Spring Embedder

Thomas Friedrich
25. Januar 1999

Seminar über Automatisches Zeichnen von Graphen
Universität zu Köln

1 Einführung

1.1 Zeichnen von Graphen

Graphen beschreiben Strukturen und Zusammenhänge von Objekten. Sie kommen in verschiedenen Bereichen vor; Beipiele sind Netzwerke, Diagramme, Flußgraphen und Schaltpläne.

Um Graphen visuell darzustellen, werden sie meist in der Ebene gezeichnet. Da nun Graphen häufig mit Computern verarbeitet werden, liegt es nahe, diese Zeichnung automatisch erstellen zu lassen.

Wir werden zwei Algorithmen kennenlernen, die diese Aufgabe übernehmen. Dabei betrachten wir allgemeine ungerichtete ungewichtete Graphen $G = (V, E)$ mit $|V| = n$; o. B. d. A. sei $V = \{1, \ldots, n\}$. Die Knotenpositionen sind frei, d. h. sie unterliegen keinen Restriktionen. Kanten werden als gerade Linien zwischen den Knoten dargestellt.

Außerdem können wir uns auf zusammenhängende Graphen beschränken. Bei nicht-zusammenhängenden Graphen zeichnen wir die Zusammenhangskomponenten einzeln. Diese können (z. B. mittels *Depth First Search*, DFS) in Zeit $O(|V| + |E|)$, also linear in der Eingabelänge, bestimmt werden.

Das Ergebnis der Zeichenalgorithmen sollte natürlich gut aussehen.

Was ist „schön"?

Was als schön, ästhetisch ansprechend oder gelungen bezeichnet wird, ist sehr subjektiv und läßt sich nicht in eine allgemeine Definition fassen. Einige Kriterien für gutaussehende Zeichnungen von Graphen sind:

1. möglichst wenige Kantenüberkreuzungen
2. Symmetrien des Graphen sind leicht erkennbar
3. benachbarte Knoten liegen nahe beieinander
4. Knoten liegen aber nicht *zu* dicht zusammen
5. einheitliche Kantenlängen

Dabei muß beachtet werden, daß nicht alle diese Kriterien gleichzeitig erfüllt werden können, ja sich einige sogar gegenseitig ausschließen. Ein Beispiel hierfür zeigt die folgende Abbildung:

Beide Bilder sind Zeichnungen desselben Graphen. Wer nicht unbedingt an einer planaren Zeichnung interessiert ist, wird die linke Zeichnung als besser empfinden, da sie die Symmetrie des Graphen besser zeigt, obwohl sie 5 Kantenkreuzungen enthält. Daß der Graph planar ist, zeigt die rechte Zeichnung. Dafür ist hier die symmetrische Grundstruktur nicht so einfach auszumachen.

1.2 Physikalische Modelle

Den ersten Ansatz, physikalische Modelle als Grundlage zum Zeichnen von Graphen zu verwenden, hatte Peter Eades 1984. Die Idee dahinter ist, physikalische Zusammenhänge auf Graphen zu übertragen und in dem entstehenden Kräftespiel die Gesamtenergie des Systems möglichst klein werden zu lassen.

Die in diesem Vortrag vorgestellten Algorithmen verwenden zwei verschiedene Möglichkeiten, die Natur als Vorbild zu nehmen. Zum einen ist da der *Spring Embedder*, der auf einem mechanischen System aus beweglichen Ringen und zwischen den Ringen gespannten Federn (engl. *spring*=Feder) basiert. Die Ringe werden dabei von den Federn in bestimmte Positionen gezogen.

Zum anderen werden wir es mit dem *Simulated Annealing* zu tun haben; hierbei betrachtet man das Modell atomarer Teilchenbewegungen in einer sich abkühlenden Masse (engl. *annealing*=Abkühlen). Die Kernteilchen üben anziehende (Gravitations-)Kräfte und abstoßende (elektrische Ladungs-)Kräfte aufeinander aus.
Nach Aussage seiner Autoren handelt es sich bei diesem Algorithmus allerdings eigentlich um eine modifizierte Variante eines Spring Embedders.

2 Der Algorithmus von Kamada und Kawai

2.1 Federmechanik-Modell: Spring Embedder

Der Spring Embedder-Algorithmus von Kamada und Kawai modelliert ein physikalisches System von Eisenringen und Stahlfedern.

Wird eine Feder aus ihrer entspannten Ruhelage heraus auseinandergezogen, so wirkt sie dieser Auslenkung mit einer rückstellenden Kraft entgegen. Wird sie aber zusammengestaucht, versucht sie, wieder zu expandieren.

In jedem Fall übt sie also eine zu ihrer Ausgangsposition gerichtete Kraft aus. Dabei ist die Größe dieser Rückstellkraft proportional zur Auslenkung. Der Faktor, der dabei eine Rolle spielt, ist die individuelle Härte der Feder, die sogenannte *Federkonstante*. Sie kann für jede Feder experimentell ermittelt werden.

Jede Feder ist nur begrenzt elastisch; wird sie einmal über eine bestimmte Grenze hinaus deformiert, kehrt sie nicht mehr in ihre ursprüngliche Ausgangslage zurück. Dieser Aspekt wird aber in dem betrachteten Modell vernachlässigt.

2.2 Konzept und Ziele

Die Knoten des Graphen sollen die Eisenringe darstellen. Zwischen je zwei Knoten (also nicht nur bei durch eine Kante verbundenen Knoten) wird nun eine Stahlfeder gespannt.

Sei d_{ij} der Abstand von Knoten i zu Knoten j im Graphen; das ist die Länge eines kürzesten Weges von i nach j.

Dieser (graphentheoretische) Abstand von Knoten im Graphen wird nun direkt mit dem (euklidischen) Abstand der die Knoten repräsentierenden Punkte in der Ebene verglichen. Die Summe der Fehlerquadrate sollte dabei möglichst klein werden.

Dies entspricht der Minimierung der Gesamtenergie des Systems, in dem die Stahlfedern an den Eisenringen ziehen.

In der resultierenden Zeichnung werden Knoten nicht allzu dicht gedrängt liegen, da sie durch die Federn abstoßende Kräfte aufeinander ausüben.

Ist der zu zeichnende Graph symmetrisch, so wirken auch die Kräfte im Federmodell symmetrisch, und wir können eine symmetrische Zeichnung erwarten.

2.3 Minimierung der Energie

Die Ausgangslänge l_{ij} der Feder, die Knoten i mit Knoten j verbindet, ist im wesentlichen proportional zu d_{ij} und sei wie folgt festgelegt:

$$l_{ij} = L \cdot d_{ij}.$$

Dabei bezeichne L einen Richtwert für die Länge einer Kante. Er berechnet sich aus der Seitenlänge L_0 der (quadratisch angenommenen) Zeichenfläche und dem größten Abstand zweier Knoten (dem *Diameter*) im Graphen:

$$L = \frac{L_0}{\max\limits_{i<j} d_{ij}}.$$

Die Härte k_{ij} der Feder zwischen i und j wird auf

$$k_{ij} = \frac{K}{d_{ij}^2}$$

gesetzt; dies ist physikalisch motiviert. Die Konstante K kann in unserem Modell frei gewählt werden, z. B. gleich 1.

Wenn p_i den Punkt in der Ebene bezeichnet, der Knoten i repräsentiert, beträgt die Gesamtenergie des Systems also

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} \cdot k_{ij} \cdot (||p_i - p_j||_2 - l_{ij})^2,$$

wobei $l_{ij} = L \cdot d_{ij}$ mit $L = \frac{L_0}{\max_{i<j} d_{ij}}$ und $k_{ij} = \frac{K}{d_{ij}^2}$ mit einer Konstanten K sind.
Setzt man nun die Koordinaten ein, also $p_i = (x_i, y_i)$, so ergibt sich

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} \cdot k_{ij} \cdot \left((x_i - x_j)^2 + (y_i - y_j)^2 + l_{ij}^2 - 2 \cdot l_{ij} \cdot \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right).$$

Hiervon wollen wir jetzt ein Minimum bestimmen.

Die Bestimmung eines globalen Minimums einer nichtlinearen Funktion in $2n$ Variablen ist i. a. eine komplizierte Aufgabe. Alle ernstzunehmenden Verfahren beruhen darauf, daß sie die lokalen Minima aufzählen.

Wir werden uns also mit der Suche nach einem lokalen Minimum begnügen. Für einen lokalen Extremwert müssen alle partiellen Ableitungen Null sein, also

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = 0, \quad 1 \leq m \leq n.$$

Sie lauten:

$$\frac{\partial E}{\partial x_m} = \sum_{i \neq m} k_{mi} \cdot \left((x_m - x_i) - \frac{l_{mi} \cdot (x_m - x_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}} \right) \stackrel{!}{=} 0$$

$$\frac{\partial E}{\partial y_m} = \sum_{i \neq m} k_{mi} \cdot \left((y_m - y_i) - \frac{l_{mi} \cdot (y_m - y_i)}{\sqrt{(x_m - x_i)^2 + (y_m - y_i)^2}} \right) \stackrel{!}{=} 0.$$

Diese $2n$ nichtlinearen Gleichungen müssen jetzt simultan gelöst werden. Da sie voneinaner abhängig sind, ist dies sehr schwierig.

Deshalb werden wir anders vorgehen.

2.4 Das Newton-Raphson-Verfahren

Wir halten alle Punkte bis auf einen fest und bringen lediglich die Position dieses einen freien Punktes in eine energieminimale Lage, indem wir das *Newton-Raphson-Verfahren* anwenden.

Dabei wählen wir denjenigen Punkt aus, der den größten Fortschritt zur Erreichung unseres Zieles der Energieminimierung verspricht. Das ist also der Punkt (x_m, y_m), bei dem die Norm des Gradienten, also

$$\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2}$$

am größten ist. Ist dieses Maximum gleich (oder sehr nahe bei) Null, so sind wir fertig.

Das Newton-Raphson-Verfahren ist ein Iterationsverfahren zur Bestimmung von Nullstellen von Funktionen. Wir wenden es hier 2-dimensional auf die partiellen Ableitungen an. Dabei werden wir nicht unbedingt genau den Wert

Null erhalten; ein (vorher festgelegter) genügend kleiner Wert ε sollte uns reichen.

Ausgehend von $(x_m^{(0)}, y_m^{(0)})$ lautet die Iterationsvorschrift:

$$x_m^{(t+1)} = x_m^{(t)} + \delta_x, \quad y_m^{(t+1)} = y_m^{(t)} + \delta_y$$

für $t = 1, 2, \ldots$. Die Werte für δ_x und δ_y werden aus dem folgenden linearen Gleichungssystem bestimmt:

$$\frac{\partial^2 E}{\partial x_m^2}(x_m^{(t)}, y_m^{(t)}) \cdot \delta_x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^{(t)}, y_m^{(t)}) \cdot \delta_y = -\frac{\partial E}{\partial x_m}(x_m^{(t)}, y_m^{(t)})$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m}(x_m^{(t)}, y_m^{(t)}) \cdot \delta_x + \frac{\partial^2 E}{\partial y_m^2}(x_m^{(t)}, y_m^{(t)}) \cdot \delta_y = -\frac{\partial E}{\partial y_m}(x_m^{(t)}, y_m^{(t)}),$$

wobei die Koeffizienten die Einträge der Jacobi-Matrix sind:

$$\frac{\partial^2 E}{\partial x_m^2} = \sum_{i \neq m} k_{mi} \cdot \left(1 - \frac{l_{mi} \cdot (y_m - y_i)^2}{\left((x_m - x_i)^2 + (y_m - y_i)^2\right)^{3/2}}\right)$$

$$\frac{\partial^2 E}{\partial x_m \partial y_m} = \sum_{i \neq m} k_{mi} \cdot \quad \frac{l_{mi} \cdot (x_m - x_i) \cdot (y_m - y_i)}{\left((x_m - x_i)^2 + (y_m - y_i)^2\right)^{3/2}}$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m} = \sum_{i \neq m} k_{mi} \cdot \quad \frac{l_{mi} \cdot (x_m - x_i) \cdot (y_m - y_i)}{\left((x_m - x_i)^2 + (y_m - y_i)^2\right)^{3/2}}$$

$$\frac{\partial^2 E}{\partial y_m^2} = \sum_{i \neq m} k_{mi} \cdot \left(1 - \frac{l_{mi} \cdot (x_m - x_i)^2}{\left((x_m - x_i)^2 + (y_m - y_i)^2\right)^{3/2}}\right)$$

Das Newton-Raphson-Verfahren bricht ab, wenn Δ_m bei $(x_m^{(t)}, y_m^{(t)})$ kleiner als ε wird.

2.5 Der Algorithmus

Wir können nun den Algorithmus formulieren; es fehlen lediglich noch die initialen Startpositionen für $p_1, \ldots, p_n$. Wir könnten die Punkte zufällig über der zur Verfügung stehenden Fläche verteilen. Die Anfangskonfiguration hat nur wenig Einfluß auf die Qualität des Endergebnisses, solange sie nicht entartet ist; dies ist z. B. der Fall, wenn alle Punkte auf einer Geraden liegen.

Stattdessen plazieren wir die Punkte jedoch auf den Ecken des regelmäßigen n-Ecks, dessen Durchmesser dem Diameter des Graphen entspricht.

Freie Parameter sind die Breite bzw. Höhe L_0 der (quadratischen) Zeichenfläche, die Konstante K, die die Federhärten beeinflußt, sowie der Wert von ε. Als Eingabe dient der zu zeichnende Graph.

Kurz zusammengefaßt lautet der Algorithmus also:

Algorithmus KK;
Berechne d_{ij} $(1 \leq i \neq j \leq n)$;
Berechne l_{ij} $(1 \leq i \neq j \leq n)$;
Berechne k_{ij} $(1 \leq i \neq j \leq n)$;
Initialisiere $p_1, \ldots, p_n$;
while $(\max_i \Delta_i > \varepsilon)$ {
    Sei m der Index mit $\Delta_m = \max_i \Delta_i$;
    while $(\Delta_m > \varepsilon)$ {
        Berechne δ_x und δ_y;
        $x_m := x_m + \delta_x$;
        $y_m := y_m + \delta_y$;
    }
}

Ausgabe sind die x- und y-Koordinaten der die Knoten repräsentierenden Punkte in der Ebene.

Wir betrachten die Anwendung des Algorithmus auf ein einfaches Beispiel (Abbildung nächste Seite). Ausgehend von den Eckpunkten des regelmäßigen 6-Ecks (a) wird zunächst der Knoten B in eine stabile Position gebracht (b).

Danach wirkt auf Knoten F die größte Spannung; er ist der nächste Kandidat, der versetzt wird (c). Hier ist bereits eine geordnete Struktur des Graphen erkennbar.

Nach 21 Iterationen ist der endgültige Zustand erreicht (d).

.

2.6 Laufzeit-Analyse

Die Berechnung der d_{ij} erfordert Zeit $O(n^3)$ bei Verwendung von Dijkstra's Algorithmus (besonders für große Graphen: es sind auch Algorithmen mit mittlerer Laufzeit $O(n^2 \cdot \log^2 n)$ bzw. $O(n^2 \cdot \log n)$ bekannt). Die l_{ij} und k_{ij} können in $O(n^2)$ berechnet werden; die Initialisierung der p_i läuft in $O(n)$.

Die meiste Laufzeit wird allerdings von den geschachtelten *while*-Schleifen verbraucht. In der inneren Schleife (also beim Newton-Raphson-Verfahren) können Δ_m, sowie δ_x und δ_y, in jeder Iteration in $O(n)$ berechnet werden. Die äußere Schleife benötigt nur Zeit $O(n)$ zur Bestimmung von $\max_i \Delta_i$, weil mit Hilfe der alten Position von p_m in jedem Δ_i nur ein einziger Korrekturterm berechnet werden muß; dies geht in $O(1)$.

Insgesamt benötigen die geschachtelten *while*-Schleifen also Zeit $O(T \cdot n)$, wobei T die Gesamtzahl der Durchläufe der inneren Schleife ist. Das Problem hierbei ist, daß über T leider keine vernünftige Aussage getroffen werden kann; T hängt stark vom Graphen (und damit von n) und der Qualität der initialen Startposition der Knoten ab. Man kann zeigen, daß das Newton-Raphson-Verfahren, gestartet mit einer weit von einer Nullstelle entfernten Anfangskonfiguration, nur sehr schlecht konvergiert.

2.7 Verbesserung und Erweiterung

Eine erste Verbesserung des Algorithmus besteht darin, die unbekannte Anzahl der Iterationen in jedem Schritt zu begrenzen, z. B. auf $10 \cdot n$. Weiterhin kann die Gesamtzahl der Iterationen des Newton-Raphson-Verfahrens natürlich gesenkt werden, indem man die Konstante ε größer wählt.

Der Algorithmus läßt sich leicht auf einen Graphen mit nicht-negativen Kantengewichten erweitern, indem einfach die Abstände der Knoten im Graphen voneinander modifiziert werden.

Es kann auch eine 3D-Version des Algorithmus angegeben werden: das (dann dreidimensionale) Newton-Raphson-Verfahren verwendet als Koeffizienten die entsprechenden Einträge der 3×3 großen Jacobi-Matrix. Die Darstellung in der Ebene wird dann allerdings etwas schwieriger. Entweder man wählt eine geeignete Projektion in die Ebene, oder der Graph wird mit Hilfe des Computers dreidimensional dargestellt; sei es durch Bewegung am Bildschirm (z. B. Rotation) oder technisch aufwendiger mittels Stereo-Brille mit eigenem Bild für jedes Auge.

3 Der Algorithmus von Fruchterman und Reingold

3.1 Teilchenphysik: Simulated Annealing

Der nun folgende Algorithmus betrachtet die Knoten als Kernteilchen in einer sich abkühlenden Masse, die gegenseitig Kräfte aufeinander ausüben.

Grundlage hierfür ist das folgende physikalische Modell:

Bei einem Abstand von 1 Femtometer ($1\,\mathrm{fm}{=}10^{-15}\,\mathrm{m}$) ist die anziehende Kraft zwischen zwei Protonen etwa 10mal so groß wie die abstoßende elektrische Kraft. Bei einem Abstand, der kleiner als 0.4 fm ist, ändert sich dies; hier ist die elektrische Kraft größer als die anziehende Gravitationskraft.

Aus diesem Grund kollabieren Kernteilchen nicht.

3.2 Konzept und Ziele

Wir betrachten also die Knoten als Kernteilchen. Je zwei üben anziehende und abstoßende Kräfte aufeinander aus. Ein System von n solchen Objekten wird n-Körper-Problem genannt und ist für n größer als 3 nur mit hohem Aufwand und nur numerisch lösbar.

Daher werden wir die anziehenden Kräfte nur zwischen benachbarten Punkten betrachten.

Im Ergebnis können wir also erwarten, daß benachbarte Knoten dicht beieinander liegen, da sie (und nur sie) anziehende Kräfte aufeinander ausüben. Knoten werden sich auch nicht häufen, da bei zu geringem Abstand große abstoßende Kräfte zwischen ihnen wirken.

3.3 Anziehende und abstoßende Kräfte

Wenn W die Breite der zur Verfügung stehenden Zeichenfläche und L die Höhe bezeichnet, und ferner

$$k = \sqrt{\frac{W \cdot L}{n}}$$

ist, so benutzen wir in Abhängigkeit des Abstandes d für die anziehenden bzw. abstoßenden Kräfte (es können natürlich auch andere Funktionen benutzt werden):

$$f_a(d) = \frac{d^2}{k} \quad \text{bzw.} \quad f_r(d) = -\frac{k^2}{d}.$$

In der Summe ergibt sich daraus folgendes Bild:

Bei einem Abstand von genau k voneinander heben sich also bei benachbarten Knoten anziehende und abstoßende Kräfte auf. Deswegen kann man k als die „optimale Kantenlänge“ ansehen. Leider tritt sie nur im trivialen Fall auf, bei dem der Graph nur aus zwei Knoten besteht.

Durch die gegenseitig wirkenden Kräfte werden die Teilchen in der Ebene verschoben. Wichtig dabei ist, daß in unserer Modellierung jede Kraft eine Bewegung induziert und nicht, wie in der Natur, eine Beschleunigung. Das hat den Vorteil, daß sich bei Gleichgewichtssituationen keine Teilchen mehr bewegen.

Damit verschobene Knoten innerhalb der Zeichenfläche bleiben, begrenzen wir vor jeder neuen Iteration die Koordinaten auf die maximale Breite und Höhe.

3.4 Änderung der Temperatur

Außerdem beschränken wir die maximale Änderung auf einen Temperaturwert t, der langsam kleiner wird und sich Null nähert. Wir starten z. B. mit einem Wert von einem Zehntel der Breite der Zeichnung und erniedrigen ihn nach jeder Iteration gemäß einer inversen linearen Funktion.

Dies simuliert das Abkühlen der Masse.

Auch hier sind durchaus andere Temperaturfunktionen anwendbar.

3.5 Der Algorithmus

Wir verteilen anfangs die Knoten zufällig (gleichverteilt) in der Zeichenfläche.

Jeder Knoten v hat eine Variable v.pos (=*position*), die seine aktuelle Position in der Ebene angibt, und eine Variable v.disp (=*displacement*), die angibt, um wieviel der Knoten am Ende der Iteration verschoben werden soll.

Dabei sind natürlich die durch die herrschende Temperatur und die Grenzen der Zeichenfläche gegebenen Beschränkungen einzuhalten.

Damit lautet der Algorithmus:

Algorithmus FR;
for $i := 1$ to iterations {
  /* abstoßende Kräfte */
  for all $v \in V$ {
    v.disp:=0;
    for all $u \in V \setminus \{v\}$ {
      Δ:=v.pos$-u$.pos;
      v.disp:=v.disp$+\frac{\Delta}{||\Delta||_2} \cdot f_r(||\Delta||_2)$;
    }
  }
  /* anziehende Kräfte */
  for all $e = (u, v) \in E$ {
    Δ:=v.pos$-u$.pos;
    v.disp:=v.disp$-\frac{\Delta}{||\Delta||_2} \cdot f_a(||\Delta||_2)$;
    u.disp:=u.disp$+\frac{\Delta}{||\Delta||_2} \cdot f_a(||\Delta||_2)$;
  }
  /* Temperatur- und Rahmengrenzen */
  for all $v \in V$ {
    v.pos:=v.pos$+\frac{v.\text{disp}}{||v.\text{disp}||_2} \cdot \min\{(||v.\text{disp}||_2, t\}$;
    v.pos.x:=$\min\{\frac{W}{2}, \max\{-\frac{W}{2}, v.\text{pos}.x\}\}$;
    v.pos.y:=$\min\{\frac{L}{2}, \max\{-\frac{L}{2}, v.\text{pos}.y\}\}$;
  }
  t:=cool(t);
}

3.6 Laufzeit-Analyse

Die *for*-Schleife für die abstoßenden Kräfte läuft in $O(|V|^2)$; die *for*-Schleife für die anziehenden Kräfte benötigt $O(|E|)$, da diese ja nur zwischen benachbarten Knoten betrachtet werden. Die Temperatur- und Rahmenbedingungen können schließlich in $O(|V|)$ berechnet werden.

Jede Iteration benötigt daher $O(|V|^2 + |E|)$ Zeit. Aber über die Zahl der Iterationen kann, ähnlich wie beim Algorithmus von Kamada und Kawai, leider nur wenig gesagt werden. Sie hängt nicht zuletzt von den gewählten Temperatur- und Kräftefunktionen ab. Nach einer Aussage von Eades (nicht nur zu diesem Algorithmus) erreicht aber fast jeder Graph nach 100 Iterationsschritten einen minimalen Energiezustand. . .

3.7 Verbesserung und Erweiterung

Im Algorithmus kann man andere Funktionen sowohl für die Temperatur als auch für die wirkenden Kräfte wählen. Dabei sollte man allerdings beachten, daß zur Überwindung „schlechter“ Ausgangskonfigurationen die Temperatur anfänglich hoch genug ist, um auch größere Positionsveränderungen zuzulassen. Nach dem Abkühlen auf (fast) Null wird zwar meist ein stabiler Zustand erreicht; ob dieser aber energetisch günstig ist, ist nicht sicher.

Deshalb kann man vor der eigentlichen Simulation das *Quenching* anwenden. Dies ist ähnlich wie das Simulated Annealing, nur daß mit einer hohen Temperatur gestartet und dann sehr rasch abgekühlt wird. Dadurch kann eine ungünstige Startposition in wenigen Iterationen zu einer besseren gelangen.

Im Gegensatz dazu steht das *Simulated Sintering*, das die Temperatur konstant auf einem niedrigen Stand beläßt. Dadurch vollziehen sich Änderungen zwar nur langsam, und Werte hängen schnell in lokalen Minima fest. Aber dieses Verfahren ist gut geeignet für eine Feinabstimmung aus einer bereits günstigen Ausgangslage heraus.

Auch eine Kombination von Quenching und Simulated Sintering ist sinnvoll.

Zur Reduktion des Berechnungsaufwandes ist es möglich, abstoßende Kräfte zwischen weit auseinanderliegenden Knoten zu vernachlässigen. Bei einer gleichmäßigen Verteilung der Punkte in der Ebene kann die Einsparung hier groß sein.

Auch für den hier vorgestellten Algorithmus gibt es eine dreidimensionale Version.

4 Weitere Algorithmen und Vergleich

4.1 Andere Ansätze

Es gibt eine Reihe anderer Algorithmen, die ungerichtete Graphen mit geraden Kanten zeichnen. Hier seien nur zwei genannt.

Der Algorithmus von Davidson und Harel implementiert den Ansatz des Simulated Annealing ohne Beschränkungen, wie wir sie in Kapitel 3 gemacht haben, ist also sehr rechenintensiv. Als zu minimierende Funktion verwenden

Davidson und Harel aber nicht eine physikalische Energiefunktion, sondern die gewichtete Summe von vier einzelnen Funktionen, die „Strafen“ für Kantenkreuzungen, Nähe zum Rand, Abstand der Knoten voneinander sowie die Verteilung der Knoten in der Ebene angeben. Die Gewichte dieser Funktionen zueinander werden als Parameter eingegeben. So ist es z. B. möglich, durch eine sehr hohe Strafe auf Kantenkreuzungen die Chancen auf eine planare Darstellung zu erhöhen.

Desweiteren gibt es einen Algorithmus von Tunkelang, der einfach in jeder Iteration für jeden Knoten 16 relativ zur aktuellen Position fest gegebene Punkte durchprobiert und darunter den günstigsten bestimmt. Die Bewertungsfunktion ist dabei ähnlich zu der von Davidson und Harel.

4.2 Vergleich

Zum direkten Vergleich der in Kapitel 2 und Kapitel 3 vorgestellten Algorithmen seien an dieser Stelle einige (einfache) Beispiele angegeben.

Die linken Bilder zeigen jeweils das Ergebnis von Kamada und Kawai; die rechten sind die Zeichnungen, die Fruchterman und Reingold liefern.

Bei dem in Kapitel 1 genannten Graphen (Symmetrie oder Planarität ?) sehen sich die Ergebnisse recht ähnlich; beide Algorithmen stellen die Symmetrie in den Vordergrund:

Auch den durch einen dreidimensionalen Würfel induzierten Graphen zeichnen beide gleichermaßen „räumlich“ und nehmen dabei Kantenkreuzungen in Kauf, obwohl der Graph planar ist:

Im folgenden Beispiel kann man gut erkennen, daß das Ergebnis von Fruchterman und Reingold nicht so ganz „gerade“ erscheint; dies ist auch bei Zeichnungen anderer Graphen häufig zu beobachten:

Bei größeren allgemeinen Graphen liefern alle vier genannten Algorithmen gute Ergebnisse. Brandenburg, Himsolt und Rohrer äußern in ihrem Artikel, in dem sie vergleichende Testergebnisse vorstellen, die Empfehlung, die Algorithmen von Kamada und Kawai bzw. von Fruchterman und Reingold zu bevorzugen.

Der Algorithmus von Davidson und Harel nimmt wegen seiner aufwendigen Implementation des Simulated Annealing viel Zeit in Anspruch, läßt dafür aber die freie Wahl der Gewichte zu. Der Algorithmus von Tunkelang zeigt durch seine diskrete Positionsbestimmung teilweise stark von den anderen Algorithmen abweichende, ungewöhnliche Darstellungen von Graphen.

Bei Kamada und Kawai sind die Zeichnungen von isomorphen Graphen meistens sofort als gleichartig erkennbar; es findet höchstens eine Drehung oder Spiegelung statt.

Der Spring Embedder-Algorithmus, der im Softwarepaket AGD (Algorithms for Graph Drawing) implementiert ist, verwendet sowohl in der zwei- als auch in der dreidimensionalen Variante Algorithmen von Fruchterman und Reingold.

Literatur

Kamada, Tomihisa and Kawai, Satoru:
An Algorithm for Drawing General Undirected Graphs
Information Processing Letters 31 (1989), pp. 7–15

Kuma, Aruna and Fowler, Richard H.:
A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions
available online
URL: `http://bahia.cs.panam.edu/info_vis/spr_tr.html`

Fruchterman, Thomas M. J. and Reingold, Edward M.:
Graph Drawing by Force-directed Placement
Software – Practice and Experience Vol. 21 (1991), pp. 1129–1164

Brandenburg, Franz J.; Himsolt, Michael and Rohrer, Christoph:
An Experimental Comparison of Force-directed and Randomized Graph Drawing Algorithms
in:
Brandenburg, Franz J. (Ed.):
Graph Drawing (Symposium on Graph Drawing, GD '95)
Springer Verlag 1995

Inhaltsverzeichnis

1 Einführung **2**

1.1 Zeichnen von Graphen . . . 2
1.2 Physikalische Modelle . . . 3

2 Der Algorithmus von Kamada und Kawai **4**

2.1 Federmechanik-Modell: Spring Embedder . . . 4
2.2 Konzept und Ziele . . . 4
2.3 Minimierung der Energie . . . 5
2.4 Das Newton-Raphson-Verfahren . . . 6
2.5 Der Algorithmus . . . 7
2.6 Laufzeit-Analyse . . . 9
2.7 Verbesserung und Erweiterung . . . 10

3 Der Algorithmus von Fruchterman und Reingold **10**

3.1 Teilchenphysik: Simulated Annealing . . . 10
3.2 Konzept und Ziele . . . 11
3.3 Anziehende und abstoßende Kräfte . . . 11
3.4 Änderung der Temperatur . . . 12
3.5 Der Algorithmus . . . 12
3.6 Laufzeit-Analyse . . . 13
3.7 Verbesserung und Erweiterung . . . 14

4 Weitere Algorithmen und Vergleich **14**

4.1 Andere Ansätze . . . 14
4.2 Vergleich . . . 15

Literatur **17**