BIOINFORMATICS ORIGINAL PAPER Vol. 24 no. 12 2008, pages 1433–1441
doi:10.1093/bioinformatics/btn196

Systems biology

Fast grid layout algorithm for biological networks with sweep calculation

Kaname Kojima, Masao Nagasaki* and Satoru Miyano

Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan

Received on January 17, 2008; revised on March 7, 2008; accepted on April 16, 2008

Advance Access publication April 18, 2008

Associate Editor: Jonathan Wren

ABSTRACT

Motivation: Properly drawn biological networks are of great help in the comprehension of their characteristics. The quality of the layouts for retrieved biological networks is critical for pathway databases. However, since it is unrealistic to manually draw biological networks for every retrieval, automatic drawing algorithms are essential. Grid layout algorithms handle various biological properties such as aligning vertices having the same attributes and complicated positional constraints according to their subcellular localizations; thus, they succeed in providing biologically comprehensible layouts. However, existing grid layout algorithms are not suitable for real-time drawing, which is one of requisites for applications to pathway databases, due to their high-computational cost. In addition, they do not consider edge directions and their resulting layouts lack traceability for biochemical reactions and gene regulations, which are the most important features in biological networks.

Results: We devise a new calculation method termed sweep calculation and reduce the time complexity of the current grid layout algorithms through its encoding and decoding processes. We conduct practical experiments by using 95 pathway models of various sizes from TRANSPATH and show that our new grid layout algorithm is much faster than existing grid layout algorithms. For the cost function, we introduce a new component that penalizes undesirable edge directions to avoid the lack of traceability in pathways due to the differences in direction between in-edges and out-edges of each vertex.

Availability: Java implementations of our layout algorithms are available in Cell Illustrator.

Contact: masao@ims.u-tokyo.ac.jp

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 INTRODUCTION

For biological pathways such as signal transduction pathways, gene regulatory networks and metabolic pathways, one of the crucial techniques for understanding their characteristics is to use graph representation. Both publicly available (Kanehisa, 2002) and commercial pathway databases (Schacherer *et al.* 2001) display retrieved pathways in the form of graphs to facilitate the users' comprehension of them. Considering the large numbers of pathways retrieved with various types of criterion according to biologists' purposes and the tediousness of manually drawing graphs, automatic layout algorithms are strongly required.

There exist rigorous studies in standard graph layout algorithms including circular, orthogonal or planar drawing, and force-directed heuristics (Battista *et al.*, 1994; Battista *et al.*, 1999; Brandenburg *et al.*, 1997). However, none of these algorithms give satisfactory results when applied to pathway networks due to insufficient use of biological properties such as structural characteristics and subcellular localization information (Becker and Rojas, 2001; Li and Kurata, 2005; Wegner and Kummer, 2005).

Thus far, several types of drawing algorithms have been designed for biological pathways and they have been integrated in biological simulation software (Demir *et al.*, 2002; Dogrusoz *et al.*, 2006; Doi *et al.*, 2003; Kurata *et al.*, 2003, 2005; Nagasaki *et al.*, 2003; Shannon *et al.*, 2003).

Karp and Paley (1994) proposed the extraction of biological topologies such as linear, cyclic and branching pathways and their utilization as the backbone of the layout. For chemical reaction networks, the method described by Becker and Rojas (2001) uses the longest directed cycle drawn as the backbone of the layout to capture the flow of reactions. On the other hand, as small cycles are known to participate in important recycling processes, Wegner and Kummer (2005) proposed the use of recursively extracted small cycles as the backbone of the layout and their work has been integrated in Systems Biology Markup Language (SBML) with layout extensions (Gauges *et al.*, 2006).

Several biological properties are considered in force-directed approaches. The use of edge directions and simple positional constraints (Dogrusoz *et al.*, 2004; Genc and Dogrusoz, 2003) have been proposed for more general metabolic pathways. In GOlorize (Garcia *et al.*, 2007), the extra attractive force is applied to vertices belonging to the same Gene Ontology class. An SBML layout extension, SBWAutoLayout (Deckard *et al.*, 2006), employs the force-directed approach as its layout algorithm. However, force-directed approaches are not suitable for generating compact layouts of complex pathways (Li and Kurata, 2005). In addition, force-directed approaches have a difficulty in handling complicated positional constraints such as arranging some vertices only on a tours-shaped region.

*To whom correspondence should be addressed.

Grid layout algorithms were originally proposed by Li and Kurata (2005), in which vertices composing the graph are mapped to grid points, and are arranged to minimize a cost function defined over all possible mappings. Cost functions comprise several components: vertex distances weighed according to the graph structure (Li and Kurata, 2005), edge–edge and vertex–edge crossings (Kato *et al*., 2005), and biologically preferable criterion such as aligning vertices having the same attribute (Kojima *et al*., 2007). Since even finding the layout with the minimum edge–edge crossings is NP-hard (Gary and Johnson, 1983) the basic grid layout algorithm repeatedly updates the layout by moving vertices one by one under a greedy search strategy, and a locally optimal layout is obtained after convergence, while its variants consider vertex position swappings (Kojima *et al*., 2007) or restrict the search space to stochastically chosen adjacents (Barsky *et al*., 2007). In addition, grid layout algorithms can deal with complicated positional constraints and thus they succeed in generating compact and biologically comprehensible layouts.

In this study, we propose a new grid layout algorithm intended for applications of biological pathway databases. We adopt the approach proposed by Kato *et al*. (2005), termed CB-grid layout, as the baseline of our algorithm because CB-grid layout first considers the edge–edge and vertex–edge crossings, which strongly influence the understandability of layouts. The CB-grid layout is, however, unsuitable for applications to pathway databases in the following reasons. (i) Although at each update step, all the differences in the cost between a current layout and its candidate adjacents are calculated efficiently using the calculation results from the previous step, current grid layout algorithms still require a high-computational time. In addition, the initial step is calculated inefficiently and has a large computational overhead, due to the unavailability of previous calculation results. Thus, current layout algorithms cannot be used for real-time drawing. (ii) Since the extant grid layout algorithms do not consider the edge directions, the resulting layouts lack traceability for biochemical reactions and gene regulations, which are very important characteristics of biological pathways.

In order to address the first problem, we propose a new calculation method for cost differences termed *sweep calculation*. In sweep calculation, costs changed by moving a vertex of interest are encoded, and then the cost differences corresponding to the movements are calculated by using the encoded data. Since both the encoding and decoding processes require less-time complexities than that of the method used in CB-grid layout, sweep calculation succeed in reducing the computational time of CB-grid layout, in particular that of the initial step.

For the second problem, we consider that the lack of traceability is caused by the differences in direction between in-edges and out-edges of each vertex. We introduce a new component to the cost function that employs the negative-inner product between the in-edges and out-edges on each vertex as the cost to penalize undesirable directions.

The remainder of this article is organized as follows. In Section 2, after presenting details of the basic grid layout algorithm, we describe sweep calculation and how to use this method for the calculation of the initial step. We also compare the time complexities of the CB-grid layout and our new grid layout algorithm. Section 3 introduces a new component of the cost function, the flow penalizing cost, and describes its efficient calculation. In Section 4, the performance of our approach is evaluated by using various pathway models from a biological pathway database. Section 5 summarizes our study.

2 METHODS

2.1 Grid layout and extant search strategy

Given a graph $G = (V, E)$ with vertices V and edges E and a grid of h rows and w columns, a *layout* $L = (V, E, U, P)$ of G comprises the underlying graph G, $w \cdot h$ grid points U, and a function $P : V \to U$ such that $P(v_\alpha) \neq P(v_\beta)$ for any two distinct vertices $v_\alpha, v_\beta \in V$. Hereafter, we denote the degree of vertex v as $\deg(v)$, the cardinality of a set as $|\cdot|$, a set of edges connected to vertex v as E_v, a set of vertices adjacent to vertex v as V_v and an edge between vertices v and v' as $e(v, v')$. We define the following functions:

- $\mathrm{Cross}_{e_i, e_j}(L)$: a binary function that returns 1 if an edge e_i crosses an edge e_j and 0 otherwise.
- $\mathrm{Cross}_{v_i, e_j}(L)$: a binary function that returns 1 if an edge e_j crosses a vertex v_i and 0 otherwise.
- $\mathrm{Distance}_{v_i, v_j}(L)$: a function that returns

$$\begin{cases} w_{v_i, v_j} \cdot md(v_i, v_j) & \text{if } md(v_i, v_j) \leq sd \wedge w_{v_i, v_j} < 0 \\ w_{v_i, v_j} \cdot sd & \text{otherwise} \end{cases}, \tag{1}$$

where w_{v_i, v_j} is the weight of a pair of vertices v_i and v_j according to the graph structure; $md(v_i, v_j)$, the Manhattan distance between v_i and v_j; and sd, the saturation distance. For details see Li and Kurata (2005).

By using the above functions, the *layout cost* $C(L)$ of L is defined as follows:

$$C(L) = W_{ee} \sum_{e_i, e_j \in E} \mathrm{Cross}_{e_i, e_j}(L) + W_{ne} \sum_{v_k \in V, e_l \in E} \mathrm{Cross}_{v_k, e_l}(L) + W_{dc} \sum_{v_m, v_n \in V} \mathrm{Distance}_{v_m, v_n}(L), \tag{2}$$

where W_{ee}, W_{ne} and W_{dc} are constant values used to adjust the effect of each component.

At each step, the algorithm calculates the costs of all layouts that can be generated by moving one of all vertices to one of all vacant points. The layout with the minimum cost is selected as a starting layout for the next step. After convergence, the algorithm outputs a locally optimal layout. Since inefficiently calculating all possible adjacent layouts requires a high-time complexity, the CB-grid layout algorithm uses a Δ matrix, whose entry $\Delta_{v,p}$ stores the cost difference by moving vertex v to a vacant point p at the previous step, and reduces the time complexity at each step from $O(w \cdot h \cdot (|V|^2 + |E|^2))$ to $O(|V|^2 + |E|^2 + w \cdot h \cdot |E_{v_\beta}|(|V| + |E|))$, where v_β is the vertex moved at the previous step.

In addition, when vertices and grid points in the model hold subcellular-localization information as positional constraints, CB-grid layout limits vertices to be mapped to the grid points that satisfy their subcellular localizations.

In the following subsections, we introduce a new calculation method termed sweep calculation, which can efficiently calculate the Δ matrix without using the previous calculation results, and describe how to reduce the time complexity of the CB-grid layout.

2.2 Sweep calculation

As mentioned in the previous subsection, at the initial step, the Δ matrix is calculated inefficiently and $O(w \cdot h \cdot (|V|^2 + |E|^2))$ time is required due to the unavailability of the previous calculation results.

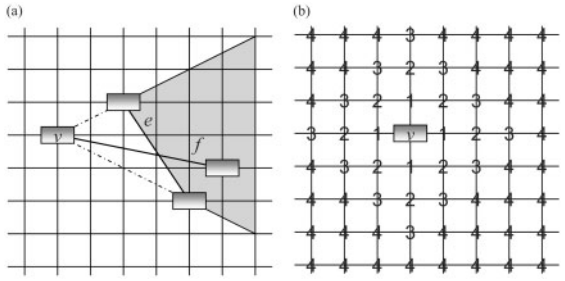

Fig. 1. (**a**) Edge e and vertex v are lying on the grid. A crossing occurs only when another end point of f opposite to v lies in the shaded region. The shaded region always forms a convex set. (**b**) An image of a distance pattern on the grid. The digit on each grid point denotes the distance from a vertex v, which is defined in Equation (1).

Sweep calculation can calculate the same operation in $O(\min(w,h)\cdot(|V|^2+|E|^2)+w\cdot h|V|)$ time, i.e. $\max(w,h)$ times faster than the inefficient method, without increasing the space complexity. Below we show the principle of sweep calculation through Lemmas 1–5, and prove the time complexity of the above case in Proposition 1.

Assume that an edge e and a vertex v are lying on the grid, as shown in Figure 1a. We also assume that there exists an edge f connected to the vertex v. Edge f has a crossing with edge e only when the other end point of edge f is in the shaded region $R_{e,v}$ shown in Figure 1a. We call this shaded region $R_{e,v}$ the *crossing region*.

Except for the case where v is on the extension of e, the crossing region is surrounded by edge e and two half-lines from the end points of e to the counterside of vertex v, as shown in Figure 1a. Since the size of the grid is finite in practice, the crossing region is also surrounded by the boundary of the grid. We define grid points with the same x-coordinate as *vertical grid point set*, i.e. $(x,0),\ldots,(x,h-1)$. The crossing region has a convex form, as shown in Figure 1a, and thus the intersection of the vertical grid point set and the crossing region can be represented as $(x,y_1),\ldots,(x,y_2)$. We call the two bookending points (x,y_1) and (x,y_2) the *upper crossing boundary point* and *lower crossing boundary point*, respectively, and the set of these crossing boundary points for all x *crossing boundary*. We denote the crossing boundary induced by edge e and the edge connected to vertex v as $B_{e,v}$. Note that some vertical grid point sets do not intersect the crossing region and hence have no crossing boundary point.

Each crossing boundary point is obtained in constant time because a crossing region is surrounded by at most six lines, and for each line, the y-coordinate value corresponding to a specific x-coordinate value is calculated in constant time. Without loss of generality, we assume that $w \le h$, and by following the above fact, we have the following lemma.

Lemma 1. *Crossing boundary $B_{e,v}$ can be obtained in $O(\min(w,h))$ time.*

We introduce the following two functions associated with the crossing region $R_{e,v}$ and crossing boundary $B_{e,v}$:

- $r_{e,v}(x,y)$: a binary function $[0,w-1]\times[0,h-1]\mapsto\{0,1\}$ that returns 1 if (x,y) is in the crossing region $R_{e,v}$ and 0 otherwise.
- $b_{e,v}(x,y)$: a function $[0,w-1]\times[0,h-1]\mapsto\{-1,0,1\}$ that returns

$$\begin{cases} 1 & \text{if } (x,y) \text{ is a upper crossing boundary point in } B_{e,v} \\ -1 & \text{if } (x,y-1) \text{ is lower crossing boundary point in } B_{e,v}. \\ 0 & \text{otherwise} \end{cases}$$

Note that $r_{e,v}(x,y)$ and $b_{e,v}(x,y)$ represent the same things as crossing region $R_{e,v}$ and crossing boundary $B_{e,v}$, respectively. We implement these two functions as an $h\times w$ matrix.

Lemma 2. *$r_{e,v}$ can be obtained from $b_{e,v}$ in $O(w\cdot h)$ time.*

Proof. Let $s(x,y)=\sum_{j=0}^{y}B_{e,v}(x,j)$. Since $r_{e,v}(x,y)=s(x,y)$ and $s(x,y)=s(x,y-1)+B_{e,v}(x,y)$, it is evidenced that $r_{e,v}(x,0),\ldots,r_{e,v}(x,h-1)$ can be calculated in $O(h)$ time. Thus, $R_{e,v}$ is obtained from $B_{e,v}$ in $O(w\cdot h)$ time.

Lemma 3. *A function that returns the same values as $\sum_{i=1}^{n}\alpha_i r_{e_i,v_i}$ can be obtained in $O(w\cdot h+n\min(w,h))$ time.*

Proof. From the proof of Lemma 2, $\sum_{i=1}^{n}\alpha_i r_{e_i,v_i}(x,y)=\sum_{i=1}^{n}\alpha_i\sum_{j=0}^{y}b_{e_i,v_i}(x,j)=\sum_{j=0}^{y}\sum_{i=1}^{n}\alpha_i b_{e_i,v}(x,y')$. Instead of preparing b_{e_i,v_i}, by preparing a $h\times w$ matrix whose every entry is 0, and adding non-trivial values to it, which are supposed to be used for each b_{e_i,v_i}, we obtain a function that returns the same value as $\sum_{i=1}^{n}\alpha_i b_{e_i,v}$ in $O(w\cdot h)$ time. Thus, by following Lemma 2 as well, $\sum_{i=1}^{n}\alpha_i r_{e_i,v}$ can be obtained in $O(w\cdot h+n\cdot\min(w,h))$ time.

Lemma 4. *Difference of the number of edge–edge crossings induced by moving a vertex v from (x,y) to (x',y') can be calculated by*

$$\sum_{v'\in V_v}\sum_{e\in E\setminus E_v}(r_{e,v'}(x',y')-r_{e,v'}(x,y)).$$

The calculation of all the differences induced by the movement of v requires $O(w\cdot h+\min(w,h)\cdot\deg(v))$ time.

Proof. The former part of this lemma is obvious. We prove the latter part. Since $\sum_{v'\in V_v}\sum_{e\in E\setminus E_v}r_{e,v'}$ can be prepared in $O(\min(w,h)\deg(v))$ time from Lemma 3 and each movement can be calculated in $O(1)$ time, $O(w\cdot h+\min(w,h)\deg(v))$ time is required in total.

We can consider the case of vertex-edge crossings in a similar manner because in our setting, the regions of vertices are denoted by a rectangle, and thus we can detect the vertex–edge crossing by checking the edge–edge crossings between the edge of interest and lines surrounding the rectangular region of the vertex of interest. Assume that vertex v and edge e are lying on the grid and e is connected to vertex v'. By moving v' around, a crossing between v and e occurs when v' is in some region. On the other hand, by moving v around, a crossing between v and e occurs when v is in some different region. As in the case of edge–edge crossings, we call these regions crossing regions and denote the region in the former case as $R^{ev}_{v,v',e}$ and the region in the latter case as $R^{ve}_{v,e}$. Further, we denote the crossing boundary for $R^{ev}_{v,v',e}$ as $B^{ev}_{v,v',e}$ and that for $R^{ve}_{v,e}$ as $B^{ve}_{v,e}$.

Unfortunately, we cannot directly use the same strategy for the distance cost because the value of each grid point for the distance from a vertex v changes gradually with the distance from v until the distance attains the saturation distance, as shown in Figure 1b. Beyond the region bounded by the saturation distance, values corresponding to grid points are set to the same value or the saturation distance. Thus, we focus on the region bounded by the saturation distance, which we call the *distance region*, and introduce the new strategy. For simplicity, we consider that the distance region is completely contained in the grid. It is evidenced that the distance region is surrounded by four line segments induced by the saturation distance. We call these line segments the *distance boundary* and denote them as B^d_v. We define the upper and lower distance boundaries as in the case of crossings. We introduce the following functions, which are also implemented as $h\times w$ matrices:

- $dr_v(x,y)$: a function $[0,w-1]\times[0,h-1]\mapsto Z$ that returns the distance between (x,y) and vertex v defined in Equation (1).
- $db_v(x,y)$: a function $[0,w-1]\times[0,h-1]\mapsto Z$ that returns

$$\begin{cases} -1 & \text{if } (x,y) \text{ is on the upper distance boundary} \\ -1 & \text{if } (x,y-1) \text{ is on lower distance boundary} \\ 2 & \text{if } v_x - sd < x < v_x \text{ and } y = v_y \\ 0 & \text{otherwise} \end{cases}$$

```
initialization(v ∈ V, Δ)
1  d_min ← 0
2  for each vertex v_α ∈ V
3     reset(R, 0)
4     setEdgeEdge(v_α, E_{v_α}, E, W_ee)
5     setEdgeVertex(v_α, F_{v_α}, V, W_ve)
6     setVertexEdge(v_α, V, W_ve)
7     setDistance(v_α, W_dc)
8     for each p ∈ U
9        d_tmp ← r(p) − r(P(v_α))
10       Δ_{v_α,p} ← d_tmp
11       if p is vacant
12          if d_tmp < d_min
13             v_min ← v, q ← p, d_min ← d_tmp
14          end if
15       end if
16    end for
17 end for
18 return v_min, q, d_min
```

```
setEdgeVertex(v ∈ V, E'_v, V', W)
1  reset(B, 0)
2  for each edge e_v ∈ E'_v
3     for each vertex v' ∈ V' \ {v}
4        for i ← 1 to w
5           setEdgeVertexBoundary(v', e_v, v, i)
6        end for
7     end for
8  end for
9  for i ← 1 to w
10    s ← 0
11    for j ← 1 to h
12       s ← s + b(i, j)
13       r(i, j) ← r(i, j) + W · s
14    end for
15 end for
```

```
setEdgeEdge(v ∈ V, E'_v, E', W)
1  reset(B, 0)
2  for each edge e_v ∈ E'_v
3     for each edge e ∈ E' \ E_v
4        for i ← 1 to w
5           setEdgeEdgeBoundary(e, v, i)
6        end for
7     end for
8  end for
9  for i ← 1 to w
10    s ← 0
11    for j ← 1 to h
12       s ← s + b(i, j)
13       r(i, j) ← r(i, j) + W · s
14    end for
15 end for
```

```
setVertexEdge(v ∈ V, E', W)
1  reset(B, 0)
2  for each edge e ∈ E' \ E_v
3     for i ← 1 to w
4        setVertexEdgeBoundary(v, e, i)
5     end for
6  end for
7  for i ← 1 to w
8     s ← 0
9     for j ← 1 to h
10       s ← s + b(i, j)
11       r(i, j) ← r(i, j) + W · s
12    end for
13 end for
```

```
setDistance(v ∈ V, V', W)
1  reset(B, sd)
2  for each vertex v' ∈ V' \ v
3     for i ← 0 to w − 1
4        setDistanceBoundary(v, w_{v,v'}, i)
5     end for
6  end for
7  for i ← 1 to w
8     s ← 0, t ← 0
9     for j ← 1 to h
10       s ← s + b(i, j), t ← t + s
11       r(i, j) ← r(i, j) + W · t
12    end for
13 end for
```

Fig. 2. Pseudo-codes of the functions of sweep calculation and the initial step of the CB-grid layout using sweep calculation.

where v_x and v_y are the x and y-coordinate values of the grid point to which v_x is mapped, respectively.

The following lemma shows that dr_v can be decoded from db_v.

LEMMA 5. *dr_v can be generated from db_v. This operation requires $O(w \cdot h)$ time.*

PROOF. All $dr_v(x, y)$ are set to sd, which requires $O(w \cdot h)$ time. Let $s(x, y) = \sum_{i=0}^{y} db(x, i)$ and $t(x, y) = \sum_{j=0}^{y} s(x, j)$. When $v_y - sd < y < v_y$, the distance from v is decremented as y is incremented. On the other hand, when $v_y < y < v_y + sd$, the distance from v is incremented as y is incremented. Since $s(x, y)$ returns the distance between v and (x, y) subtracted from the distance between v and $(x, y-1)$, which satisfies this law, $dr_v(x, y) = \sum_{i=0}^{y} t(x, i)$. Since $s(x, y) = s(x, y-1) + db_v(x, y)$, $t(x, y) = t(x, y-1) + s(x, y)$, and $dr_v(x, y) = dr_v(x, y-1) + t(x, y)$, dr_v can be generated from db_v in $O(w \cdot h)$ time.

By using the facts from Lemmas 1–5, we present sweep calculation for the following cases and function initialization, which computes the initial step of the CB-grid layout using sweep calculation, as shown in Figure 2 with pseudo-codes.

- setEdgeEdge(v, E'_v, V', W_{ee}): a function that calculates the costs from the sum of the edge–edge crossings between $e_v \in E'_v \subset E_v$ and $e \in e'$ for all the mappings of v to a point.
- setEdgeVertex(v, E'_v, V', W_{ve}): a function that calculates the costs from the sum of the vertex–edge crossings between $e_v \in E'_v \subset E_v$ and $v' \in V'$ for all the mappings of v to a point.
- setVertexEdge(v, E', W_{ve}): a function that calculates the costs from the sum of the vertex–edge crossings between v and $e' \in E'$ for all the mappings of v to a point.
- setDistance(v, V'): a function that calculates the sum of the distance costs between v and $v' \in V'$ for all the mappings of v to a point.

In the pseudo-codes, the following matrices and functions are newly introduced:

- R: A matrix of h rows and w columns. $r(j, i)$ denotes the ith row and jth column of R and corresponds to a grid point of the ith row and jth column. We also use the notation $r(p)$ to denote an element of R corresponding to grid point p.
- B: A matrix of $h-1$ rows and $w-1$ columns. $b(j, i)$ denotes the ith row and jth column of B and corresponds to a grid point of the ith row and jth column. We also use the notation $b(p)$ to denote an element of B corresponding to grid point p.
- reset(M, *value*): a function that sets all values in matrix M to *value*. This function requires $O(w \cdot h)$ time.

- setEdgeEdgeBoundary(e, v, i): Let (i, y^u) be an upper boundary point and $(i, y^l - 1)$ be a lower boundary point in $B_{e,v}$. This function increments $b(i, y^u)$ and decrements $b(i, y^l)$. This function requires $O(1)$ time.
- setEdgeVertexBoundary(v, v', e, i): Let (i, y^u) be an upper boundary point and $(i, y^l - 1)$ be a lower boundary point in $B^{ev}_{v,v',e}$. Note that e must be connected to v'. This function increments $b(i, y^u)$ and decrements $b(i, y^l)$. This function requires $O(1)$ time.
- setVertexEdgeBoundary(v, e, i): Let (i, y^u) be an upper boundary point and $(i, y^l - 1)$ be a lower boundary point in $B^{ve}_{v,e}$. This function increments $b(i, y^u)$ and decrements $b(i, y^l + 1)$. This function requires $O(1)$ time.
- setDistanceBoundary($v, w_{v,v'}, i$): Let (i, y^u) be an upper boundary point and $(i, y^l - 1)$ be a lower boundary point in B^d_v. This function subtracts $w_{v,v'}$ from $b(i, y^u)$ and $b(i, y^l + 1)$, and adds $2w_{v,v'}$ to $b(i, v_y)$. This function requires $O(1)$ time.

In the initialization, for each vertex $v \in V$, all the values in R are set to 0 by reset($R, 0$) and setEdgeEdge, setEdgeVertex, setVertexEdge and setDistance are calculated. In setEdgeEdge, B is initialized by reset($B, 0$) and the crossing boundaries are recorded in B. The cost from the edge–edge crossings is then calculated from the crossing boundaries by following Lemma 3 and it is summed up to R. In setEdgeVertex and setVertexEdge, the vertex–edge crossings are calculated in a manner similar to that of setEdgeEdge. In setDistance, all values in B are set to sd by reset(R, sd) and the distance boundaries are recorded in B. The distance cost is then calculated from the crossing boundaries by following Lemma 5 and it is summed up to R. After the calculation of these four functions, the differences of the cost between the current layout and its adjacents are calculated and stored in the Δ matrix by using the values in R. During this process, the function selects the best movement of vertex $v_{\min}$ to the grid point q and finally returns the best movement and its corresponding cost difference $d_{\min}$.

The following proposition concludes the time and space complexity of the initialization.

Proposition 1. *Initialization requires* $O(\min(w,h) \cdot (|V|^2 + |E|^2) + (w \cdot h)|V|)$ *time and* $O(w \cdot h)$ *space*.

Proof. First, we focus on vertex v and show the time complexity corresponding to vertex v at each step. Since setEdgeEdgeBoundary, setEdgeVertexBoundary, setVertexEdgeBoundary and setDistance Boundary require constant time and reset requires $O(w \cdot h)$ time, setEdgeEdge, setEdgeVertex, setVertexEdge and setDistance require $O(\min(w,h) \cdot |E| \cdot \deg(v) + w \cdot h)$, $O(\min(w,h) \cdot |V| \cdot \deg(v) + w \cdot h)$, and $O(\min(w,h) \cdot |V| + w \cdot h)$ time, respectively. In addition, the calculation of all cost differences requires $O(w \cdot h)$ time, and the time complexity required for vertex v amounts to $O(\min(w,h) \cdot (|V| + |E|) \times (\deg(v) + 1) + w \cdot h)$ time.

Thus, by summing up $O(\min(w,h) \cdot (|V| + |E|) \cdot (\deg(v) + 1) + w \cdot h)$ time with respect to v, we observe that the initialization requires $O(\min(w,h)(|V|^2 + |E|^2) + (w \cdot h)|V|)$ time, where we use the fact that $\sum_{v \in V} \deg(v) = 2|E|$.

Since $O(w \cdot h)$ space is required to store the data of R and B, sweep calculation requires $O(w \cdot h)$ space.

2.3 Grid layout algorithm with sweep calculation

The previous subsection shows the efficient calculation of the initial step in the CB-grid layout using sweep calculation. At the update step, the time complexity is mainly dependent on the crossings related to a vertex of interest v and a vertex moved at the previous step v_β, which cannot be calculated by the Δ matrix:

- edge–edge crossing between $e_v \in E_v$ and $e_{v_\beta} \in E_{v_\beta}$: $O(|E_v||E_{v_\beta}|)$ time for each movement.
- vertex–edge crossing between $e_v \in E_v$ and v_β: $O(|E_v|)$ time for each movement.
- vertex–edge crossing between $e_{v_\beta} \in E_{v_\beta}$ and v: $O(|E_{v_\beta}|)$ time for each movement.
- edge–edge crossing between edge $e(v, v_\beta)$ and $E \setminus \{E_v \cup E_{v_\beta}\}$ and vertex-edge crossing between edge $e(v, v_\beta)$ and $V \setminus \{v, v_\beta\}$ if edge $e(v, v_\beta)$ exists: $O(|E|)$ and $O(|V|)$ time for each movement.

The calculations for these cases require a total time of $O(w \cdot h \cdot \deg(v_\beta)|E|)$ at each step. Sweep calculation can be applied to these calculations and it reduces the time complexity to $O(\min(w,h) \cdot \deg(v_\beta)|E|)$.

However, when the vertex is moved to the point to which v_β was mapped at the previous step, the cost difference is calculated inefficiently since the cost difference for the point was not calculated at the previous step. Thus, this case requires a total time of $O(|E|^2)$ at each step.

As is done in Kojima *et al.* (2007) by defining the cost differences corresponding to occupied points, this case can be efficiently calculated as well. This is because when two vertices are mapped to the same point, we can calculate the cost in the same manner as a layout in which no two vertices are mapped to the same point, and the cost differences corresponding to the occupied points are also defined in the same manner as those for vacant points. Note that if an edge passes sufficiently close to a grid point to which two vertices are mapped to and the edge makes crosses with both vertices, the number of crossings is counted as two.

Due to its definition, we observe that the cost differences obtained by sweep calculation include the cases of occupied points, and thus we can use the cost differences obtained by sweep calculation directly to update the cost differences for the occupied points.

By summarizing the above definition and properties, we show the new grid layout algorithm, *CBS-grid layout* (CB-grid layout with sweep calculation), with the pseudo-code in Figure 3.

The next proposition gives the time complexity of the CBS-grid layout.

Proposition 2. *At each step, the CBS-grid layout shown in Figure 3 calculates all the cost differences as in the case of the CB-grid layout in* $O(\min(w,h) \cdot \deg(v_\beta) \cdot |E| + w \cdot h \cdot |V|)$ *time*.

Proof. As discussed abov, since sweep calculation requires $O(\min(w,h) \cdot \deg(v_\beta) \cdot |E| + w \cdot h \cdot |V|)$ time and by using the data from the Δ matrix and sweep calculation, the calculation of each cost difference requires a constant time, as shown in Figure 3, the CB-grid layout with sweep calculation requires $O(\min(w,h) \cdot \deg(v_\beta) \cdot |E| + w \cdot h \cdot |V|)$ time.

2.4 Comparison of time complexities

We compare the time complexities of the CB and CBS-grid layouts. It is assumed that $|V| \leq |E|$, $w \cdot h$ is proportional to $|V|$, and all vertices are equally selected to update the layout on average. At each step, the CB-grid layout requires $O(|V|^2 + |E|^2 + w \cdot h \cdot \deg(E_{v_\beta})(|V| + |E|))$, where v_β is the vertex moved at the previous step (Kato *et al.*, 2005; Kojima *et al.*, 2007), while the CBS-grid layout requires $O(\min(w,h)|E|(|V| + |E|) + (w \cdot h)|V|)$ time.

Since from the above assumptions, $\deg(E_{v_\beta})$ equals to avedeg $\equiv |E|/|V|$ and $w \cdot h = C|V|$, where C is a constant, the time complexities of the CB and CBS-grid layouts are $O(\text{avedeg}^2 \cdot |V|^2 \cdot C)$ and $O(\text{avedeg}^2 \cdot |V|^{3/2} C^{1/2} + C|V|^2)$, respectively. Thus, for the case where avedeg $\leq (C \cdot |V|)^{1/4}$, the second term of the time complexity of the CBS-grid layout dominates and the CBS-grid layout is avedeg^2 times faster than the CB-grid layout, and $\sqrt{C|V|}$ times faster otherwise.

```
CBS-grid layout
(v_β, q, d_min) ← initialization(V, Δ)
while d_min < 0
  r_β ← P(v_β)
  move v_min to q
  d_min ← 0
  for each vertex v_α ∈ V
    reset(R, 0)
    setEdgeEdge(v_α, E_{v_α}\e(v_α, v_β), E_{v_β}\e(v_α, v_β), W_ee)
    setEdgeVertex(v_α, E_{v_α}\e(v_α, v_β), {v_β}, −W_ve)
    setVertexEdge(v_α, E_{v_β}\e(v_α, v_β), −W_ve)
    setEdgeEdge(v_α, {e(v_α, v_β)}, E, −W_ee)
    setEdgeVertex(v_α, {e(v_α, v_β)}, V, −W_ve)
    setDistance(v_α, −W_dc)
    move v_β to p
    setEdgeEdge(v_α, E_{v_α}\e(v_α, v_β), E_{v_β}\e(v_α, v_β), −W_ee)
    setEdgeVertex(v_α, E_{v_α}\e(v_α, v_β), {v_β}, W_ve)
    setVertexEdge(v_α, E_{v_β}\e(v_α, v_β), W_ve)
    setEdgeEdge(v_α, {e(v_α, v_β)}, E, W_ee)
    setEdgeVertex(v_α, {e(v_α, v_β)}, V, W_ve)
    setDistance(v_α, W_dc)
    move v_β to r_β
    for each p ∈ U
      d_tmp ← Δ_{v_α,p} − Δ_{v_α,P(v_α)} + r(p) − r(P(v_α))
      Δ'_{v_α,p} ← d_tmp
      if p is vacant
        if d_tmp < d_min
          v_min ← v, q_tmp ← p, d_min ← d_tmp
        end if
      end if
    end for
  end for
  v_β ← v_min, q ← q_tmp, Δ ← Δ'
end while
```

Fig. 3. Pseudo-code of CBS-grid layout.

Since the CB-grid layout calculates the cost differences inefficiently at the initial step, its time complexity is $O(w \cdot h \cdot |E|^2)$, while CBS-grid layout requires $O(\min(w,h) \cdot |E|^2)$). Arranging these time complexities by using the above assumptions, we observe that the CBS-grid layout is $\sqrt{C|V|}$ times faster than the CB-grid layout at the initial step.

The method counting edge–edge and vertex–edge crossings in CB-grid layout is naïve one. In the supplementary file, we compare the time complexity of sweep calculation with those of algorithms studied in computational geometry (Akman *et al.*, 1989; Chazelle *et al.*, 1986; Cheng *et al.*, 1991; Palazzi and Snoeyink, 1993; Tunkelang *et al.*, 1994). Here, we just give the conclusion of the comparison. If the average degree of the model can be bounded by $O(\sqrt{|V|})$, sweep calculation is the most efficient of all them. In general, this assumption is reasonable for biological networks due to their small average degree.

3 FLOW PENALIZATION AND ITS CALCULATION

In biological pathways, edges have directions that represent the biochemical reactions and gene regulations. Consider the layout shown in Figure 4a. In general, we can easily trace the directions of edges in the layout since edges connected to a vertex have similar directions. In contrast, when edges have completely different directions as compared to each other, we may face a difficulty in tracing the directions, as shown in Figure 4b. Thus, considering these properties, as a component

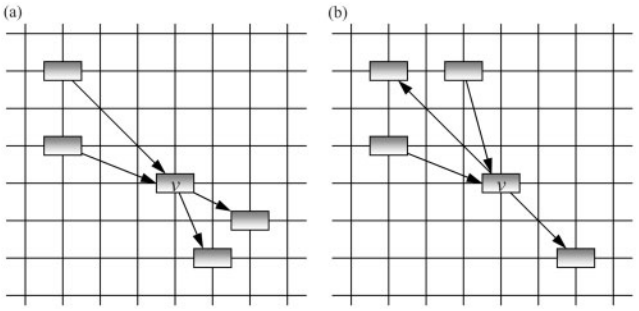

Fig. 4. Examples of (**a**) rational edge directions and (**b**) undesirable edge directions. In (a) edges connected to vertex v have similar directions, and we may easily capture the flow represented by the directions, while the example shown in (b) lacks the traceability of flows due to the different directions.

for the cost function, we define the new function that penalizes undesirable edge directions, as shown in Figure 4b:

$$\mathrm{Flow}_v(L) = -\sum_{e_{\mathrm{in}} \in E_v^{\mathrm{in}}, e_{\mathrm{out}} \in E_v^{\mathrm{out}}} < \frac{\overrightarrow{e_{\mathrm{in}}}}{|\overrightarrow{e_{\mathrm{in}}}|}, \frac{\overrightarrow{e_{\mathrm{out}}}}{|\overrightarrow{e_{\mathrm{out}}}|} >, \tag{3}$$

where E_v^{in} and E_v^{out} are sets of in-edges and out-edges connected to v, respectively; $\overrightarrow{e}$, a vector associated with edge e; operation $< \cdot, \cdot >$, the inner product; and $|\overrightarrow{e}|$, the length of vector $\overrightarrow{e}$.

Note that for edge e of length 0, let $\overrightarrow{e}/|\overrightarrow{e}|$ be 0. By using this function, we redefine the cost function given in Equation (2) as:

$$C_f(L) = C(L) + W_{\mathrm{fl}} \sum_{v \in V} \mathrm{Flow}_v(L), \tag{4}$$

where W_{fl} is a constant value called the *flow penalizing weight*.

The movement of vertex v changes Flow_v and $\mathrm{Flow}_{v'}$ related to edge $e(v, v')$, where v' is each adjacent vertex of v. Since Equation (4) can be rearranged as

$$\mathrm{Flow}_v(L) = -\left\langle \sum_{e_{\mathrm{in}} \in E_v^{\mathrm{in}}} \frac{\overrightarrow{e_{\mathrm{in}}}}{|\overrightarrow{e_{\mathrm{in}}}|}, \sum_{e_{\mathrm{out}} \in E_v^{\mathrm{out}}} \frac{\overrightarrow{e_{\mathrm{out}}}}{|\overrightarrow{e_{\mathrm{out}}}|} \right\rangle, \tag{5}$$

if $\sum_{e_{\mathrm{in}} \in E_v^{in}} \overrightarrow{e_{\mathrm{in}}}/|\overrightarrow{e_{\mathrm{in}}}|$ and $\sum_{e_{\mathrm{out}} \in E_v^{\mathrm{out}}} \overrightarrow{e_{\mathrm{out}}}/|\overrightarrow{e_{\mathrm{out}}}|$ are pre-calculated, this type of change requires $O(\deg(v))$ for Flow_v and $O(1)$ for $\mathrm{Flow}_{v'}$. Thus, the cost difference for Flow is calculated in $O(w \cdot h \cdot \sum_v \deg(v)) = O(w \cdot h \cdot |E|)$ time at each step. However, the calculation method for Flow still increases the time complexity, and thus we cache flow penalizing costs corresponding to each movement of a vertex and change only the costs influenced by the update of the layout to reduce the time complexity. When v_β was moved at the previous step, the cached flow penalizing costs to be updated are corresponding to v_β, vertices adjacent to v_β and vertices adjacent to $v' \in V_{v_\beta}$, where V_{v_β} is the set of vertices adjacent to v_β. Therefore, the time complexity of the update for the cached flow penalizing cost amounts to $O(w \cdot h \cdot \mathrm{avedeg}^2)$ and hence it does not increase the time complexity of the grid layout algorithm. We call the CBS-grid layout with the cost function given in Equation 4 the *CBSF-grid layout*.

4 EXPERIMENTAL RESULTS

We use 95 pathway models of various sizes from TRANSPATH pathway database (Schacherer *et al.*, 2001) to compare the

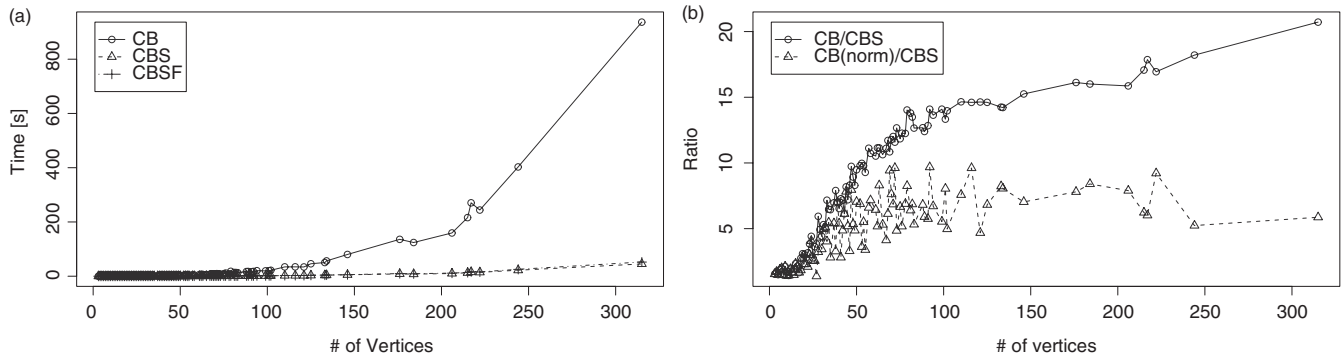

Fig. 5. Comparison of the running time of each grid layout algorithm. (**a**) The running time of the CB, CBS and CBSF-grid layouts for pathway models of various sizes. (**b**) The ratio of the running time of the CB and CBS-grid layouts (CB/CBS) and the ratio of the running time of the CB-grid layout normalized by the squared average degree of each model and that of the CBS-grid layout (CB(norm)/CBS).

performance of the CB-grid, CBS-grid and CBSF-grid layouts. The largest model contains 316 vertices and 592 edges. For each model, we set the numbers of rows w and columns h of a grid to $2\sqrt{|V|}$ and $3\sqrt{|V|}$, respectively. These pathway models do not hold subcellular localization information.

From repeated trials, we empirically decided the weights for the cost function: $W_{\mathrm{dc}} = 100$, $W_{\mathrm{ee}} = 100$, $W_{\mathrm{ve}} = 150$, and $W_{\mathrm{fl}} = 100$.

4.1 Comparison of running time

We generate 10 layouts for each model by randomly mapping vertices to the grid without their overlaps and positional violations. These are preprocessed by the Eades grid layout algorithm (Kojima *et al*., 2007) with 100 iterations and they are then used as initial layouts for the grid layout algorithms. Note that the Eades grid layout on average requires <1s even for the largest model.

For the comparison of the running time, we evaluate the CB, CBS and CBSF-grid layouts. All the algorithms were implemented in Java and experiments were performed on Xeon 3.6 GHz with 4GB RAM.

The running time of each grid layout algorithm is summarized in Figure 5a. These running time results include the time consumed by the Eades grid layout and are averaged over the 10 layouts for each model. From the comparison, we observe that the CB-grid layout is significantly slower than any other grid layout algorithm. The CBS-grid layout is the fastest among them, but since the CBS and CBSF-grid layouts have the same time complexity, there is no large gap among them in terms of running time.

We show the ratio of the running time of the CB and CBS-grid layouts (CB/CBS in Figure 5b) and the ratio of the running time of the CB-grid layout normalized by the average degree of each model and the running time of the CBS-grid layout (CB(norm)CBS in Figure 5b). For small models containing <30 vertices, the CB and CBS-grid layouts do not have large differences, while the CBS-grid layout is more than 10 times faster than the CB-grid layout for relatively large models containing more than 200 vertices. This is because the overhead time dominates in the case of a short-running time.

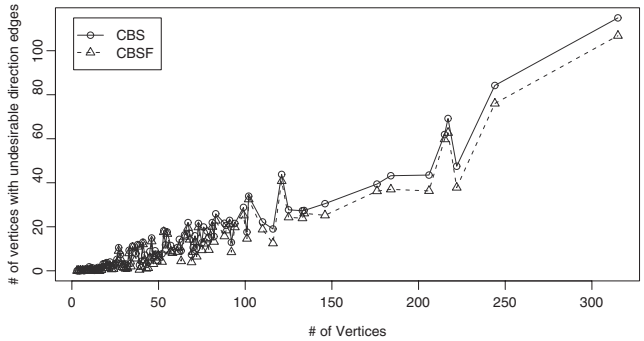

Fig. 6. Comparison of the numbers of vertices having undesirable directions edges between CBS and CBSF-grid layouts. We count the number of vertices having undesirable direction edges in the resulting layouts of CBS and CBSF-grid layouts.

For the case of CB(norm)/CBS, we observe that as the size of the model increase, its ratio becomes constant. Since biological pathway models are known to have relatively small degrees, from the discussion in Section 2.4, the running time of the CB-grid layout normalized by the squared average degree of each model is expected to be proportional to that of the CBS-grid layout. This assessment agrees with the experimental results.

4.2 Effect of flow penalization

We say that a vertex has *undesirable direction edges* if no line passing through the vertex can divide the edges connected to the vertex into in-edges and out-edges. Note that in this definition vertices of degree <4 never have undesirable direction edges. We count the number of vertices having undesirable direction edges in the resulting layouts of CBS and CBSF-grid layout obtained through the previous section process. From the comparison shown in Figure 6, we observe that flow penalizing cost reduces the number of vertices with undesirable direction edges all over the models from TRANSPATH.

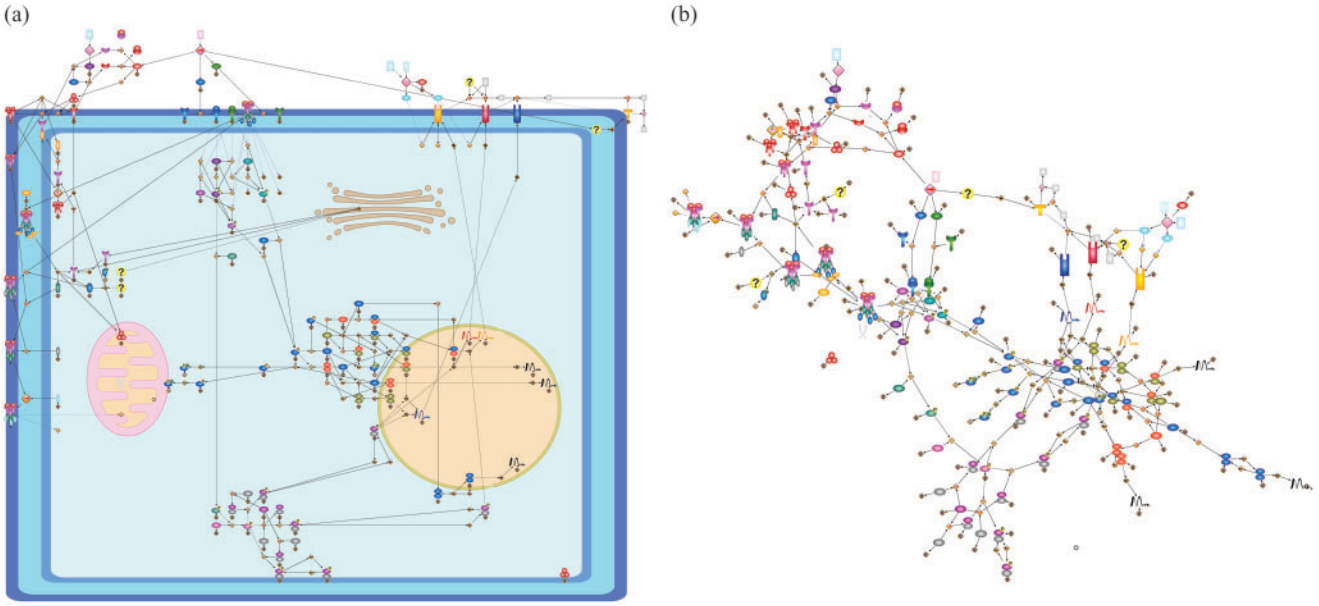

Fig. 7. Comparison of CBSF-grid layout and Jiggle using the endothelial cell model. (**a**) A resulting layout of CBSF-grid layout. (**b**) A resulting layout of Jiggle. Since Jiggle does not use subcellular localizations, they are removed from the layout in (b).

4.3 Comparison with a force-directed approach

We compare our approach with a force-directed approach termed Jiggle proposed by Tunkelang (1998). Following settings are used in Jiggle:

- Conjugate-gradient method is selected as the optimization process due to its efficiency.
- Iteration count is restricted to 1000 since according to the stopping criterion in Tunkelang (1998) Jiggle iterates around 1000 times for the largest model we use.

Figures 7a, b, respectively shows the layouts of CBSF-grid layout and Jiggle for endothelial cell model (Pober *et al.*, 2002), which consists of 318 vertices and 371 edges. Note that subcellular localizations are removed in the resulting layout of Jiggle since Jiggle, does not use them.

As stated in Li and Kurata (2005), due to the definition of distance cost, vertex clusters are formed according to the topological structure of the model in the layout of CBSF-grid layout, thus facilitate to capture the relationship among these topologically decided clusters. In addition, consideration of subcellular localizations and edge directions helps understand the system of the model.

We also compare the running time of CBS and CBSF-grid layouts and Jiggle as in Figure 8 using TRANSPATH models under the same condition of subsection Comparison of the running time. From the comparison our methods are competitive with Jiggle for models of <250 vertices. Although for models of >250 vertices our methods take more running time than Jiggle, their running time is sustainable in practical use.

In the supplementary file we also give the layouts of CBSF-grid layout and Jiggle for *Caenorhabditis elegans* cell-fate model (Saito *et al.*, 2006).

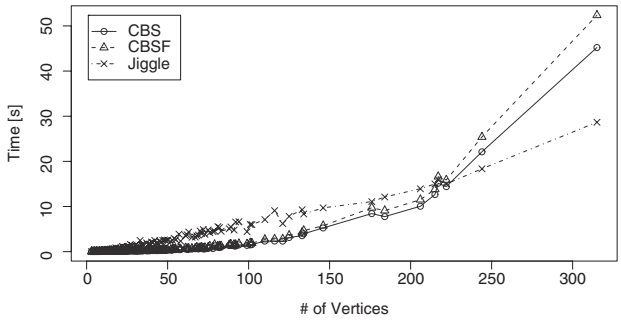

Fig. 8. Comparison of the running time of CBS and CBSF-grid layouts and Jiggle.

5 CONCLUSION

For the purpose of applications to pathway databases, we develop grid layout algorithms and address two issues: computational time and traceability of the flow. We devise sweep calculation and reduce the time complexity of existing grid layout algorithms, although for brevity, our description was limited to the CB-grid layout. Experiments using various models from an actual pathway database prove that our new grid layout algorithm is much faster than the existing grid layout algorithm, and is sufficiently fast for the real-time drawing of models containing <300 vertices. By using flow penalization, we reduce undesirable edge directions that cause a difficulty in tracing the flow of biochemical reactions and gene regulations.

Conflict of Interest: none declared.

REFERENCES

Akman,V. *et al*. (1989) Geometric computing and uniform grid technique. *Comput. Aided Des.*, **21**, 410–420.

Barsky,A. *et al*. (2007) Cerebral: a Cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics*, **23**, 1040–1042.

Battista,D.G. *et al*. (1994) Annotated bibliography on graph drawing algorithms. *Comput. Geom. Theor. Appl.*, **4**, 235–282.

Battista,D.G. *et al*. (1999) *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, New Jersey.

Becker,M.Y. and Rojas,I. (2001) A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, **17**, 461–467.

Brandenburg,F.-J. *et al*. (1997) Algorithmenzum automatischen Zeichnen von Graphen. *Inform. Spektrum*, **20**, 199–207.

Chazelle,B. (1986) Reporting and counting segment intersections. *J. Comput. Syst. Sci.*, **32**, 156–182.

Cheng,S.W. and Janardan,R. (1991) Space-efficient ray-shooting and intersection searching: algorithms, dynamization, and applications. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial & Applied, San Francisco, USA, pp. 7–16.

Deckard,A. *et al*. (2006) Supporting the SBML layout extension. *Bioinformatics*, **22**, 2966–2967.

Demir,E. *et al*. (2002) PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, **18**, 996–1003.

Dogrusoz,U. *et al*. (2004) A compound graph layout algorithm for biological pathways. In *Proceedings of the 12th International Symposium on Graph Drawing*. Springer-Verlag, New York City, USA, pp. 442–447.

Dogrusoz,U. *et al*. (2006) PATIKAweb: a Web interface for analyzing biological pathways through advanced querying and visualization. *Bioinformatics*, **22**, 374–375.

Doi,A. *et al*. (2003) Genomic Object Net: II. Modelling biopathways by hybrid functional Petri net with extension. *Appl. Bioinformatics*, **2**, 185–188.

Genc,B. and Dogrusoz,U. (2003) A constrained, force-directed layout algorithm for biological pathways. In *Proceedings of the 11th International Symposium on Graph Drawing*. Springer-Verlag, Rerugia, Italy, pp. 314–319.

Garcia,O. *et al*. (2007) GOlorize: a cytoscape plug-in for network visualization with Gene Ontology-based layout and coloring. *Bioinformatics*, **23**, 394–396.

Gary,M.R. and Johnson,D.S. (1983) Crossing number is NP-complete. *SIAM J. Algebra. Discr.*, **4**, 312–316.

Gauges,R. *et al*. (2006) A model diagram layout extension for SBML. *Bioinformatics*, **22**, 1879–1885.

Kanehisa,M. (2002) The KEGG database. *Novartis Found Symp.*, **247**, 91–101.

Karp,P.D. and Paley,S.M. (1994) Automated drawing of metabolic pathways. In *Proceedings of the 3rd International Conference on Bioinformatics and Genome Research*. World Scientific Pub. Co. Inc., Florida, USA, pp. 225–238.

Kato,M. *et al*. (2005) Automatic drawing of biological networks using cross cost and subcomponent data. *Genome Inform.*, **16**, 22–31.

Kojima,K. *et al*. (2007) An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC Bioinformatics*, **8**, 1–16.

Kurata,H. *et al*. (2003) CADLIVE for constructing a large-scale biochemical network based on a simulation-directed notation and its application to yeast cell cycle. *Nucleic Acids Res.*, **31**, 4071–4084.

Kurata,H. *et al*. (2005) CADLIVE dynamic simulator: direct link of biochemical networks to dynamic models. *Genome Res.*, **15**, 590–600.

Li,W. and Kurata,H. (2005) A grid layout algorithm for automatic drawing of biochemical networks, B*ioinformatics*, **21**, 2036–2042.

Nagasaki,M. *et al*. (2003) Genomic Object Net: I. A platform for modelling and simulating biopathways. *Appl. Bioinformatics*, **2**, 181–184.

Palazzi,L. and Snoeyink,J. (1993) Counting and reporting red/blue segment intersections. *CVGIP: Graph. Model. Im.*, **56**, 304–310.

Pober,J.S. (2002) Endothelial activation: intercellular signalizng pathways. *Arthritis Res.*, **4**, S109–S116.

Saito,A. *et al*. (2006) Cell fate simulation model of gustatory nuerons with microRNAs double-negative feedback loop by hybrid functional Petri net with extension. *Genome Inform.*, **17**, 100–111.

Schacherer,F. *et al*. (2001) The TRANSPATH signal transduction database: a knowledge base on signal transduction networks. *Bioinformatics*, **17**, 1053–1057.

Shannon,P. *et al*. (2003) Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.*, **13**, 2498–2504.

Tunkelang,D. (1994) A practical appraoch to drawing undirected graphs. *Technical Report CMUCS-94-161*. Carnegie Mellon University, School of Computer Science.

Tunkelang,D. (1998) JIGGLE: Java interactive graph layout environment. In *Proceedings of the 6th International Symposium on Graph Drawing*. Springer-Verlag, Montréal, Canada, pp. 412–422.

Wegner,K. and Kummer,U. (2005) A new dynamical layout algorithm for complex biochemical reaction networks. *BMC Bioinformatics*, **6**, 1–12.