# Praktikum im WS 05/06
# Graphengeneratoren
# Ein Generator für Smallworld-Graphen

Authoren: Bernd Ahues und Karsten Brand
Betreuer: Thomas Schank

14. März 2006

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Logik, Komplexität und Deduktionssysteme

**Zusammenfassung**

Wir befassen uns im Folgenden mit der Erstellung eines Generators für Smallworld-Graphen und der anschließenden Darstellung des generierten Graphens.
Ziel des Verfahrens ist die ersichtliche Darstellung des Werdegangs des Graphen in einer Animation. Auf eine Betrachtung der charakteristischen Pfadlänge und des Vernetzungsgrades verzichten wir.

Unser Verfahren nutzt eine abgewandelte Version des von Duncan J. Watts und Steven H. Strogatz [Str98] vorgestellten Algorithmus zur Erzeugung von Smallworld-Graphen. Zur Visualisierung nutzen wir yFiles [yWo] und Java3D [jav].

## Einführung

In dieser Arbeit stellen wir ein Verfahren vor, welches Smallworld-Graphen generiert. Weiter haben wir eine Visualisierung implemen-

tiert, welche die Struktur von Smallworld-Graphen in einer Animation grafisch verdeutlichen soll.

## Definitionen

Ein *Graph* $G$ ist ein geordnetes Paar von Mengen $G = (V, E)$. Dabei bezeichnet V die Menge der im Graph enthaltenen Knoten und E die Menge der Kanten in G. Kanten verbinden jeweils zwei Knoten, wobei wir hier zwischen Kanten $(v_1, v_2)$ und Kanten $(v_2, v_1)$ nicht unterscheiden wollen. Die Anzahl $|V|$ der Knoten im Graph wollen wir im Weiteren mit $n$ bezeichnen. Die *charakteristische Pfadlänge* eines Graphen G bestimmt man, indem man die mittlere Länge der kürzesten Pfade zwischen zwei Knoten im Graph bestimmt.

Der *Clustering Coefficient* $C$ eines Graphen berechnet sich als Mittelwert aus dem lokalen Vernetzungsgrad $C_v$ jedes Knotens $v$ in $G$. $C_v$ ist der Quotient aus der Anzahl der Kanten, die zwischen $v$'s Nachbarn $\Gamma_v$ tatsächlich verlaufen und der maximalen Anzahl Kanten die zwischen den Nachbarn von $v$ verlaufen könnten.

$$C = \frac{1}{n} \sum_{v \in V} C_v$$

$$C_v = \frac{|E(\Gamma_v)|}{\binom{k}{2}}$$

Ein *Smallworld-Graph* ist ein Graph mit $n$ Knoten und $k$ Kanten pro Knoten, der sich durch eine kurze charakteristische Pfadlänge und einen hohem Vernetzungsgrad auszeichnet. Diese Eigenschaften ermöglichen es, von einem beliebigen Knoten im Graphen in sehr wenigen Schritten alle anderen Knoten des Graphen zu erreichen. Beispiele in der Realität lassen sich sowohl in Sozial- wie auch in Computernetzwerken finden. Insbesondere wurde das Smallworld-Phänomen durch den Versuch des Psychologen Stanley Milgram [Mil67] (*Six degrees of separation*) bekannt und später auch verfilmt [Sch93].

Sei $A$ eine Matrix, $\lambda$ ein *Eigenwert* von A und $x$ der zu $\lambda$ gehörende *Eigenvektor*. Dann gilt: $A \cdot x = \lambda \cdot x$
Ein Eigenvektor ist also ein Vektor, der durch die Matrix $A$ um ein $\lambda$-faches gestreckt oder gestaucht wird. Die *Laplace Matrix* ist eine quadratische Matrix, die für jeden gewichteten Graph $G$ definiert wird als:

$$l_{v,w} = \begin{cases} \sum_{u \in V} \omega(u, v) & ,v = w \\ -\omega(v, w) & ,v \neq w \end{cases}$$

wobei $\omega$ wie folgt definiert ist:

$$\omega = \begin{cases} 1 & ,\{v,w\} \in E \\ 0 & ,\{v,w\} \notin E \end{cases}$$

Nach Definiton ist diese Matrix symmetrisch und hat damit reelle Eigenwerte. Weiter sieht man leicht, dass sie den Eigenvektor 1 mit zugehörigem Eigenwert 0 besitzt.

## Potenziteration

Die *Potenziteration* ist ein numerisches Verfahren zur Berechnung des betragsgrößten Eigenwertes einer Matrix und des dazugehörigen normierten Eigenvektors. Ausgehend von einem Startvektor $r_0$ und einer quadratischen Matrix A geschieht dabei folgendes: Die Iteration

**for** *int i=0 To k* **do**
  $q_i = r_{i-1}/||r_{i-1}||$
  $r_i = Aq_i$
**end**

wird solange ausgeführt bis $q_k$ gegen einen normierten Eigenvektor konvergiert. Die Potenziteration ist bezüglich Konvergenzgeschwindigkeit Verfahren wie der QR- Zerlegung [SW92] unterlegen. Für unsere Zwecke ist sie aber ausreichend, da sie relativ einfach zu implementieren ist und bei geeigneter Wahl von $r_0$ auch schnell konvergiert.

## Spektrales Layout

Einen Vektor $p_v$ aus Koordinaten $p_v = (x_{1v}, x_{2v}, \cdots, x_{nv})$ nennt man n- dimensionales *Layout*.

Nimmt man für die Vektoren $x_{iv}$ normierte Eigenvektoren einer graph-verwandten Matrix, so spricht man von einem *spektralen Layout* [P]. Gehören diese Eigenvektoren zu kleinen Eigenwerten, so minimiert dieses Layout den Abstand $\epsilon$ zwischen verbundenen Knoten,

$$\epsilon = \sum_{e=(i,j)\in E} (x_i - x_j)^2$$

was bei Graphen im Allgemeinen eine gewünschte Eigenschaft ist um die Übersichtlichkeit zu erhöhen [Wil05].

# Verwendete Algorithmen

## Algorithmen zur Erzeugung des Graphen

Wir verwenden den Algorithmus nach *Duncan J. Watts und Steven H. Strogatz* [Str98] In diesem Algorithmus wird ausgehend von einem festgelegten Graphen durch schrittweises Umbiegen von Kanten ein Smallworld-Graph erzeugt. Für einen Graphen mit $n$ Knoten und $k$ von jedem Knoten ausgehende Kanten wird der Ausgangsgraph erstellt, indem man die $n$ Knoten ringförmig anordnet und sie zu ihren $k$ nächsten Nachbarn im Uhrzeigersinn verbindet.
Nachdem der regelmäßige Ausgangsgraph erstellt ist, wählt man einen Knoten und die Kante, die ihn mit seinem nächsten Nachbarn im Uhrzeigersinn verbindet. Mit einer Wahrscheinlichkeit $p$ biegt man die Kante auf einen zufällig ausgewählten Knoten, mit dem der derzeitige Knoten noch keine Verbindung hat, wobei auch reflexive Kanten ausgeschlossen sind. Dieser Schritt wird für jeden Knoten im Graphen wiederholt, danach betrachtet man die Kanten, die entferntere Knoten verbinden.
Je nach Wahl von $p$ können hierbei Graphen entstehen, die dem Ausgangsgraphen nahezu gleichen (für sehr kleine $p$), oder komplette Zufallsgraphen (für große $p$). Es bleibt noch anzumerken, dass dieses Verfahren keine Verifikation besitzt, ob der resultierende Graph wirklich ein Smallworld-Graph ist. Bei kleinen $p$ ist die Wahrscheinlichkeit, einen Smallworld-Graphen zu erzeugen, am höchsten [Str98].

```
int i=0;
while i < k do
    foreach Knoten K In Graph do
        if p > Zufall() then
            //Biege die i-te Kante des Knoten auf einen Zufallsknoten
            um BiegeKante(K, K.Kante(i), ZufallsKnoten());
        end
    end
    i++;
end
```

**Algorithm 1**: Algorithmus nach Watts und Strogatz

### Unsere Modifikation

Da die reihenweise Betrachtung der Kanten zum Nachteil hat, dass sich die Änderungen in der Graphenstruktur nur langsam vom Startknoten ausbreiten und man – gerade beim *Spektral Layout* in der Animation –

die gerade betrachteten Kanten schlecht ausfindig machen kann, haben wir den Algorithmus modifiziert.

Anstatt das Augenmerk auf die Kanten zu legen, betrachten wir erst alle ausgehenden Kanten eines Knotens und biegen diese entsprechend dem ursprünglichen Algorithmus um. Wurden alle Kanten des Knotens betrachtet, so wählen wir einen zufälligen − noch nicht betrachteten − Knoten aus dem Graphen aus und betrachten nun dessen Kanten. Der Algorithmus endet, wenn alle Knoten und ihre Kanten betrachtet wurden. Die Modifikation ist zwar gering, doch lässt sich in der späteren Animation die Transformation zum Smallworld-Graphen weit besser beobachten.

```
while noch nicht alle Knoten markiert do
    finde zufällig unmarkierten Knoten K
    for i=0 To k do
        if p > Zufall() then
            // Biege die i-te ausgehende Kante von K auf einen
            // Zufallsknoten ohne Verbindung zu K
            BiegeKante(K, K.Kante(i), ZufallsKnoten());
        end
        markiere Knoten
    end
end
```

**Algorithm 2**: Modifizierter Algorithmus

## Algorithmus zur Erzeugung eines spektralen Layouts

Da die Potenziteration nur den größten Eigenwert und zugehörigen Eigenvektor liefert, wir für das spektrale Layout aber die kleinsten Eigenvektoren benötigen, verwenden wir für die Iteration $L_g = g \cdot I - L$ als graph-verwandte Matrix. Diese besitzt die gleichen Eigenvektoren wie $L$, jedoch in umgekehrter Reihenfolge, wenn man $g$ größer oder gleich dem größtem Eigenwert von $L$ wählt.
Um mehrere Eigenvektoren zu gewinnen, bedarf es einer weiteren leichten Modifikation:

In jedem Iterationsschritt orthogonalisieren wir den Vektor $r_k$ mit allen bisher gewonnenen Eigenvektoren und dem trivialen Eigenvektor $1 = (1, 1, \cdots, 1)^T$. Dadurch wird gewährleistet, dass alle Vektoren, die für das spektrale Layout verwendet werden, orthogonal sind.

Wir generieren nach jeder Änderung im Graphen ein neues Spektral Layout und erhalten so eine Folge von Spektral-Layouts. Diese Folge wollen wir *dynamisches Spektral Layout* nennen.

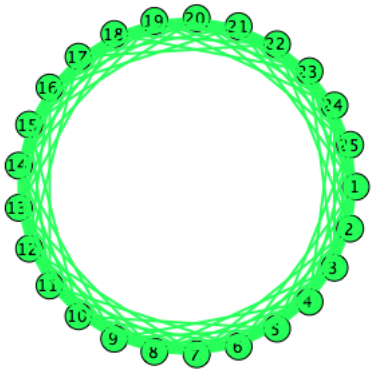
# Darstellung des Graphen

Zuerst haben wir eine zweidimensionale Darstellung gewählt, da das von uns verwendete yFiles bereits eine solche anbietet und wir yFiles zur Implementierung unseres Algorithmuses nutzen.

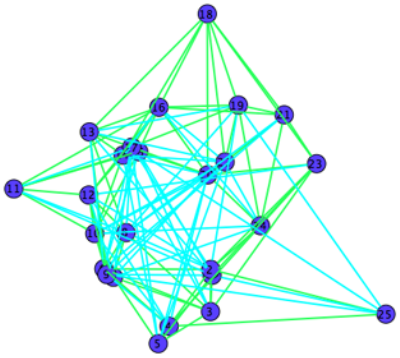
Nach erfolgreicher Implementierung hat es sich gezeigt, dass die zweidimensionale Darstellung zwar gut für feste Layouts ist, jedoch für das dynamische spektralen Layout unzureichend in der Darstellung ist, da hier Knoten auch übereinander liegen können. Aufgrund dessen wählen wir zur bessern Darstellung des Graphen eine dreidimensionale Darstellung mittels Java3D.

## 2D-Darstellung

Wir stellen zur klareren Einsicht in die Schritte auf dem Weg zum Smallworld- Graphen die gerade betrachteten Knoten und Kanten rot dar. Kanten, die umgebogen wurden, werden blau gefärbt, die die nicht umgebogen wurden, dunkelgrün. Abgearbeitete Knoten werden blau markiert.



$n = 25, k = 5$



$p = 0.4$

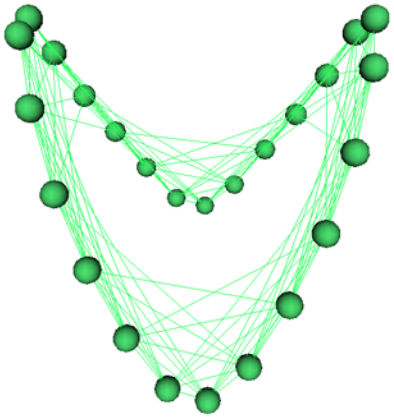
### 2D-Animation

yFiles bietet zur Animation von Graphen ein *morph-Objekt*, das durch Interpolation eine Animation aus zwei Graphenlayouts erstellt. In unserem Fall sind das das Layout des Graphen vor einem Transformationsschritt und nach einem Transformationsschritt.

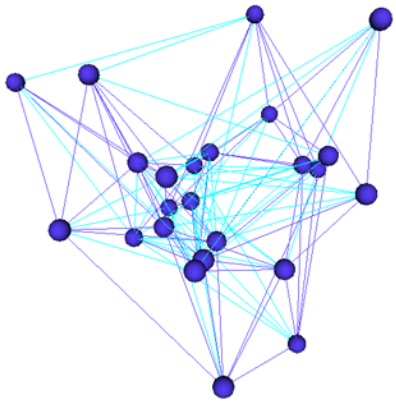
## 3D-Darstellung

Wie in der zweidimensionalen Darstellung stellen wir hier auch den Fortschritt des Algorithmus im Graphen farbig dar. Nur werden hier

umgebogene Kanten hellblau, nicht umgebogene dunkelblau dargestellt.



$n = 25, k = 5$



$p = 0.4$

### 3D-Animation

Die in Java3D angeboten Animationen lassen sich leider nicht mit unseren Vorstellungen bezüglich der Animation vereinbaren, so dass wir eine eigene Animationsalgorithmus geschrieben haben. Dieser errechnet für jeden der Knoten den Vektor zwischen ihrer alten und neuen Position und unterteilt diese in die Anzahl der gewünschten Einzelbilder.
Durch aufeinanderfolgendes Vorranschreiten auf den unterteilen Vektoren entsteht dann die Animation.

## Literatur


[jav] java3d. Java3d. http://java3d.dev.java.net.

[Mil67] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[P] Ulrik Brandes & Daniel Fleischer & and Thomas Puppe. Dynamic Spectral Layout of Small Worlds. Departmant of Computer & Information Science, University of Konstanz, Germany.

[Sch93] Fred Schepisi. Six degrees of seperation. Film, 1993.

[Str98] Duncan J. Watts & Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[SW92] Robert Schaback and Helmut Werner. *Numerische Mathematik*. Springer, 4., vollst. überarb. aufl. edition, 1992.

[Wil05] Thomas Willhalm. *Engineering Shortest Paths and Layout Algorithms for Large Graphs*. Universität Karlsruhe, Fak. f. Informatik. Diss. v. 16.02.2005., 2005.

[yWo] yWorks. yfiles. http://www.yworks.com.