulsive forces, most nodes tend to be far from each other. Therefore, clusters are hard to form. In our cost functions, the interactions between nodes are determined by the topological structures rather than the direct connectivity. Nodes within the same clusters attract each other, while nodes belonging to different clusters repulse each other. Thus, our cost functions are preferable for displaying the topological structures clearly. In the grid layout algorithm, we introduced the saturated repulsive distance \( d_{\text{max}} \) to decrease the influence among different clusters. The \( d_{\text{max}} \) prevents repulsive node pairs from separating too far, which is in favor of creating compact layouts.

The second difference concerns the optimization methods. Efficient minimization methods are crucial for producing satisfactory force-directed layouts. A number of standard algorithms are available for continuous optimization, but optimizing efficiently a discrete cost function poses a serious challenge. There is no general algorithm applicable to discrete functions. We propose a fast discrete local minimization method, which quickly generates candidate layouts and thus makes it possible to choose the best layout from many candidates. Notice that discrete ‘local search’ means for single node to
find the best change each time, but the operational region for each node is the whole layout region. In this aspect, our discrete method is different from continuous local minimization methods, where ‘local search’ means for all nodes to find the best nearby positions.

The third difference relates to the geometrical distribution of nodes. In grid layouts, nodes’ distribution is relatively uniform and the minimum distance between nodes is determined. This is an advantage for complex networks because the node size needs taking into account. By contrast, in continuous force-directed layouts, most nodes scatter as far as possible but the minimum distances between neighboring nodes are often too small compared with the size of layout area. In cases where some nodes are located too close, the entire layout area must be magnified to avoid overlapping of the graphical node symbols.

4.4 Limitations

4.4.1 Metabolic pathways The design principle behind the grid layout algorithm is applicable to general networks. Metabolic pathways as networks can, of course, be drawn by our algorithm (Supplementary materials). In practice, a metabolic pathway usually represents a small part of a complex metabolic circuit; but it has special requirements on its layouts to meet the established drawing styles:

1. Node sizes should be considered to accommodate node labels that are often very complex (such as chemical structures) and need large and variable spaces to display.
2. Edges should be carefully routed to highlight the main paths.
3. Enzyme names should be properly displayed as edge labels, some of which may be very long.

However, for large complex networks, it is more important to show their cluster structures in layouts. Our algorithm aims at making clear topological structures of biochemical networks; thus we provide the same node size to each species (enzymes, metabolite or genes). It does not deal with the issues specific to metabolic pathways.

4.4.2 Computation speed For networks with several tens of nodes, it needs a few tens of seconds to compute a grid layout. The computation time increases rapidly as network size grows. For the 200-node yeast cell cycle network, it took ~10 min to produce a layout on a Pentium IV 1.7 GHz personal computer. Biochemical networks with a few hundred nodes are in the scope of the current grid layout algorithm. We are developing further acceleration techniques to allow the use of grid layout algorithm for very large networks.

4.4.3 Memory requirement The memory needed by the grid layout algorithm is $O(nm)$ ($m =$ the number of grid points in the layout region), which is required for storing the $\Delta$-matrices. This is not a serious limitation for most up-to-date desktop computers.

5 CONCLUSION

Two essential issues need to be addressed to find acceptable grid layouts: an elaborately defined cost function and an efficient discrete minimization method. The proposed cost function together with the optimization algorithm has demonstrated the ability to make satisfactory layouts of biochemical networks with the following advantages: (1) to produce layouts clearly exhibiting the clusters which are often functional modules and (2) to produce compact layouts with nodes distributed uniformly.

ACKNOWLEDGEMENTS

We thank Kanako Sumi and Natsumi Shimizu for their help in the data preparation. This work is supported by the project for development of a technological infrastructure for industrial bioprocesses on R&D of New Energy and Industrial Technology Development Organization (NEDO).

REFERENCES