Extremwertproblem Wegfindung

Vergleich verschiedener Algorithmen

Maximilian Stark

24. Oktober 2015

Inhaltsverzeichnis

1 Einleitung

2 Grundlagen und Terminologie

Zu Beginn werden in diesem Abschnitt die grundlegenden Begriffe der Graphen-Theorie geklärt. Auch Fachbegriffe aus der Implementierung durch die Informatik werden erläutert.

Das Ziel dieser Arbeit ist die Darstellung und der Vergleich verschiedener *Wegfindungs-Algorithmen* in der Anwendung an verschiedenen generierten *Graphen*.

Zugrunde all dem liegt die Graphen-Theorie. Deren Fundament ist der namensgebende *Graph* $G = (V, E)$, alternativ auch *Netz* genannt, welcher aus einer Menge von *Knoten* V (von engl. „Vertex") und aus einer Menge *Kanten* E (von engl. „Edge").

Zeichnerisch werden *Knoten* als Punkte oder Kreise dargestellt; *Kanten* als Verbindungslinien zwischen zwei *Knoten*. Jede *Kante* hat einen *Startknoten* und einen *Endknoten*. Wenn von einer *gerichteten Kante* die Rede ist, lässt sich das als Pfeil interpretieren, da die Verbindung unidirektional gilt. Ebenso gibt es die *gewichteten Kanten*, denen nicht nur zwei *Knoten* zugeordnet werden, sondern zusätzlich noch ein Gewicht w (von engl. „Weight"), ein Zahlenwert, der als Kosten der Beziehung zwischen den beiden *Knoten* gesehen werden kann.

In der Wegfindung ist ein *Weg* P (von engl. „Path") als geordnete Abfolge von *Knoten* definiert. Da in der Regel jedes *Knoten*-Paar nur einfach verbunden ist, reicht in der Implementierung dieser Ansatz aus.

Visuell wird der *Graph* durch einen *Layout-Algorithmus* dargestellt, welcher allen *Knoten* durch gewisse Berechnungen Positionen zuteilt (vgl. Abschnitt 5).

Generell sind *Algorithmen* eine festgelegte Abfolge von Schritten um Daten zu verarbeiten. In der Informatik sind diese einzelnen Schritte Befehle.

Um ein *Netz* zu generieren, wird eine Zufallsfunktion verwendet. Hierzu wird ein standardisierter *Pseodozufall*-Generator verwendet [1]. Dieser generiert kaum oder nur schwer vorhersagbare Abfolgen von Zahlen und genügt für unsere Zwecke. Aufgrund der nicht echten Zufälligkeit wird ein sogenanntes *Seed*-System benutzt, eine spezielle Zahl, mit deren Übergabe an den Generator stets die selbe Zahlenfolge erzeugt werden kann.

3 Aufbau und Bedienung des Programms

Das Programm, im eigentlichen Fokus stehend, fungiert sowohl als visuelle Möglichkeit der Darstellung, als auch als Quelle für Vergleichsdaten und Messungen in selbst erzeugten Szenarien. Geschrieben ist das Programm in der Programmiersprache *Java* unter Verwendung der *JavaFX*-Standardbibliothek [2].

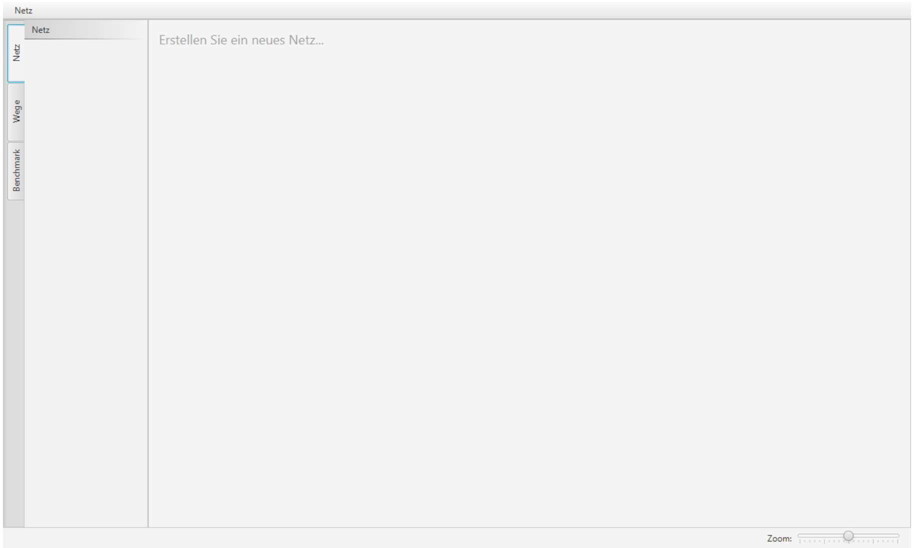

Abb. 1: Die Start- und Hauptansicht der Applikation

Auf den ersten Blick ist die Anwendungsoberfläche in zwei größere Bereiche aufgeteilt. Im linken, kleineren Seitenbereich werden detaillierte Informationen über den *Graphen*, bereits berechnete *Wege* und die Konfigurationsmöglichkeiten neuer Wege, in mehreren „Tabsünterteilt, angezeigt. Der große rechte Bereich, zu Beginn der Anwendung nur mit „Erstellen Sie ein neues Netz..."(Abb. 1) beschriftet, dient als Hauptansicht von sowohl des *Graphen*, als auch der Vergleichsstatistiken und Tabellen.

Die Bedienung kann vollständig mit der Maus erfolgen, da sich sämtliche Features visuell intuitiv und minimalistisch präsentieren. Nur vereinzelt führen Tastatureingaben oder „Hotkeys" zu mehr Komfort oder Genauigkeit der Anwendung. So kann beispielsweise das *Relayout* (siehe Abschnitt 5) des *Graphen* per „L" Taste, das *Generieren* (siehe Abschnitt 4) eines neuen unter Benutzung von „N".

4 Konstruktion des Graphen

Im nächsten Schritt werden wir nun einen Graphen erzeugen lassen und die Funktionsweise des Generators betrachten. Durch Klicken auf „Erstellen Sie ein neues Netz...” wird ein Dialog-Fenster geöffnet (Abb. 2), welches verschiedene Genererierungs-*Parameter* zur Konfiguration anbietet.

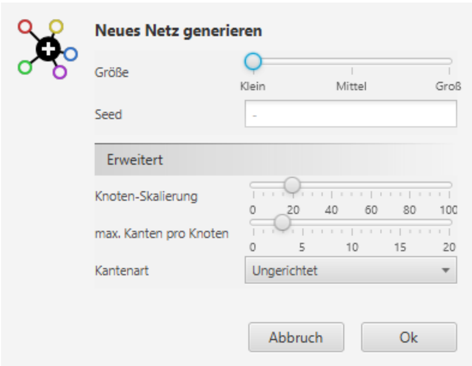

Abb. 2: Dialog zur *Netz*-Generierung

Unterteilt sind diese Einstellungen in zwei Bereiche: *Generell* und *Erweitert*. *Generelle* Optionen sind für den einfachen Gebrauch ausreichend mit einem Regler für die Größe s und einem *Seed*[1]-Eingabefeld ausgestattet.

Im *Erweitert*-Bereich lässt sich die Generierung aufs Genaueste einstellen. So können die Anzahl an Maximalknoten n_{max} und die maximale *Kanten*-Anzahl e_{max} pro *Knoten* festgelegt werden. Ebenso kann die Wahl zwischen drei Typen t von *Kanten* getroffen werden: *Ungerichtet*, *Gemischt* und *Gerichtet*[1], was alle *Kanten* des zu generierenden *Graphen* betrifft. Die Option *Gemischt* bewirkt, dass die Gerichtetheit jeder *Kante* zufallsbedingt ist.

Durch Bestätigen per Klick auf „Ok”wird der Generator mit diesen *Parametern* gestartet.

Zunächst wird der *Seed* für den Zufallsgenerator gesetzt. Danach wird aus den gegebenen Grenzwerten die tatsächliche Menge von *Knoten* berechnet und in den *Graphen* eingesetzt. Daraufhin wird für jeden *Knoten* eine Anzahl an *Kanten* bestimmt. Durch das „Clampen”, d.h Einzwicken, Eingrenzen, der Start- und Generierungswerte durch

$$e = max(1,\ R(0,\ min(n/2 - 1,\ e_{max})))$$

$$\begin{pmatrix} max(a, b) \rightarrow \text{Größere der beiden Parameter} \\ min(a, b) \rightarrow \text{Kleinere der beiden Parameter} \end{pmatrix}$$

wird gewährleistet, dass der Generator nicht mehr *Kanten* platzieren kann, als eindeutig möglich ist. Jetzt wird versucht, sämtliche *Knoten* durch zufällige Wahl mit einem anderen *Knoten* zu verbinden, wobei der jeweils gesuchte *Knoten* weder der

[1] Abschnitt 2 - Grundlagen und Terminologie

Ausgangsknoten selbst, noch ein bereits verbundener *Knoten* sein soll. Sobald eine Kombination gefunden wurde, wird die entsprechende *Kante* mit einem ebenfalls zufallsgenerierten *Gewicht* erstellt. Dann wird auf Basis des *Kanten-Typs* die Gerichtetheit bestimmt und schließlich wird die *Kante* im *Graphen* platziert (Alg. 2).

Alg. 1 *Graph-Generator v1*

Require: Zufallsgenerator R, max. Kantengewicht $W_{max} = 30$
Ensure: Graph g

```
procedure generateGraph(seed, s, n_max, e_max, t)
    R.seed ← seed
    n ← (R(0, n_max/2) + n_max/2) * s
    add n nodes to g
    for i = 0 → n do
        e ← max(1, R(0, min(n/2 − 1, e_max)))
        for j = 0 → e do
            index ← i
            repeat
                index ← R(0, n)
            until index = i or g.nodes[i] is connected to g.nodes[index]
            e ← edge from g.nodes[i] to g.nodes[index], w = R(0, W_max)
            if t = 1 or (t = 2 and R() > R()) then
                e.directed ← true
            end if
            add e to g
        end for
    end for
end procedure
```

Alg. 2 *Graph-Generator v2*

geg.: Zufallsgenerator R, max. Kantengewicht $W_{max} = 30$
ges.: Graph g

prozedur GENERIEREGRAPH($seed$, s, n_{max}, e_{max}, t)
  setze Seed von R zu $seed$
  sei n $R(n_{max}/2,\ n_{max}) * s$ ▷ Zufällige Anzahl im Interval $\left[n_{max}/2;\ n_{max}\right[$
  füge n $Knoten$ zu g hinzu
  für $i = 0 \rightarrow n$ **wiederhole**
    sei e $max(1,\ R(0,\ min(n/2-1,\ e_{max})))$
    für $j = 0 \rightarrow e$ **wiederhole**
      sei $index$ i
      wiederhole
        sei $index$ $R(0,\ n)$
      solange $index$ gleich i oder $Knoten_i$ mit $Knoten_{index}$ verbunden
      sei e Kante von $Knoten_i$ zu $Knoten_{index}$, Gewicht $w = R(0, W_{max})$
      wenn $t =$ GEMISCHT oder ($t =$ GERICHTET und $R() > R()$) **dann**
        ▷ $R > R =$ Zufallstest
        setze e gerichtet
      ende wenn
      füge e zu g hinzu
    ende für
  ende für
ende prozedur

5 Visuelles Layout des Graphen

In vorangegangen Abschnitten wurde das grundlegende Konzept eines *Graphen* bereits dargestellt. Wenn man sich nun mit der optimalen visuellen Darstellung eines *Graphen* auseinandersetzt, begibt man sich in die Thematik von *Layouts* (von engl. „Anordnung”) eines *Graphen*.

Es gibt die verschiedensten Ansätze, zu einer übersichtlichen Visualisierung zu gelangen, darunter die *force-directed algorithms* (von engl. „kraft-gerichtetöder „kraft-basierend”). Diese simulieren ein einem großen Molekül ähnelndes Konstrukt, in dem verschiedene *Kräfte*, die von *Knoten* und *Kanten* ausgehen, aufeinander wirken. Das Ziel solcher Simulationen ist das *mechanische Equilibrium*, die gegenseitige Aufhebung jeglicher wirkenden Kräfte.

Vorteile dieser Methode sind die ungleiche Flexibilität der Simulation und die sehr zufriedenstellenden Resultate in Bezug auf die *Planarität*[2] des visualiserten *Graphen*. Ein *Graph* gilt als *planar*, sobald sich keinerlei *Kanten* kreuzen.

Als Nachteil lässt sich die mitunter sehr lange Laufzeit der Berechnung sehen, um ein akzeptables Ergebnis zu erhalten, insbesondere bei sehr großen *Netzen*.

[2]

6 Wegfindungs-Algorithmen

6.1 Gröbste Züge von Intelligenz: Tiefensuche

-

6.2 Heuristik als Mittel zum Ziel: Der Dijkstra-Algorithmus

-

6.3 Der Allstar: Der A*-Algorithmus

-

7 Vergleichsstatistik und Fazit

-

8 Schluss

Literatur

[1] Java Zufalls-Funktion http://docs.oracle.com/javase/7/docs/api/java/util/Random.html *abgerufen am 18.10.15*

[2] JavaFX-Homepage http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm *abgerufen am 14.10.15*