

A grid layout algorithm for automatic drawing of  
biochemical networks

Weijiang Li and Hiroyuki Kurata

Seminar: Visualisierung in der Bioinformatik  
Andre Wiesniewski

9. August 2007

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Über die Autoren</b>	<b>3</b>
<b>3</b>	<b>Motivation</b>	<b>3</b>
3.1	Was ist ein Layout? . . . . .	3
3.2	Die Situation in der Biologie . . . . .	3
<b>4</b>	<b>Force-directed Layouts</b>	<b>4</b>
4.1	Idee . . . . .	4
4.2	Anwendung auf biologische Netzwerke . . . . .	4
<b>5</b>	<b>Grid-Layout</b>	<b>5</b>
5.1	Idee . . . . .	5
5.2	Kostenfunktion . . . . .	6
5.3	Kandidatenlayouts . . . . .	7
5.4	Nachbarlayouts . . . . .	8
5.5	Ablauf des Algorithmus . . . . .	8
5.6	Die Methode <b>LOKALESMINIMUM(<math>\mathbf{R}</math>)</b> . . . . .	11
5.6.1	Einfache Variante . . . . .	11
5.6.2	Effizientere Variante . . . . .	12
5.7	Auswahl der Parameter . . . . .	13
5.8	Vergleich anhand eines Beispiels . . . . .	14
5.9	Laufzeit . . . . .	17
<b>6</b>	<b>Fazit</b>	<b>17</b>
<b>7</b>	<b>Literaturverzeichnis</b>	<b>18</b>

# 1 Einleitung

Im Rahmen des Seminars *Visualisierung in der Bioinformatik* am Lehrstuhl 11 des Fachbereichs Informatik der Universität Dortmund habe ich mich mit der Abhandlung „A grid layout algorithm for automatic drawing of biochemical networks“ von Li und Kurata beschäftigt. Die Autoren stellen darin einen Layoutalgorithmus für komplexe biologische Netzwerke vor. Das Ziel dieses Layouts ist es, ein besseres Verständnis der topologischen Struktur des Netzwerkes zu vermitteln.

# 2 Über die Autoren

Die Autoren Weijiang Li und Hiroyuki Kurata sind Mitarbeiter des Department of Bioscience and Bioinformatics am Kyushu Institute of Technology in Japan. Ihre Arbeit „A grid layout algorithm for automatic drawing of biochemical networks“ wurde im Jahre 2005 im Bioinformatics - Magazin Vol. 21, No.9 (Seiten 2036-2042) publiziert.

# 3 Motivation

## 3.1 Was ist ein Layout?

Zu Beginn habe ich mir die Frage gestellt, was das Layout eines Graphen eigentlich ist. Dazu habe ich in der Dissertation „Layout of Graph Visualizations“ von Ulrik Brandes einen passenden Absatz gefunden: „Die entscheidende Aufgabe bei der Visualisierung von Graphen [...] ist der Abgleich der räumlichen Anordnung innerhalb des Diagramms mit den strukturellen Eigenschaften des Graphen. Dieser Schritt und sein Ergebnis werden auch das Layout [...] genannt.“. Die im Zitat erwähnten strukturellen Eigenschaften spielen auch im Layout von Li und Kurata eine große Rolle.

## 3.2 Die Situation in der Biologie

In den letzten Jahren gab es große Fortschritte in der Molekularbiologie. Dies führte zu einer Menge von sehr großen und komplexen biologischen Netzwerken (gene regulatory networks, signal transduction pathways). Diese waren ohne computergestützte Analyseprogramme nur sehr schwer zu verstehen. Außerdem musste für diese Netzwerke eine geeignete Darstellungsform gefunden werden, da die rein textuelle Form dieser Datenflut sehr unübersichtlich ist. Eines dieser Visualisierungstools ist CADLIVE (Computer-Aided Design of LIVING systEms) (Kurata et al., 2003). In CADLIVE und in vielen anderen Programmen werden biologische Netzwerke als Graphen modelliert. Dabei gibt es für jede Komponente des biologischen Netzwerkes eine graphische Notation.

Das größte Problem dieser Visualisierungstools ist das Fehlen eines Layoutalgorithmus, der eine für den Biologen geeignete Darstellung automatisch erzeugt.

So sind die Biologen bisher gezwungen ihre Darstellungen manuell zu erzeugen. Die manuelle Platzierung der Komponenten ist allerdings sehr zeitaufwendig. Wie bereits erwähnt werden biologische Netzwerke oft als Graph dargestellt. Dabei ist es notwendig eine geeignete Informationsreduzierung vorzunehmen, da sonst der Graph zu komplex und damit unübersichtlich wird. Seine eigentliche Funktion, die Daten verständlicher darzustellen, geht sonst wieder verloren. Das Ziel des hier behandelten Grid-Layout-Algorithmus ist es, ein besseres Verständnis der topologischen Struktur des biologischen Netzwerkes zu vermitteln. Es sollen vor allem Cluster, d.h. „eng-verknüpfte“ Knoten dargestellt werden, da diese meist eine funktionale Einheit innerhalb des biologischen Netzwerkes bilden.

## 4 Force-directed Layouts

### 4.1 Idee

Force-directed Layoutalgorithmen werden oft zur Darstellung von großen Graphen benutzt. Die Idee hinter diesen Algorithmen ist es, den Graphen als mechanisches System zu verstehen. Die Knoten des Graphen stoßen sich ab und die Kanten halten sie zusammen. Man kann sich dabei eine Kante als eine mechanische Feder zwischen zwei Knoten vorstellen. Deshalb werden auch nur die



Abbildung 1: Knoten stoßen sich ab, Kanten halten diese zusammen.

Knoten zusammengehalten, die direkt durch eine Kante verbunden sind. Die Abstossungs- und Anziehungskräfte die zwischen den Komponenten des Graphen wirken, werden durch eine Energiefunktion oder auch Kostenfunktion modelliert. Ein Layout des Graphen ist ein Zustand, bei dem die Energie des System minimal ist. Force-directed Layouts werden oft für sehr große Graphen benutzt, z.B. PPI-Netzwerke (*Ju and Han, 2003; Han and Ju, 2003; Ju and Han, 2003*).

### 4.2 Anwendung auf biologische Netzwerke

Wenn man force-directed Layouts auf die hier betrachteten biologischen Netzwerke anwendet, gibt es zwei Nachteile:

1. Schlechte Identifikation der Cluster:  
Wie bereits oben kurz erwähnt, bilden eng-verknüpfte Knoten meist eine funktionale Einheit. Deshalb sollten diese Knoten auch im Layout eine benachbarte Position haben. Da bei den force-directed Layouts aber nur direkt-verknüpfte Knoten zusammengehalten werden, werden die eng-verknüpften Knoten nicht immer nah beieinander plaziert. Dies führt dazu, dass man Cluster nicht klar erkennen kann.

## 2. Geometrische Verteilung der Knoten:

Aufgrund der Abstossungskräfte werden die Knoten oft zerstreut, so dass es selten möglich ist ein kompaktes Layout zu erhalten. Die Anziehungskräfte dagegen führen dazu, dass die minimale Distanz zwischen zwei Knoten sehr klein werden kann. Dadurch kann es zu Knotenüberlappungen kommen, vor allem wenn die Knoten größere Symbole entsprechend einer biologischen Notation sind.

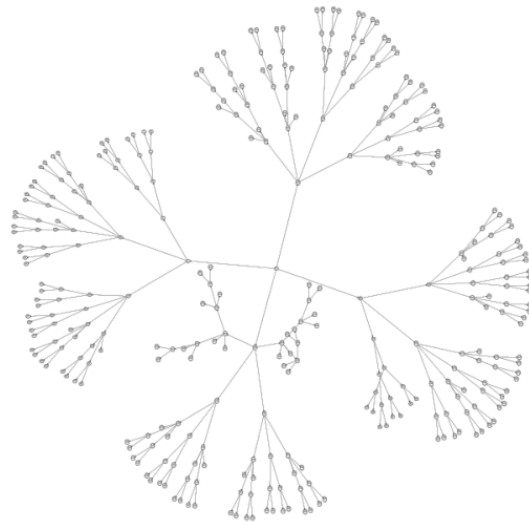


Abbildung 2: Force-directed Layout (Quelle: *aiSee: Galerie der Graphen*, <http://www.absint.com/>)

Die Autoren Li und Kurata haben daher versucht, einen Layoutalgorithmus zu entwickeln, bei dem diese Nachteile ausgeräumt werden und bei dem man die topologische Struktur des biologischen Netzwerkes klar erkennen kann. Ihr Layout nennen Sie „A grid layout algorithm for automatic drawing of biochemical networks“.

## 5 Grid-Layout

### 5.1 Idee

Beim Grid-Layout wird das biologische Netzwerk auch als interaktives System von Knoten modelliert. Die Knoten interagieren wieder entsprechend einer Energiefunktion. Die Autoren versuchen jedoch die Struktur des biologischen Netzwerkes besser darzustellen, indem sie die Abstossungs- und Anziehungskräfte neu definieren. Eng-verknüpfte Knoten sollen sich anziehen und entfernt-verknüpfte Knoten sollen sich abstossen. Dadurch sollen Cluster besser erkennbar werden. Außerdem werden die Knoten auf einem 2D-Raster gezeichnet, d.h.

es gibt nur diskrete Koordinaten. Deshalb auch der Name Grid-Layout. Knotenüberlappungen lassen sich vermeiden, indem man die Auflösung des Rasters an die Knoten- bzw. Symbolgröße anpasst. Das Grid-Layout ist also ein diskretes force-directed Layout.

Da die Knoten entsprechend einer Energiefunktion interagieren, kann man eine Konfiguration dieser Knoten als ein Layout des Graphen betrachten. Die Energie einer solchen Konfiguration entspricht den Kosten eines Layouts. Eine stabile Konfiguration hat niedrige Energie; äquivalent dazu hat ein akzeptables Layout niedrige Kosten. Der Grid-Layout-Algorithmus ist somit ein Optimierungsalgorithmus bzw. ein Minimierungsalgorithmus der Kostenfunktion des Layouts. Die Kostenfunktion des Grid-Layouts wird im nächsten Kapitel definiert.

## 5.2 Kostenfunktion

Ein Layout  $\mathbf{R}$  wird durch den Vektor der Knotenkoordinaten beschrieben:

$\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n)$ , mit  $n$  Knoten und den **ganzzahligen** Koordinaten  $\mathbf{r}_i = (x_i, y_i)$ .

Die Menge aller möglichen Koordinaten ist  $L$ . Insgesamt gibt es  $|L| = m$  viele Koordinaten auf denen die  $n$  Knoten platziert werden können.

Der Kostenwert des Layouts  $\mathbf{R}$  ist die Summe der Kosten zwischen allen Knotenpaaren:

$$f(\mathbf{R}) = \sum_{i < j} f_{ij}(\mathbf{R}) \quad (1)$$

Die Kosten zwischen zwei Knoten  $i$  und  $j$  sind das Produkt zwischen dem Kostengewicht  $w_{ij}$  und dem Abstand  $d(\mathbf{r}_i, \mathbf{r}_j)$  der Knoten:

$$f_{ij}(\mathbf{R}) = w_{ij} * d(\mathbf{r}_i, \mathbf{r}_j) \quad (2)$$

Zur Vereinfachung wird für die Distanz  $d(\mathbf{r}_i, \mathbf{r}_j)$  zwischen zwei Knoten  $i$  und  $j$  der vereinfachte euklidische Abstand benutzt:

$$d(\mathbf{r}_i, \mathbf{r}_j) = |x_i - x_j| + |y_i - y_j| \quad (3)$$

Dieser kann schneller berechnet werden und die Genauigkeit genügt für die weiteren Berechnungen. Desweiteren hängen die Kosten ab von dem Kostengewicht  $w_{ij}$  zwischen den Knoten  $i$  und  $j$ . Hierzu sollte man sich noch mal an das Ziel der Kostenfunktion erinnern: Eng-verknüpfte Knoten sollen möglichst nah und entfernt-verknüpfte Knoten sollen möglichst weit auseinander platziert werden. Dies wird erreicht, indem eng-verknüpften Knotenpaaren ein positives Kostengewicht und entfernt-verknüpften Knotenpaaren ein negatives Kostengewicht zugeordnet wird. Denn wenn ein Knotenpaar ein positives Kostengewicht hat, muss die Distanz zwischen ihnen möglichst klein gewählt werden, um die Kostenfunktion des Layouts zu minimieren. Analog für Knotenpaare mit negativem Kostengewicht.

Ein übliches Kriterium für die Definition eng-verknüpfter Knoten ist die Anzahl der kürzesten Wege zwischen ihnen. Der Wert  $M_{\alpha\beta}^{(k)}$  ist die Anzahl der  $k$ -Pfade

zwischen Knoten  $\alpha$  und  $\beta$ . Ein  $k$ -Pfad ist definiert als ein Pfad mit Länge  $\leq k$ . Unter Berücksichtigung der oben genannten Punkte führt dies zu folgender Verteilung der positiven und negativen Werte bei der Definition des Kostengewichtes  $w_{ij}$ :

$$w_{ij} = \begin{cases} 3, & \text{falls } M_{ij}^{(1)} > 0; \\ 1, & \text{falls } M_{ij}^{(1)} = 0 \text{ und } M_{ij}^{(2)} > 0; \\ 0, & \text{falls } M_{ij}^{(2)} = 0 \text{ und } M_{ij}^{(3)} > 0; \\ -1, & \text{falls } M_{ij}^{(3)} = 0 \text{ und } M_{ij}^{(4)} > 0; \\ -2, & \text{sonst} \end{cases} \quad (4)$$

Die Autoren sagen allerdings nicht, wie Sie zu den genauen Werten gekommen sind. Es wird sich noch zeigen, dass viele Parameter empirisch bestimmt wurden.

Da wir auch negative Werte bei den Kostengewichten haben, könnte man die Distanz zwischen zwei Knoten beliebig groß wählen, um die Kosten des Layouts immer weiter zu minimieren. Dadurch würde aber kein kompaktes Layout entstehen. Somit ist es notwendig eine obere Grenze für die Distanz anzugeben. Das Kostengewicht zwischen zwei Knoten wird wie folgt neu definiert:

$$f_{ij}(\mathbf{R}) = \psi_{ij}(\mathbf{r}_i, \mathbf{r}_j) \quad (5)$$

mit

$$\psi_{ij}(\mathbf{r}_i, \mathbf{r}_j) = \begin{cases} w_{ij} * d(\mathbf{r}_i, \mathbf{r}_j), & \text{falls } w_{ij} \geq 0; \\ w_{ij} * \min(d(\mathbf{r}_i, \mathbf{r}_j), d_{max}), & \text{sonst.} \end{cases} \quad (6)$$

$d_{max}$  ist die maximale Abstoßungsdistanz.

### 5.3 Kandidatenlayouts

Aus einem biologischen Netzwerk können sehr viele Layouts gebildet werden, indem man alle Knoten beliebig auf freien Koordinaten platziert. Allerdings ist es sehr unwahrscheinlich, dass man bei diesen zufällig gebildeten Layouts ein Layout erhält, das unseren oben genannten Anforderungen entspricht. Deshalb muss man eine sinnvolle Vorauswahl treffen. Die Autoren benutzen hierzu den Begriff des Kandidatenlayouts.

Ein Kandidatenlayout ist ein Layout mit niedrigen Kosten. Außerdem sollte das Layout stabil sein, d.h. dass man durch die Verschiebung eines Knoten auf einen anderen freien Punkt kein besseres, also kostengünstigeres, Layout bekommt. Diese Verschiebung eines Knoten auf einen anderen freien Platz wird kleinste Änderung genannt. Es ist die einfachste Operation auf einem Layout und wird formal durch den Operator  $T_{\alpha\mathbf{p}}$  ausgedrückt. Dabei ist  $\alpha$  ein Knoten und  $\mathbf{p}$  ein freier Punkt in  $L$ . Wenn ein Layout  $\mathbf{R}$  gegeben ist, dann ist das Layout  $T_{\alpha\mathbf{p}}\mathbf{R}$  das gleiche Layout, aber mit Knoten  $\alpha$  an der Stelle  $\mathbf{p}$ . Ein Kandidatenlayout ist somit ein lokales Minimum der Kostenfunktion. Ein optimales Layout wäre dementsprechend ein globales Minimum der Kostenfunktion.

Die Methode  $\text{LOKALES MINIMUM}(\mathbf{R})$  berechnet aus einem gegebenen Layout  $\mathbf{R}$

das dazugehörige Kandidatenlayout  $\mathbf{R}'$ . Die Autoren stellen für die Berechnung zwei Varianten vor. Die erste Variante ist leicht verständlich, hat aber eine hohe Komplexität. Die zweite Variante ist effizienter, indem einige Sonderfälle betrachtet werden. Durch diese Sonderfälle wird die Variante aber schwerer verständlich.

Eine genauere Betrachtung der beiden Varianten erfolgt in Kapitel 5.6.

## 5.4 Nachbarlayouts

Neben den Kandidatenlayouts führen die Autoren einen weiteren Begriff ein, die Nachbarlayouts. Um von einem gegebenen Layout  $\mathbf{R}$  zu einem Nachbarlayout  $\mathbf{R}'$  zu kommen, werden alle Knoten mit der Wahrscheinlichkeit  $p$  auf einen anderen freien Punkt verschoben. Bei  $p = 1$  erhält man ein komplett anderes Layout; analog dazu bleibt bei  $p = 0$  das Layout unverändert.

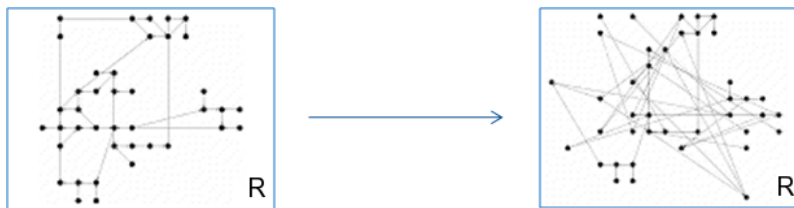


Abbildung 3: Aus dem Layout  $\mathbf{R}$  wird ein Nachbarlayout  $\mathbf{R}'$  erzeugt,  $p = 0.55$  (Quelle: *Li, Weijiang und Kurata, Hiroyuki*)

## 5.5 Ablauf des Algorithmus

Der Grid-Layout-Algorithmus benutzt das bekannte Verfahren der simulierten Abkühlung (simulated annealing). Das ist ein heuristisches Optimierungsverfahren, bei dem ein Abkühlungsprozess nachgebildet wird. Zu Beginn hat das System (der Algorithmus) eine maximale Temperatur. Bei jeder Iteration kühlt sich das System ab, bis es eine Minimaltemperatur erreicht hat. Dann ist das System eingefroren und der Algorithmus terminiert. Ein besonderes Merkmal ist, dass sich während des Abkühlungsprozesses die Zwischenergebnisse verschlechtern können. Dadurch kann das System von einem lokalen Minimum zu einem möglicherweise besseren lokalen Minimum gelangen. Dieses Verfahren wird häufig für Optimierungsprobleme mit hoher Komplexität eingesetzt, unter anderem auch im Bereich des Graphzeichnens (R. Davidson, D. Harel. *Drawing graphs nicely using simulated annealing*. ACM Transactions on Graphics, Volume 15(4):301–331, 1996).

Zu Beginn des Grid-Layout Algorithmus wird ein randomisiertes Layout  $\mathbf{R}$  generiert. Layout  $\mathbf{R}$  wird durch die Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  in ein Kandidatenlayout umgewandelt. Jetzt startet die simulierte Abkühlung

mit der Temperatur  $T = T_{max}$ . Solange das System noch nicht eingefroren ist, werden folgende Schritte  $n_e$ -mal wiederholt ( $n_e$  ist ein weiterer Parameter, der empirisch bestimmt werden muss): Zu dem Kandidatenlayout  $\mathbf{R}$  wird ein Nachbarlayout  $\mathbf{R}'$  berechnet. Dieses wird auch durch die Methode  $\text{LOKALESMINIMUM}(\mathbf{R}')$  in ein Kandidatenlayout umgewandelt. Somit sind jetzt sowohl  $\mathbf{R}$  als auch  $\mathbf{R}'$  lokale Minima der Kostenfunktion. Die Kosten der beiden Layouts werden verglichen. Wenn die Kosten von  $\mathbf{R}'$  kleiner sind als die Kosten von  $\mathbf{R}$ , wird Layout  $\mathbf{R}'$  als bisher bestes Layout  $\mathbf{R}_{min}$  gespeichert. Falls die Kosten von  $\mathbf{R}'$  nicht günstiger sind, wird  $\mathbf{R}'$  trotzdem mit einer gewissen Wahrscheinlichkeit akzeptiert. Damit ist garantiert, dass sich das Zwischenergebnis in einigen Fälle verschlechtert. Am Ende jeder Iteration wird die aktuelle Temperatur  $T$  des Systems um den Faktor  $r_c$  verringert, damit der Algorithmus nach endlicher Zeit terminiert. Die Verschlechterung des Zwischenergebnisses kann dazu führen, dass bei der nächsten Iteration ein Nachbarlayout generiert werden kann, dessen Kandidatenlayout günstigere Kosten hat, als das bisher beste Layout. Wenn der Algorithmus terminiert ist, ist  $\mathbf{R}_{min}$  das günstigste bisher gefundene Layout.

Zusammenfassend könnte man den Grid-Layout-Algorithmus folgendermaßen beschreiben: Kandidatenlayouts werden durch die Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  generiert und das Verfahren der simulierten Abkühlung wählt bessere Kandidatenlayouts aus.

Der Grid-Layout-Algorithmus wird mit folgenden Parametern gestartet:  $T_{max}$  ist die Starttemperatur und  $T_{min}$  die Minimaltemperatur, die die Abbruchbedingung darstellt. Bei jeder Iteration wird die aktuelle Temperatur um den Abkühlungsfaktor  $r_c$  verringert. Nach dem Durchlauf des Algorithmus ist das günstigste gefundene Layout  $R_{min}$  mit dem dazugehörigen Kostenwert  $f_{min}$ .  $n_e$  ist die Anzahl der Wiederholungen innerhalb einer Iteration des Algorithmus und  $p$  die Wahrscheinlichkeit, mit der ein Knoten bei der Generierung des Nachbarlayouts verschoben wird. Die Parameter  $n_e$  und  $p$  sind Werte, die empirisch ermittelt werden müssen.

Der Grid-Layout-Algorithmus in Pseudocode:

**GridLayout**( $T_{max}, T_{min}, \mathbf{R}_{min}, f_{min}, r_c, n_e, p$ )

- (1)  $T \leftarrow T_{max}$ ,  $\mathbf{R} \leftarrow$  ein zufälliges Layout
- (2)  $f_{min} \leftarrow f \leftarrow \text{LokalesMinimum}(\mathbf{R})$ ,  $\mathbf{R}_{min} \leftarrow \mathbf{R}$
- (3) **do while**  $T > T_{min}$
- (4)     **repeat**
- (5)          $\mathbf{R}' \leftarrow \text{Nachbar}(\mathbf{R}, p)$
- (6)          $f' \leftarrow \text{LokalesMinimum}(\mathbf{R}')$
- (7)          $\varepsilon \leftarrow$  eine (0,1)-Zufallszahl
- (8)         **if**  $\varepsilon < \exp((f - f')/T)$  **then**
- (9)              $f \leftarrow f'$ ,  $\mathbf{R} \leftarrow \mathbf{R}'$
- (10)            **if**  $f < f_{min}$  **then**
- (11)                  $f_{min} \leftarrow f$ ,  $\mathbf{R}_{min} \leftarrow \mathbf{R}$

```

(12)         endif
(13)     endif
(14) until  $n_e$  mal
(15)      $T \leftarrow r_c T$ 
(16) enddo

```

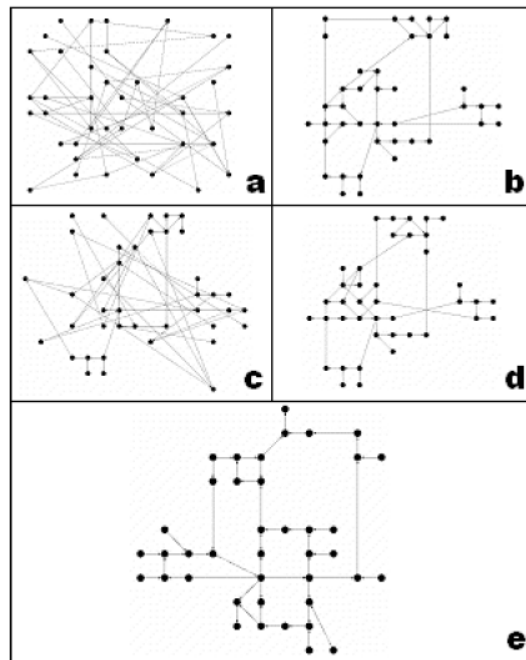


Abbildung 4: Der Ablauf des Algorithmus am Beispiel des Escherichia coli heat shock response regulatory network: (a) initiales zufälliges Layout; (b) Kandidatenlayout durch Anwendung der Methode  $\text{LOKALES MINIMUM}(\mathbf{R})$  auf das zufällige Layout (a); (c) Nachbarlayout zu (b); (d) Kandidatenlayout generiert durch das Nachbarlayout zu (b); (e) Ausgabelayout am Ende des Algorithmus. (Quelle: *Li, Weijiang und Kurata, Hiroyuki*)

## 5.6 Die Methode $\text{LOKALESMINIMUM}(\mathbf{R})$

Wie bereits erwähnt stellen die Autoren zwei Varianten zur Berechnung der Kandidatenlayouts vor.

### 5.6.1 Einfache Variante

Die einfache Variante der Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  überprüft in jeder Phase alle möglichen kleinsten Änderungen auf dem gegebenen Layout  $\mathbf{R}$  und speichert die beste Änderung ab. Falls es mehrere beste Änderungen gibt, wird nur die zuerst gefundene Änderung gespeichert. Diese Verschiebung eines Punktes wird auf das gegebene Layout angewandt und der Algorithmus startet eine neue Phase. Die Methode bricht ab, wenn die Kosten des gesamten Layouts nicht mehr durch eine kleinste Änderung verbessert werden können.

Das resultierende Layout ist ein lokales Minimum der Kostenfunktion. Diese Variante ist jedoch sehr komplex, da in jeder Phase alle möglichen kleinsten Änderungen berechnet werden müssen. Es gibt  $n$  Knoten und  $(m - n)$  freie Punkte, so dass die for-Schleife  $n(m - n)$ -mal durchlaufen werden muss. Allein die Berechnung der Kosten eines Layouts hat die Komplexität  $\mathcal{O}(n^2)$ , da die Summe aller Knotenpaare berechnet werden muss. Aufgrund dieser hohen Komplexität war es nötig, eine verbesserte Variante zu entwickeln.

Der in der Originalarbeit enthaltene Pseudocode für die einfache Variante der Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  enthält einen Fehler. Durch diesen Fehler ist am Ende der Methode  $\mathbf{R}_{\min}$  zwar das beste Layout, aber der minimale Kostenwert  $f_{\min}$  ist der Kostenwert des ursprünglichen Layouts. Der Fehler wird korrigiert, indem man die ersten beiden Zeilen vertauscht.

Die verbesserte Version der einfachen Variante der Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  in Pseudocode:

#### **LokalesMinimum**( $\mathbf{R}$ )

```
(2) repeat
(1)  $f_0 \leftarrow f(\mathbf{R})$ 
(3)  $f_{\min} \leftarrow f_0$ 
(4) for jeden Knoten  $\alpha$  und jeden freien Punkt  $\mathbf{p} \in L$ 
(5)  $f_{\text{trial}} \leftarrow f(T_{\alpha\mathbf{p}}\mathbf{R})$ 
(6) if  $f_{\text{trial}} < f_{\min}$  then
(7)  $f_{\min} \leftarrow f_{\text{trial}}, \quad \beta \leftarrow \alpha, \quad \mathbf{q} \leftarrow \mathbf{p}$ 
(8) endif
(9) endfor
(10) if  $f_{\min} = f_0$  then return  $f_{\min}$ 
(11)  $\mathbf{R} \leftarrow T_{\beta\mathbf{q}}\mathbf{R}$ 
(12) endrepeat
```

### 5.6.2 Effizientere Variante

Die Kostendifferenz zwischen einem Layout  $\mathbf{R}$  und dem Layout  $T_{\beta q}\mathbf{R}$ , also nachdem die kleinste Änderung  $T_{\beta q}$  auf dem Layout  $\mathbf{R}$  durchgeführt worden ist, definieren die Autoren wie folgt:

$$\Delta_{\alpha p}(\mathbf{R}) = f(T_{\alpha p}\mathbf{R}) - f(\mathbf{R}) \quad (7)$$

Alle diese  $\Delta_{\alpha p}$ -Werte bilden eine  $\Delta$ -Matrix der Größe  $n \times m$ . Ist ein Eintrag negativ, dann kann durch die dazugehörige kleinste Änderung das Layout verbessert werden. Der minimale Wert der  $\Delta$ -Matrix ist die beste kleinste Änderung, die man auf dem gegebenen Layout ausführen kann. Genau dieser Wert wird ja bereits indirekt bei der einfachen Variante der Methode  $\text{LOKALESMINIMUM}(\mathbf{R})$  berechnet, um das Kandidatenlayout zu erzeugen.

Die Idee der Effizienzsteigerung ist folgende: Bei einer kleinsten Änderung wird per Definition nur ein einziger Knoten verschoben. Der größte Teil der Kostenfunktion des Layouts bleibt somit identisch.

Wenn das gegebene Layout  $\mathbf{R}$  durch eine kleinste Änderung  $T_{\beta q}$  in ein Layout  $T_{\beta q}\mathbf{R}$  überführt wird, kann die neue  $\Delta$ -Matrix durch eine Aktualisierung der alten  $\Delta$ -Matrix in deutlich kürzerer Zeit berechnet werden.

Die Berechnung der neuen Werte der  $\Delta$ -Matrix lässt sich meist in  $\mathcal{O}(1)$  durchführen, nur in  $(n - 1)$  Spezialfällen benötigt die Berechnung der Werte  $\mathcal{O}(n)$ . Leider beweisen oder erklären die Autoren in ihrer Abhandlung die Formel zur Berechnung dieser Werte nicht, deshalb möchte ich in meiner Ausarbeitung auf die  $(n - 1)$  Spezialfällen eingehen und begründen, warum diese aufwändiger zu berechnen sind.

Zu Beginn der effizienteren Variante muss einmal die komplette  $\Delta$ -Matrix des gegebenen Layouts  $\mathbf{R}$  berechnet werden. Danach lässt sich direkt durch das minimale Element der Matrix die beste kleinste Änderung  $T_{\beta q}$  bestimmen. Somit haben wir das Layout  $T_{\beta q}\mathbf{R}$  gegeben. Jetzt folgt die Aktualisierung der  $\Delta$ -Matrix, d.h. alle  $\Delta_{\alpha p}(T_{\beta q}\mathbf{R})$  müssen aktualisiert werden. Diese Werte geben die Kostendifferenz zwischen unserem bisherigen Layout  $T_{\beta q}\mathbf{R}$  und dem Layout  $T_{\alpha p}T_{\beta q}\mathbf{R}$ , d.h. inklusive der Verschiebung des Knoten  $\alpha$  auf den Punkt  $q$ , an. Bei dieser Verschiebung müssen drei Fälle unterschieden werden:

1. Fall:  $\alpha = \beta$ , d.h. wir betrachten die Verschiebung desselben Knotens, den wir bei der ersten kleinsten Änderung bereits verschoben haben. In diesem Fall lässt sich die Kostendifferenz durch folgende Formel berechnen:

$$\Delta_{\alpha p}(T_{\beta q}\mathbf{R}) = \Delta_{\beta p}(T_{\beta q}\mathbf{R}) = \Delta_{\beta p}(\mathbf{R}) - \Delta_{\beta q}(\mathbf{R}) \quad (8)$$

Die Berechnung ist in  $\mathcal{O}(1)$  möglich, da  $\Delta_{\beta p}(\mathbf{R})$  und  $\Delta_{\beta q}(\mathbf{R})$  bereits in der  $\Delta$ -Matrix von Layout  $\mathbf{R}$  stehen.

2. Fall:  $\alpha \neq \beta$ ,  $\mathbf{p} \neq \mathbf{r}_\beta$ , d.h. wir betrachten die Verschiebung des Knoten  $\alpha$ , dieser ist ungleich Knoten  $\beta$ , auf den Punkt  $p$ . Punkt  $p$  ist verschieden von der alten Position  $\mathbf{r}_\beta$  von Knoten  $\beta$ .

Die Kostendifferenz  $\Delta_{\alpha\mathbf{p}}(T_{\beta\mathbf{q}}\mathbf{R})$  kann durch folgende Formel berechnet werden:

$$\Delta_{\alpha\mathbf{p}}(T_{\beta\mathbf{q}}\mathbf{R}) = \Delta_{\alpha\mathbf{p}}(\mathbf{R}) + f_{\alpha\beta}(T_{\alpha\mathbf{p}}T_{\beta\mathbf{q}}R) - f_{\alpha\beta}(T_{\beta\mathbf{q}}R) - f_{\alpha\beta}(T_{\alpha\mathbf{p}}R) + f_{\alpha\beta}(R) \quad (9)$$

Diese Formel benutzt den Wert  $\Delta_{\alpha\mathbf{p}}(\mathbf{R})$ . Das ist die Kostendifferenz zwischen dem ursprünglichen Layout  $\mathbf{R}$  und dem Layout, bei dem der Knoten  $\alpha$  auf die Position  $p$  verschoben worden ist. Dieser Wert kann aus der zu Beginn berechneten  $\Delta$ -Matrix in  $\mathcal{O}(1)$  ausgelesen werden. Die vier übrigen  $f_{\alpha\beta}$ -Werte, die Kosten zwischen den Knoten  $\alpha$  und  $\beta$ , lassen sich jeweils in  $\mathcal{O}(1)$  durch Formel (2) berechnen. Insgesamt lässt sich somit die Kostendifferenz wieder in  $\mathcal{O}(1)$  berechnen.

3. Fall:  $\alpha \neq \beta$ ,  $\mathbf{p} = \mathbf{r}_\beta$ , d.h. wir betrachten die Verschiebung des Knoten  $\alpha$ , dieser ist erneut ungleich Knoten  $\beta$ , auf den Punkt  $p$ . Aber Punkt  $p$  ist diesmal die alte Position  $\mathbf{r}_\beta$  von Knoten  $\beta$ .

Wenn man nun versucht die Kostendifferenz wie im zweiten Sonderfall zu berechnen, benötigt man wieder den Wert  $\Delta_{\alpha\mathbf{p}}(\mathbf{R})$ . Dieser lässt sich jedoch nicht aus der alten  $\Delta$ -Matrix von Layout  $\mathbf{R}$  auslesen, da beim Layout  $\mathbf{R}$  die Position  $p$  von Knoten  $\beta$  besetzt war. Somit muss die Kostendifferenz  $\Delta_{\alpha\mathbf{p}}(T_{\beta\mathbf{q}}\mathbf{R})$  durch folgende Formel aufwendig berechnet werden:

$$\Delta_{\alpha\mathbf{p}}(T_{\beta\mathbf{q}}\mathbf{R}) = F_\alpha(T_{\alpha\mathbf{p}}T_{\beta\mathbf{q}}R) - F_\alpha(T_{\beta\mathbf{q}}R) \quad (10)$$

wobei

$$F_\alpha(R) = \sum_j f_{\alpha j}(R) \quad (11)$$

Der Wert  $F_\alpha(R)$  ist die Summe aller Kosten zwischen Knoten  $\alpha$  und allen anderen Knoten. Die Berechnung dauert  $\mathcal{O}(n)$ . Dieser Fall tritt genau  $(n-1)$  mal auf, immer dann wenn ein Knoten  $\alpha$  auf die alte Position von  $\beta$  verschoben wird.

Nachdem die  $\Delta$ -Matrix aktualisiert wurde, lässt sich erneut durch das minimale Element der Matrix die beste kleinste Änderung bestimmen und die Matrix muss wieder aktualisiert werden. Wenn alle Werte der  $\Delta$ -Matrix positiv sind, gibt es keine kleinste Änderung mehr, die die Kosten des Layouts verbessern könnte.

## 5.7 Auswahl der Parameter

Der Grid-Layout-Algorithmus benutzt einige Parameter, die für die Generierung eines geeigneten Layouts benötigt werden. Das Problem ist, dass diese Parameter von den Autoren für ihr konkretes Beispiel, das in Kapitel 5.8 vorgestellt wird, empirisch bestimmt wurden. Für andere biologische Netzwerke müssten die Parameter ggf. erneut ermittelt werden, um eine gute Darstellung zu erreichen.

Durch den Layoutbereich  $L$  wird die Anzahl  $|L| = m$  der möglichen Positionen der Knoten definiert. Ein kleiner Bereich führt zu einer schnellen Berechnung, aber auch zu einer Limitierung des Layouts. Möglichweise wird so keine geeignete Darstellung des biologischen Netzwerkes erreicht. Die Autoren haben diesen Parameter empirisch ermittelt. Der Layoutbereich sollte rechteckig sein und ungefähr drei bis sechs mal so viele Punkte wie Knoten enthalten. In ihrem Beispiel haben die Autoren folgenden Wert für die maximale Ausdehnung in  $x$ - und  $y$ -Richtung benutzt:  $x_{max} = y_{max} = 2\sqrt{n}$ .

Zur Berechnung der Nachbarlayouts wird eine Änderungsrate  $p$  benötigt. Diese wird auf  $p = 0.55$  gesetzt, d.h. alle Knoten werden mit einer Wahrscheinlichkeit von 55% verschoben.

Die maximale Abstoßungsdistanz, die zur Berechnung der Kosten zwischen zwei Knoten benötigt wird, ist  $d_{max} = 5$ .

Außerdem müssen die Parameter der simulierten Abkühlung passend initialisiert werden. Die Temperaturwerte werden wie folgt initialisiert:  $T_{min} = 0.1$  und  $T_{max} = (f_{max}^{(10)} - f_{min}^{(10)})/2$ , dabei sind  $f_{min}^{(10)}$  bzw.  $f_{max}^{(10)}$  der minimale bzw. maximale Kostenwert von 10 zufälligen Layouts.

Bei gleicher Temperatur  $T$  werden  $n_e = 10$  Wiederholungen durchgeführt und der Abkühlungsfaktor nach jeder Iteration ist  $r_c = 0.8$ .

## 5.8 Vergleich anhand eines Beispiels

Li und Kurata benutzen das *yeast cell-cycle* Netzwerk (Kurata et al., 2003) mit 200 Knoten, um den Vorteil Ihres Grid-Layouts gegenüber einem force-directed Layout zu demonstrieren. Leider sagen sie nicht, welche Implementierung des force-directed Layouts sie bei Ihrem Vergleich benutzt haben.

In Abbildung 5 sieht man ein force-directed Layout angewandt auf das yeast cell-cycle Netzwerk. Die funktionalen Einheiten wurden von den Autoren nachträglich eingefärbt, um so die Sichtbarkeit hervorzuheben. Sie kritisieren an dieser Darstellung vor allem, dass einige Knoten zu nah beieinander liegen und dadurch überlappende Symbolen entstehen.

In Abbildung 6 ist das yeast cell-cycle Netzwerk mit dem Algorithmus der Autoren gelayoutet worden. Sie sagen, dass das Layout kompakter ist und die funktionalen Einheiten besser erkennbar sind. Dies sei aus Sicht der Autoren das Wichtigste, wenn man komplexe biologische Netzwerke darstellen will. Aufgrund des Rasters auf dem die Knoten platziert worden sind, gibt es keine Überlappungen der Symbole. Jedoch sieht man einige sehr lange Kanten, die quer durch das Netzwerk verlaufen und sowohl andere Kanten als auch Knoten kreuzen. Diese empfinde ich als sehr störend.

In Abbildung 7 habe ich einen Ausschnitt des Netzwerkes vergrößert. Dieser stellt eine funktionale Einheit dar. An diesem Ausschnitt kann man deutlich die Verbesserung des Layouts bezüglich der Knotenüberlappungen sehen. Allerdings sieht man auch die störenden langen Kanten, die diagonal

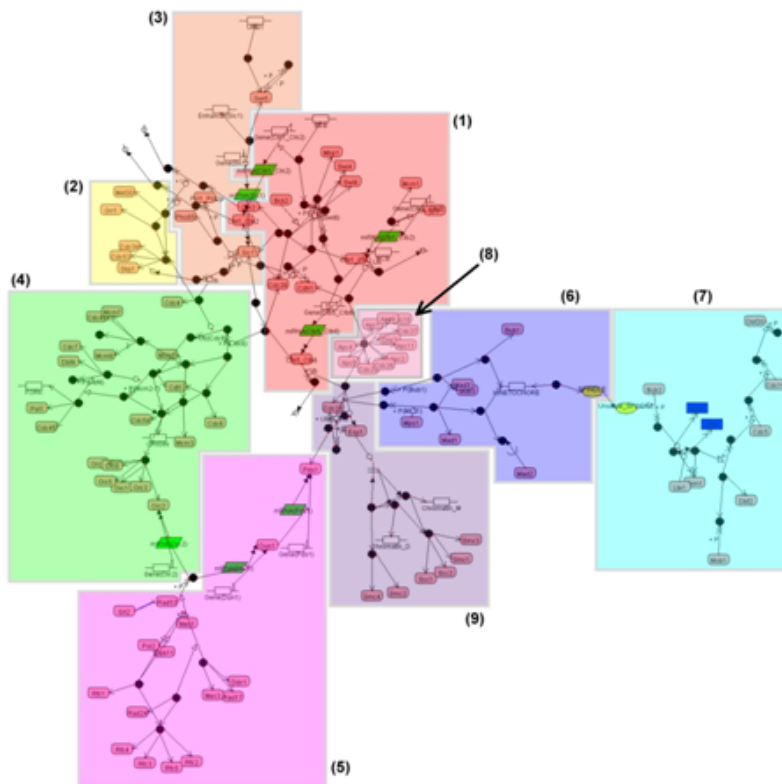


Abbildung 5: force-directed Layout des yeast cell-cycle Netzwerks. (Quelle: *Li, Weijiang und Kurata, Hiroyuki*)

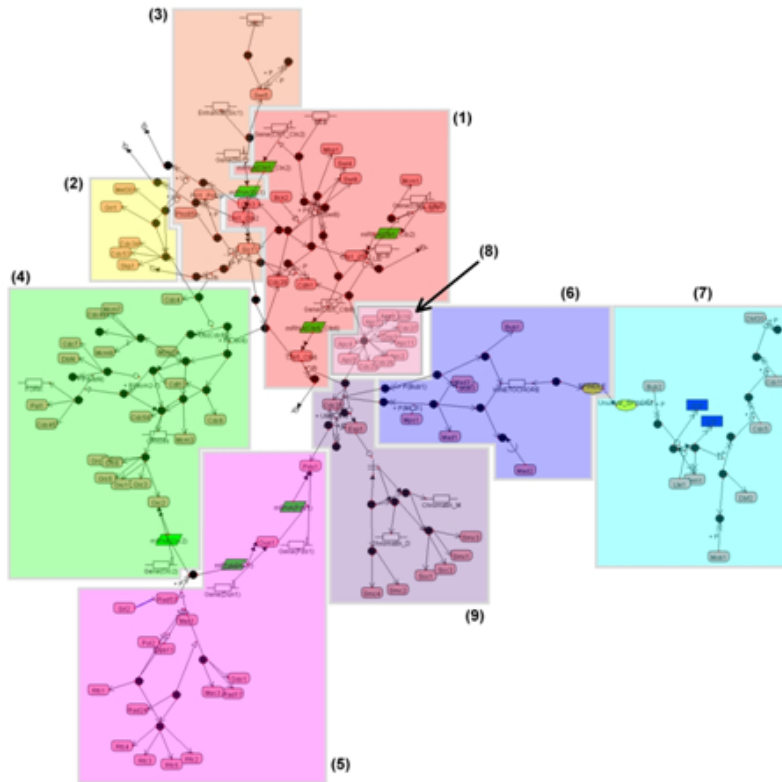


Abbildung 6: Grid-Layout des yeast cell-cycle Netzwerks. (Quelle: *Li, Weijiang und Kurata, Hiroyuki*)

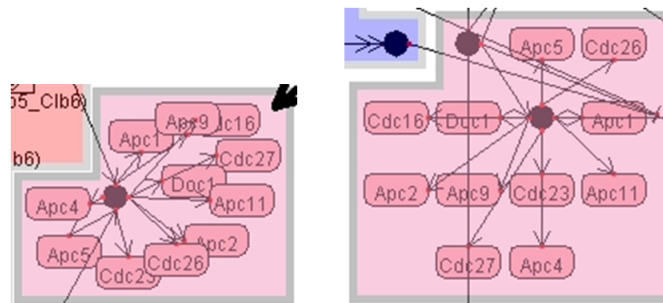


Abbildung 7: Vergleich einer funktionalen Einheit in beiden Layouts. (Quelle: *Li, Weijiang und Kurata, Hiroyuki*)

durch die funktionale Einheit verlaufen.

## 5.9 Laufzeit

Laut den Autoren kann ein Grid-Layout für ein biologischen Netzwerk mit maximal einigen hundert Knoten in akzeptabler Zeit berechnet werden. Bei weniger als 50 Knoten dauert die Berechnung allerdings schon wenige Minuten. Ein force-directed Layout kann für 50 Knoten in deutlich kürzerer Zeit berechnet werden.

Um das im Kapitel 5.8 vorgestellte yeast cell-cycle Netzwerk zu layouten, haben die Autoren mit einem Pentium IV mit einer Taktfrequenz von 1.7 Ghz ungefähr 10 Minuten benötigt. Außerdem haben sie eine Implementierung ihres Algorithmus ins Netz gestellt, um selbst biologische Netzwerke für das am Anfang bereits erwähnte Tool CADLIVE zu layouten. Mit dieser Implementierung habe ich für das yeast cell cycle network mit einem Core2Duo T7100 mit einer Taktfrequenz von je 1.8 Ghz ungefähr 7 Minuten gebraucht, um das Grid-Layout berechnen zu lassen.

Der Speicherplatzbedarf für das Grid-Layout liegt bei  $\mathcal{O}(nm)$ . Dieser Platz wird benötigt, um die  $\Delta$ -Matrizen zu speichern. Dies stellt für Rechner der heutigen Generation kein Problem mehr dar.

Die Autoren versprechen am Ende ihrer Arbeit weitere Verbesserungen, allerdings ist auf ihrer Website bisher keine weitere Veröffentlichung zu diesem Thema erschienen.

## 6 Fazit

Li und Kurata wollten einen Layoutalgorithmus entwickelt, der vor allem die funktionalen Einheiten eines komplexen biologischen Netzwerkes besser darstellt, als bisher verfügbare Layouts. Bei dem als Beispiel angeführten yeast cell-cycle Netzwerk ist es ihnen mit der Entwicklung des Grid-Layout-Algorithmus gelungen. Es bleibt jedoch offen, wie deutlich die Verbesserungen bei anderen realen biologischen Netzwerken sind.

Das Grid-Layout stellt biologische Netzwerke kompakt dar und vermeidet Knotenüberlappungen. Eng-verknüpfte Knoten werden beim Grid-Layout nahbeieinander platziert, so dass Cluster gut zu erkennen sind.

Dabei sollte man allerdings beachten, dass die Qualität der Darstellung von einigen Parametern abhängt, die möglicherweise für jedes einzelne biologische Netzwerk empirisch bestimmt werden müssen. Die Kompaktheit des Layouts ist abhängig von der bereitgestellten Layoutfläche. Knotenüberlappungen können nur verhindert werden, wenn das Raster, auf dem die Knoten platziert werden, zu der Größe der Knotensymbole passt.

Die Vermeidung von Kantenüberlappungen und eine Minimierung der Kantenlänge wird in ihrem Algorithmus jedoch nicht extra beachtet, so dass es zu störenden langen Kanten kommen kann, die diagonal durch das Layout und durch andere Kanten verlaufen.

Ein weiterer problematischer Punkt ist die Berechnungsdauer für das Layout eines Netzwerkes. Die Autoren sehen zu Beginn das Anwendungsgebiet ihres Algorithmus bei der Darstellung von sehr komplexen biologischen Netzwerken. Offen bleibt jedoch wieviele Knoten komplexe biologische Netzwerke im „schlimmsten“ Fall haben können, denn bereits für einige hundert Knoten muss man ein paar Minuten auf das Layout warten. Eventuell muss man ein Layout auch mehrmals berechnen lassen, um die oben angeführten Parameter zu bestimmen, damit man eine optimierte Darstellung des Netzwerkes erhält.

Die Laufzeit führt auch zu einem weiteren Punkt. Das Grid-Layout ist nicht für interaktive Anwendungen geeignet, da bei einer Veränderung des dargestellten Netzwerkes, z.B. der Ausblendung von funktionalen Einheiten, das Layout komplett neu berechnet werden muss. Die Wartezeit auf eine Aktualisierung der Darstellung wäre zu lange, um effektiv damit zu arbeiten. Für die statische Darstellungen eines biologischen Netzwerkes ist das Grid Layout jedoch geeignet.

## 7 Literaturverzeichnis

- Li, Weijiang und Kurata, Hiroyuki. *A grid layout algorithm for automatic drawing of biochemical networks*. 2005
- Brandes, Ulrik. *Layout of Graph Visualizations*. 1999, iii
- R. Davidson, D. Harel. *Drawing graphs nicely using simulated annealing*. ACM Transactions on Graphics, Volume 15(4):301–331, 1996
- Wikipedia.de. *Simulated Annealing*.