technische universität
dortmund

fakultät für informatik

Bachelor-Arbeit

MetroMap-Layout-konforme Visualisierung von Höchstspannungsnetzen

Robin Möhring

9. August 2017

Gutachter:

Prof. Dr. Heinrich Müller

M.Sc. Dominic Siedhoff

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

Inhaltsverzeichnis

Mathematische Notation

Notation	Bedeutung
$\mathbb{N}$	Menge der natürlichen Zahlen $1, 2, 3, \ldots$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^d$	d-dimensionaler Raum
$\mathcal{M} = \{m_1, \ldots, m_N\}$	ungeordnete Menge $\mathcal{M}$ von N Elementen m_i
$\mathcal{M} = \langle m_1, \ldots, m_N \rangle$	geordnete Menge $\mathcal{M}$ von N Elementen m_i
$\mathbf{v}$	Vektor $\mathbf{v} = (v_1, \ldots, v_n)^T$ mit N Elementen v_i
$v_i^{(j)}$	i-tes Element des j-ten Vektors
$\mathbf{A}$	Matrix $\mathbf{A}$ mit Einträgen $a_{i,j}$
$G = (V, E)$	Graph G mit Knotenmenge V und Kantenmenge E

1 Einleitung

1.1 Motivation und Hintergrund

Das Höchstspannungsnetz, welches sich über Deutschland erstreckt ist riesig und besteht aus tausenden von Strommasten und Verbindungen. Da ist sehr vom Nutzen eine Karte zu haben, die einige Verbindungen und Orte verschiebt um eine weitaus bessere und übersichtlichere Darstellung zu erhalten. Das ist genau die Methode einer MetroMap, den Verlust genauer Daten um sicherzustellen, dass die wichtigen Verbindungen und Standorte schnell und sicher erkannt werden.
In dieser Arbeit geht es darum, die Methoden einer MetroMap für Bus- und Bahnverbindungen zu nutzen, um eine angepasste MetroMap für das deutsche Höchstspannungsnetz zu erhalten.

1.2 Aufbau der Arbeit

Abbildung 1.1: Karte des deutschen Höchstspannungsnetzes

Abbildung 1.2: MetroMap der Bahnverbindungen in Singapur

2 Vom Höchstspannungsnetz zum Graphen

2.1 Höchstspannungsnetz

Um elektrische Energie über große Distanzen zu transportieren werden Höchstspannungsnetze benutzt. Von einem Kraftwerk ausgehend wird versucht möglichst viele Haushalte, Industrie- und Gewerbebetriebe zu erreichen. Davon ausgehend wird auch der Standort der meisten Kraftwerke bestimmt. Einige Kraftwerke lassen sich nur an bestimmten Standorten errichten, zum Beispiel ein Wasserkraftwerk muss an einem Fluss oder Staudamm errichtet werden. So ist es oft nicht möglich genug Verbraucher zu erreichen, sodass es Mittel bedarf den elektrischen Strom auch über weite Strecken hinweg zu transportieren.

Die Generatoren der modernen Kraftwerke erzeugen eine Spannung von $10500V$, $21000V$ oder $27000V$[1]. Die Höhe dieser Spannung ist bestimmt durch die Größe bzw. die Leistungsfähigkeit des Kraftwerks und somit von der Nennleistung des Generators.

Da der überwiegende Teil der elektrischen Energie in Wärmekraftwerken erzeugt wird und diese mit einer Generatorleistung von 600 bis $1300MW$, bedeutet dies, dass Ströme zwischen $15000A$ und $30000A$ abgegeben werden müssen. Das ist jedoch weder aus technischen noch wirtschaftlichen Gründen für einen Transport über lange Distanzen lohnenswert, da es entweder sehr großen Leiterquerschnitte oder sehr große Stromverluste zur Folge hätte.

Es kann die gleiche Leistung P auch mit weniger Strom I und einer erhöhten Spannung U erreicht werden, denn die Beziehung lautet:

$$P = U \cdot I \tag{2.1}$$

Diese Eigenschaft wird ausgenutzt und das führt dazu, dass die Generatorspannung

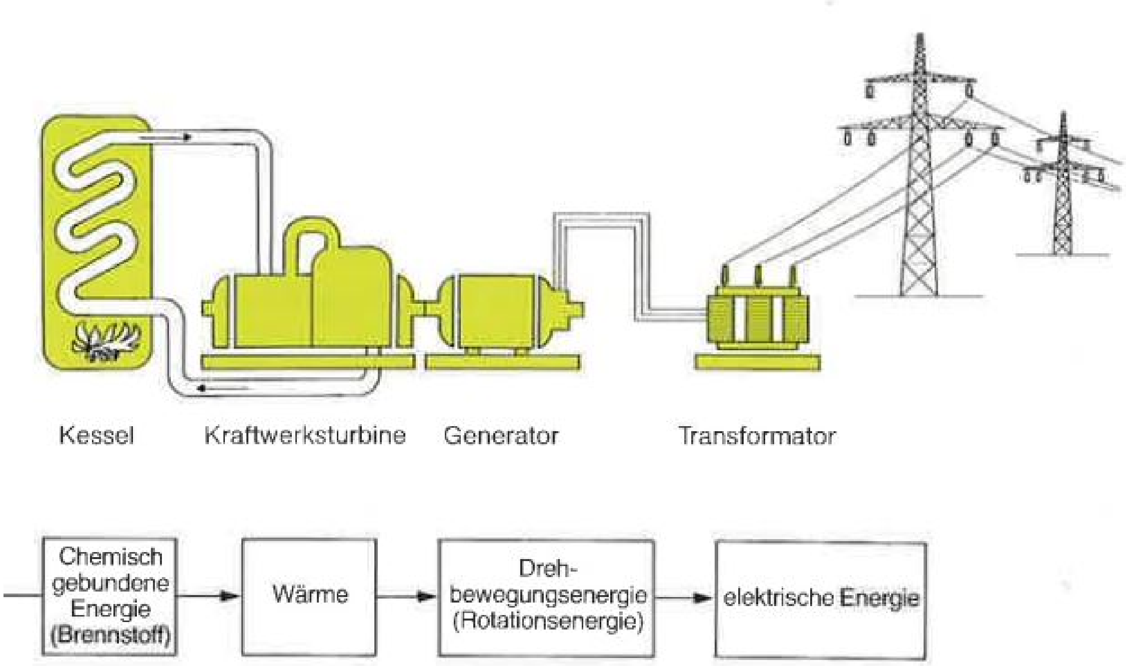


Abbildung 2.1: Prinzip der Stromerzeugung[1]

bereits direkt am Kraftwerk durch einen Transformator in eine höhere Spannung umgeformt wird. Dadurch wird die elektrische Leistung mit kleineren Stromstärken über die Netze geleitet.

2.2 Graph

Ein Graph ist eine Struktur, die Informationen über zusammenhängende Objekte repräsentiert. Die jeweiligen Objekte werden Knoten und deren Verbindungen Kanten genannt. Somit hat ein Graph eine Menge von Knoten und Kanten.

```
class Graph {
List<Knoten> knoten;
List<Kante> kanten;
}
```

Listing 2.1: Struktur des Graphen

Ein Knoten besteht aus einem Koordinatenpaar (x, y), welches definiert wo der Knoten liegt.

```
class Knoten {
int x,y;
public Knoten(int x, int y)
```

Abbildung 2.2: Karte des deutschen Höchstspannungsnetzes

```
{
this.x = x;
this.y = y;
}
}
```

Listing 2.2: Struktur eines Knotens

Die einzelnen Knoten werden über Kanten miteinander verbunden. Eine Kante speichert den Start- sowie Endknoten der Verbindung.

```
class Knoten {
int x,y;
public Knoten(int x, int y)
{
this.x = x;
```

```
this.y = y;
}
}
```

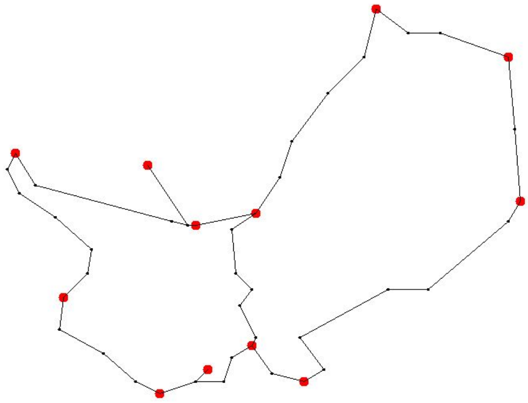
Listing 2.3: Struktur eines Knotens

Neue Knoten und Kanten können in der Graph-Klasse mit der Funktion $createKnoten()$ und $createKante()$ hinzugefügt werden. Diese Funktionen erstellen einen neuen Knoten beziehungsweise Kante und fügen sie der jeweiligen Liste hinzu.
Nun kann der Graph auf einer Oberfläche gezeichnet, indem erst die Knoten platziert werden anhand ihrer x und y Koordinate und anschließend die Kanten zwischen den jeweiligen Knoten.

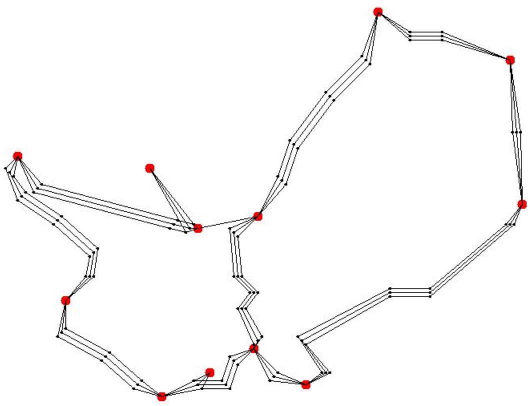
2.3 Vom Höchstspannungsnetz zum Graphen

Es wurden zwei Schritte unternommen um aus dem Höchstspannungsnetz einen repräsentierenden Graphen zu bekommen. Zum einen wurden die jeweiligen Masten zu Knoten und jeweils eins ihrer Leiterseile zu einer verbindenden Kante. Die Position der Masten war aus einer Karte des Höchstspannungsnetz zu bekommen, ebenso deren Verbindungen.
Da es besonders wichtig ist darzustellen wie viele einzelne Leiterseile über die jeweiligen Verbindungen laufen, musste dieses noch hinzugefügt werden. Es wurde für jedes Leiterseil welches über einen der Masten lief ein jeweiliger Knoten erstellt, und jeweils mit einer Kante verbunden. Diese zusätzlichen Knoten wurden direkt neben dem ursprünglichen Knoten verteilt. Nun hat man die wichtigsten Informationen des Netzes in einen Graphen übertragen.
Die Städte und Orte werden dabei als unbewegliche Knoten angelegt damit der resultierende Graph nicht zu sehr vom ursprünglichen abweicht. Jegliche Eckpunkte zwischen den Orten wurde mithilfe eines weiteren beweglichen Knotens modelliert.

(a) Ausschnit des deutschen Höchstspannungsnetzes. Die markierten Orte werden zu den Knoten und die Verbindungen werden zu den Kanten des Graphen.



(b) Den Ausschnitt der Karte als Graphen mit jeweils nur einer Leitung.



(c) Die zusätzlichen Leitungen wurden neben dem eigentlichen Knoten hinzugefügt. Hier jeweils immer genau drei.

Abbildung 2.3: Das schrittweise Vorgehen um einen repräsentierenden Graphen zu erhalten

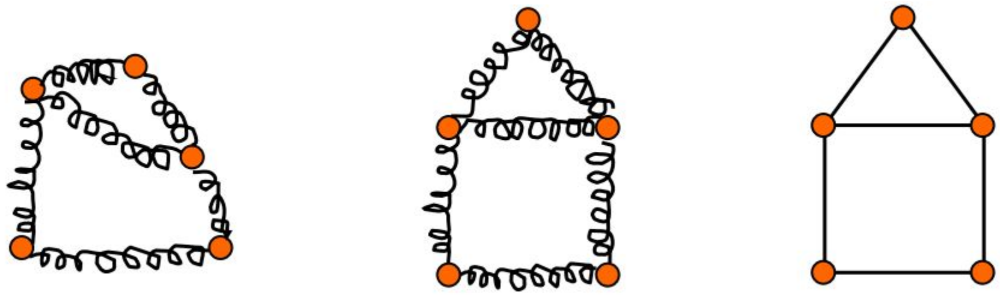
3 Spring-Algorithmus

In diesem Kapitel werden die Grundlagen des Spring-Algorithmus erklärt. Es geht um die Verwendung dieser Algorithmen, das grundsätzliche Vorgehen und die Erweiterbarkeit.
Das Problem der Darstellung basiert auf der Platzierung der Kanten sowie Knoten um eine möglichst ästhetische Zeichnung des Graphen zu erhalten, die gut lesbar und verständlich ist. Die grundlegenden Kriterien für eine ästhetischen Zeichnung eines Graphen sind:

1. die Knoten sollten gleichmäßig verteilt werden
2. minimale Überschneidung der Kanten
3. einheitliche Länge der Kanten
4. möglichst symmetrisch
5. alles soll sich innerhalb der Fläche befinden[3].

Das heißt es ist leicht möglich die wichtigen Informationen des Graphen anhand einer visuellen Darstellung zu erhalten. Dazu gehören unter anderem die Erkennung von Verbindungen zwischen Knoten oder deren Lage. Dies ist vor allem wichtig bei einer MetroMap wo es darum geht schnell herauszufinden wie es möglich ist von einem Punkt zu einem anderen zu kommen. Um dies zu erreichen gibt es verschiedenste Ansätze.
Im Folgenden wird es um ein Verfahren gehen, welches auf ein Modell der Physik basiert, indem Stahlkugeln für die Knoten, und Federn für die Kanten stehen. Durch die Federn wirken Kräfte auf die Kugeln ein, wodurch sich diese verschieben. Dies passiert solange, bis die Federn ihre optimale Länge haben. Die optimale Länge k wird durch die Anzahl der Knoten sowie der zur Verfügung stehender Fläche A bestimmt:

$$k = \sqrt{A/|V|} \tag{3.1}$$



(a) rote Knoten sind die Stahlkugeln verbunden mit Federn

(b) die Federn verschieben die Knoten

(c) darstellung als Graph

Abbildung 3.1: Darstellung als Stahlkugeln und Federn eines Graphen

$$A = W * L. \tag{3.2}$$

W ist die Weite und L die Länge der zugrunde liegenden Oberfläche, auf der, der Graph gezeichnet wird. Es ist wichtig zu wissen wie diese Maße sind um eine bessere Verteilung der Knoten zu ermöglichen.
Bildet die bisherige Platzierung der Kanten und Knoten einen planaren Graphen, so wird die neue die Platzierung in fast allen Fällen auch planar sein. Ein Graph ist planar wenn sich keine Kanten überschneiden, diese Eigenschaft ist besonders gewollt um den resultierenden Graphen noch übersichtlicher zu gestalten.

3.1 Spring-Algorithmus - Vorgehen

Zwischen jedem Knotenpaar $v_i, v_j \in V$ wird eine abstoßende Kraft $f^r_{v_i,v_j}$ berechnet. Alle benachbarten Knoten erhalten eine anziehende Kraft $f^a_{v_i,v_j}$. Zwei Knoten $v_i, v_j \in V$sind benachbart wenn $e_{v_i,v_j} \in E$ ist. Das führt dazu, dass verbundene Knoten näher zusammen gezeichnet werden, während sie noch immer einen gewissen Abstand zueinander haben. Der Algorithmus geht dabei in drei Schritten vor:

1. zwischen jedem Knotenpaar die abstoßende Kraft berechnen
2. benachbarten Knoten eine anziehende Kraft zuweisen
3. jeden Knoten seiner neuen Kraft nach bewegen.

Diese drei Schritte werden werden so oft wiederholt bis keine weitere Bewegung mehr stattfindet. Die Funktionen $f^a_{v_i,v_j}$ und $f^r_{v_i,v_j}$ sind wie folgt definiert:

Eingabe: $G := (V, E)$
Ausgabe: $G := (V, E)$
for $i := 1 \leq iterations$ **do**
  for $v_i \in V$ **do**
    $d_{v_i} := 0;$
    for $(v_j \in V)$ **do**
      if $v_i \neq v_j$ **then**
        $\Delta := p_{v_i} - p_{v_j};$
        $d_{v_i} := d_{v_i} + (\Delta/|\Delta|) \cdot f^r_{v_i,v_j}(|\Delta|));$
      end if
    end for
  end for

  for $e_{v_i,v_j} \in E$ **do**
    $\Delta := p_{v_i} - p_{v_j};$
    $d_{v_i} := d_{v_i} - (\Delta/|\Delta|) \cdot f^a_{v_i,v_j}(|\Delta|));$
    $d_{v_j} := d_{v_j} + (\Delta/|\Delta|) \cdot f^a_{v_i,v_j}(|\Delta|));$
  end for

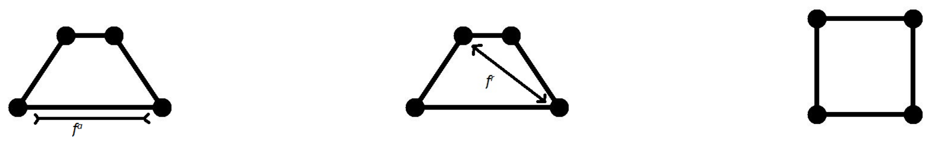
  for $v_i \in V$ **do**
    $p_{v_i} := p_{v_i} + (d_{v_i}/|d_{v_i}|) \cdot min(d_{v_i}, t);$
    $p^x_{v_i} := min(W/2, max(-W/2, p^x_{v_i}));$
    $p^y_{v_i} := min(L/2, max(-L/2, p^y_{v_i}))$
  end for
  $t := cool(t)$
end for

Algorithmus 3.1: Spring-Algorithmus

$$f^a_{v_i,v_j}(x) = x^2/k \tag{3.3}$$

$$f^r_{v_i,v_j}(x) = -k^2/x. \tag{3.4}$$

Der Vektor d_{v_i} gibt an in welcher Richtung der Knoten v_i bewegt wird. Während der Vektor p_{v_i} auf die momentane Position des Knoten v_i zeigt. Der Vektor Δ ist die Differenz zwischen p_{v_i} und p_{v_j}, der Richtungsvektor von v_i nach v_j. Die Variable *iterations* gibt die Anzahl der Durchläufe an. Bei jedem neuen Durchlauf wird der Vektor d_v eines jeden Knotens v auf 0 gesetzt. Mit zwei For-Schleifen geht man jede Knotenkombination durch und berechnet die abstoßende Kraft $f^r_{v_i,v_j}$ mithilfe der



(a) anziehende Kraft f^a zwischen dem dritten und vierten Knoten

(b) abstoßende Kraft f^r zwischen dem ersten und vierten Knoten

(c) die wirkenden Kräfte lösen sich auf

Abbildung 3.2: Wirkende Kräfte im Graphen

Länge des Δ Vektors. Anschließend wird der Bewegungsvektor auf den Einheitsvektor von Δ multipliziert mit der berechneten Kraft gesetzt. Dadurch drücken sich die Knoten direkt voneinander weg.
Der zweite Schritt besteht darin die anziehenden Kräfte zwischen den benachbarten Knoten zu berechnen. Dazu wird jede Kante einmal durchgegangen und der Bewegungsvektor von jedem betroffenen Knoten wird wie bei der abstoßenden Kraft aufaddiert.
Im letzten Schritt des Algorithmus werden die Knoten nun in Richtung ihres Bewegungsvektors bewegt. Es wird darauf geachtet, dass die Knoten dabei nicht das Bild verlassen. Die Variable t ermöglicht es, am Anfang viel Bewegung der Knoten zuzulassen und dies immer weiter einzuschränken, damit mit der Graph mit jeder Iteration feiner wird. Die Funktion $cool(t)$ regelt dabei wie weit sich jeder Knoten in der nächsten Iteration maximal bewegen darf. t könnte zum Beispiel bei der Länge des Bildes starten und sich nach jeder Iteration um ein Zehntel mit $cool(t)$ kürzen. Nach 10 Iterationen wären dadurch keine weiteren Bewegungen der Knoten mehr möglich.

3.2 Spring-Algorithmus - Erweiterbarkeit

Eine besondere Eigenschaft dieses Algorithmus ist die leichte Erweiterbarkeit. Er kann leicht an viele verschiedene Probleme angepasst werden. Der Algorithmus 2.1 ist bereits eine erste Erweiterung des ursprünglichen Algorithmus. Beim Bewegen der Knoten im dritten Schritt wird sichergestellt, dass sich die Knoten weiterhin auf der zur Verfügung stehenden Fläche befinden.
Ebenfalls modifizierbar sind die Funktionen $f^a_{v_i,v_j}$ und $f^r_{v_i,v_j}$. Im Kapitel 3 wird dies genutzt um den Graphen weiter anzupassen. Es ist leicht möglich noch weitere Kräfte

miteinzubeziehen oder sie ganz anderen Knoten zuzuweisen. Im späteren Verlauf wird es auch Knoten geben, die sich nicht bewegen sollen, weil sie zum Beispiel größere Städte oder zentrale Verbindungspunkte sind.

4 Das Kapitel 4

4.1 Kapitel 4 - Unterkapitel 1

4.2 Kapitel 4 - Unterkapitel 2

A Weitere Informationen

Abbildungsverzeichnis

Algorithmenverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis