



Max-Born-Gymnasium

**W-Seminararbeit  
aus dem Abiturjahrgang 2014/16**

**W-Seminar: Extrema**

**Seminarleiterin: Claudia Müller**

**Thema der Arbeit:**

**Wegfindung - Vergleich verschiedener Algorithmen**

**Verfasser:**

**Maximilian Léon Stark**

**Abgabetermin: 10. November 2015**

**Erreichte Punktzahl:**

**Unterschrift der Seminarleiterin/des Seminarleiters:**

# Inhaltsverzeichnis

<b>1</b>	<b>Risikofaktor Navigationsgerät</b>	<b>3</b>
<b>2</b>	<b>Grundlagen und Terminologie</b>	<b>4</b>
<b>3</b>	<b>Aufbau und Bedienung des Programms <i>PathFinder</i></b>	<b>5</b>
<b>4</b>	<b>Konstruktion eines Graphen in <i>PathFinder</i></b>	<b>6</b>
<b>5</b>	<b>Visuelles Layout von Graphen</b>	<b>7</b>
<b>6</b>	<b>Wegfindungs-Algorithmen</b>	<b>9</b>
6.1	Größte Züge von Intelligenz: Tiefensuche . . . . .	10
6.2	Heuristik als Mittel zum Ziel: Der Dijkstra-Algorithmus . . . . .	11
6.3	Der Allstar: Der A*-Algorithmus . . . . .	13
<b>7</b>	<b>Vergleichsstatistik und Fazit</b>	<b>15</b>
<b>8</b>	<b>Schluss</b>	<b>16</b>

# 1 Risikofaktor Navigationsgerät

„Wenn möglich, bitte wenden“ auf der Autobahn. „Jetzt links abbiegen“ im Kreisverkehr. Navigationssysteme können Todesfallen oder Verursacher schwerer Unglücke sein. Denn die Menschen vertrauen ihnen oft blind. So zum Beispiel erging es einem 33-jährigen im niedersächsischen Einbeck. Denn als die Polizei an der Unfallstelle eintraf, bot sich ihnen ein kurioses Bild: Der Pkw steckte auf einer abwärtsführenden Fußgängertreppe fest. Jegliche Versuche des Fahrers, sein Fahrzeug zu befreien, blieben erfolglos. Bedanken darf sich dieser Mann, ebenso wie eine junge Frau, die aufgrund eines Tippfehlers in einem Ort 850km entfernt vom gewünschten Ziel eintraf, bei der allzu freundlichen Stimme aus der Mittelkonsole [1]. Darum ist es umso wichtiger, dass „Navis“ immer über aktuellste Kartendaten verfügen und auch ausgiebig auf Fehler geprüft werden.

Aber nicht nur detailreiche Straßeninformationen sind für ein gutes Navigationsgerät von Bedeutung. Denn die Daten können noch so genau sein; wenn das Gerät keine vernünftigen Wege berechnen kann, ist es genau so unbrauchbar. Darum geht es in dieser Arbeit, nämlich die verschiedenen Methoden zur Wegberechnung in einem Straßen-Netz oder ähnlichem. Es werden drei *Algorithmen* vorgestellt und Vergleiche der selbigen angestellt, um festzustellen, für welche Zwecke welche Methode am zielführendsten ist. Als Werkzeug zur genaueren Untersuchung und für Vergleichsstatistiken habe ich zusätzlich ein Programm namens *PathFinder* geschrieben, in dem die *Algorithmen* adaptiert sind. Diese Ausführungen sind in gewisser Weise als Bedienungsanleitung des Programms zu sehen, da sich sämtliche Darstellungen und Tabellendaten darauf stützen.

Mit der Mathematik als Leitfach, im Themenbereich von Extremwertproblemen, wird konkret das *Problem des kürzesten Wegs* in Angriff genommen. Der optimale, kürzeste Weg zeichnet sich aber nicht nur durch seine Eigenschaft, am schnellsten von  $A$  nach  $B$  zu gelangen, aus, sondern auch durch die für die Bestimmung dieses Wegs benötigte Zeit und den Rechenaufwand. Denn ein „Navi“, das vier Stunden rechnet, um den besten Weg zu ermitteln, wird sich nicht bei den Konsumenten durchsetzen, aber genauso wenig ein Gerät, welches den Fahrer ohne Rechenzeit über Feld- und Waldwege lotst. Es muss eine *Balance zwischen Quantität und Qualität* gefunden werden. Und wo diese Mitte liegt, versuchen wir nun herauszufinden.

## 2 Grundlagen und Terminologie

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

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

Die Graphen-Theorie dient als Basis dieser Ausführungen. Zentrale Bedeutung hat der namensgebende *Graph*  $G = (V, E)$ , alternativ auch *Netz* genannt, welcher aus einer Menge von *Knoten*  $V$  (von engl. „Vertex“) und einer Menge *Kanten*  $E$  (von engl. „Edge“) besteht.

Zeichnerisch werden *Knoten* als Punkte oder Kreise dargestellt; *Kanten* als Verbindungslinien zwischen zwei *Knoten*. Jede *Kante* hat einen *Startknoten* und einen *Endknoten*.

Sobald sich keinerlei *Kanten* in der Darstellung kreuzen, wird ein *Graph* als *planar* bezeichnet. Wenn von einer *gerichteten Kante* die Rede ist, lässt sich diese 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 diese Annahme aus.

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

Um ein *Netz* zu generieren, wird eine Zufallsfunktion verwendet. Hierzu wird ein standardisierter *Pseudozufall*-Generator verwendet [2]. 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 *PathFinder*

Das selbstgeschriebene Programm *PathFinder*, im eigentlichen Fokus stehend, fungiert sowohl als visuelle Möglichkeit der Darstellung von *Graphen*, als auch als Quelle für Vergleichsdaten und Messungen in selbst erzeugten Szenarien. Geschrieben ist die Anwendung in der Programmiersprache *Java* unter Verwendung der *JavaFX*-Standardbibliothek [3] und umfasst über 3000 Zeilen Code in 39 Quelldateien.<sup>1</sup>

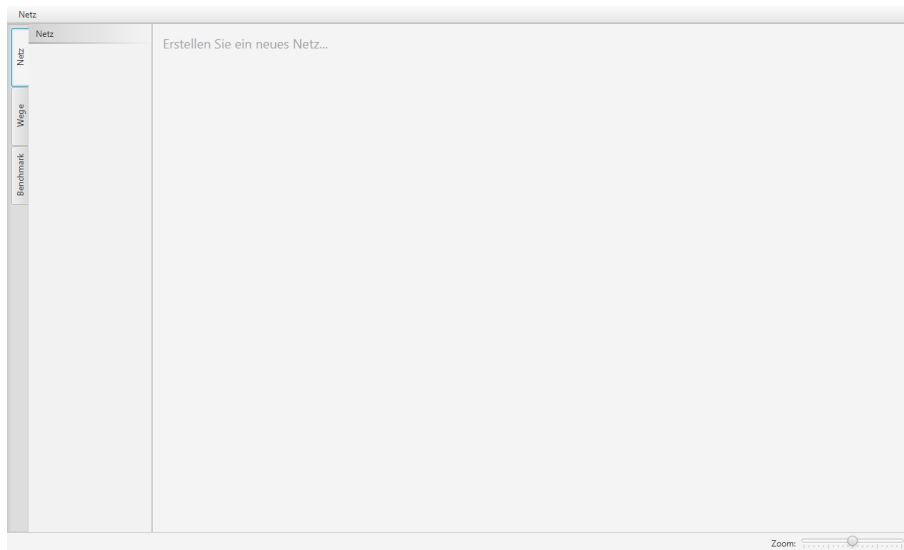


Abb. 1: Die Start- und Hauptansicht von *PathFinder*

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 „Tabs“ unterteilt, 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 *Netzes* unter Benutzung von „N“ erfolgen.

---

<sup>1</sup>siehe Anhang: CD-ROM

## 4 Konstruktion eines Graphen in *PathFinder*

Im nächsten Schritt wird nun ein Graph erzeugt und die Funktionsweise des Generators betrachtet. Durch Klicken auf „Erstellen Sie ein neues Netz...“ wird ein Dialog-Fenster geöffnet (Abb. 2), welches verschiedene Generierungs-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  $s_g$  und ein *Seed*-Eingabefeld ausgestattet. Der *Seed* wird verwendet, um die Möglichkeit zu haben, in späteren Tests mit dem gleichen *Graphen* zu arbeiten.

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*, 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 und ein *Netz* erzeugt.

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

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

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

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

*Ausgangsknoten* selbst, noch ein bereits verbundener *Knoten* sein soll. Sobald eine Kombination gefunden wurde, wird die entsprechende *Kante* mit einem ebenfalls

---

**Alg. 1** *Graph-Generator*

---

**geg.:** Zufallsgenerator  $R$ , max. Kantengewicht  $W_{max} = 30$

**ges.:** Graph  $g$

```

1: prozedur GENERIEREGRAPH( $seed, s_g, n_{max}, e_{max}, t$ )
2:   setze Seed von  $R$  zu  $seed$ 
3:   sei  $n \sim R(n_{max}/2, n_{max}) * s_g$   $\triangleright$  Zufällige Anzahl im Intervall  $[n_{max}/2; n_{max}]$ 
4:   füge  $n$  Knoten zu  $g$  hinzu
5:   für  $i = 0 \rightarrow n$  wiederhole
6:     sei  $e \sim \max(1, R(0, \min(n/2 - 1, e_{max})))$ 
7:     für  $j = 0 \rightarrow e$  wiederhole
8:       sei  $index \sim i$ 
9:       wiederhole
10:        sei  $index \sim R(0, n)$ 
11:        solange  $index$  gleich  $i$  oder  $Knoten_i$  mit  $Knoten_{index}$  verbunden
12:        sei  $e$  Kante von  $Knoten_i$  zu  $Knoten_{index}$ , Gewicht  $w = R(0, W_{max})$ 
13:        wenn  $t = \text{GEMISCHT}$  oder  $(t = \text{GERICHTET}$  und  $R() > R())$  dann
14:           $\triangleright R > R = \text{Zufallstest}$ 
15:          setze  $e$  gerichtet
16:        ende wenn
17:        füge  $e$  zu  $g$  hinzu
18:      ende für
19:    ende für
20: ende prozedur

```

---

zufallsgenerierten *Gewicht* erstellt. Dann wird auf Basis des *Kanten-Typs* die Gerichtetheit bestimmt und schließlich wird die *Kante* im *Graphen* platziert (Alg. 1)<sup>2</sup>.

## 5 Visuelles Layout von Graphen

In vorangegangenen 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 der *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-gerichtet“ oder „kraftbasiert“)[4]. Diese simulieren ein in einem großen Molekül ähnelndes Konstrukt, in dem verschiedene *Kräfte*, die von *Knoten* und *Kanten* ausgehen, aufeinander wirken. Das

---

<sup>2</sup>vgl. Anhang: GraphGenerator.java

Ziel solcher Simulationen ist das *mechanische Equilibrium*, die gegenseitige Aufhebung jeglicher wirkenden *Kräfte*.

Vorteile dieser Methode sind die enorme Flexibilität der Simulation und die sehr zufriedenstellenden Resultate in Bezug auf die *Planarität* des visualisierten *Graphen*. Als Nachteil lässt sich die mitunter sehr lange Laufzeit der Berechnung sehen, die benötigt wird, um ein akzeptables Ergebnis zu erhalten; insbesondere bei sehr großen *Netzen*.

Bei dem in *PathFinder* verwendeten *Algorithmus* handelt es sich um den *Fruchtermann-Reingold Algorithmus*, welcher 1991 von Thomas M. J. Fruchterman und Edward M. Reingold an der University of Illinois veröffentlicht wurde [5]. Ihre Methode verfolgt die folgenden Prinzipien:

1. Verbundene *Knoten* sollten nebeneinander liegen
2. *Knoten* sollten nicht zu nah beieinander platziert werden

Der Algorithmus versucht außerdem, alle *Knoten*  $n$  gleichmäßig innerhalb eines

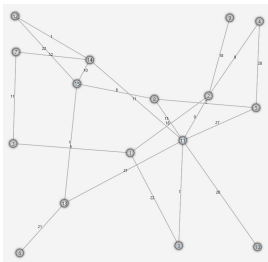


Abb. 3: F.-R. Algorithmus

Rahmens zu verteilen, einer als Parameter  $w$  und  $h$  (von engl. „width“ und „height“) definierten Maximalfläche. Zunächst wird die Konstante  $k$ , die optimale Distanz zwischen *Knoten*, als

$$k = \sqrt{\frac{w \cdot h}{n}}$$

definiert. Sie findet Verwendung in den beiden *Kräften* der Simulation. Die *Kraft*  $F_a$  beschreibt die Anziehung zwischen *Knoten*,  $F_r$  die Abstoßung dieser voneinander.

$$F_a(x) = \frac{x^2}{k} \qquad F_r(x) = \frac{k^2}{x}$$

Der Ablauf der Simulation lässt sich in drei Schritte zusammenfassen. Zuerst wird  $F_a$ , danach  $F_r$ , für jeden einzelnen *Knoten* berechnet. Im dritten Schritt werden die Effekte dieser berechneten *Kräfte* umgesetzt (*Disposition*), aber nur in durch die *Temperatur* begrenzter Länge. Die *Temperatur* ist ein Wert, der bei jedem Durchlauf des *Algorithmus* bis auf 0 verringert wird, um die Verschiebungen der *Knoten* immer präziser werden zu lassen. Anfangs wird der Wert beliebig festgelegt.



Der vorgeschlagene und damit auch in *PathFinder* umgesetzte Startwert  $t_0$ , dessen Änderung auf der Funktion  $t(s)$  abgebildet wird, entspricht

$$t_0 = \frac{1}{10} \cdot w \qquad t(s) = t_0 - \frac{t_0}{s_{max}} \cdot s \qquad s_{max} = s_g \cdot 500$$

Außerdem wird eine Maximalanzahl  $s_{max}$  an Simulationsschritten definiert, in deren Abhängigkeit die *Temperatur* verringert wird. In der Anwendung wird diese Maximalgröße wie angegeben berechnet, wobei  $s_g$  die bei der *Graph*-Erzeugung angegebene *Größe* ist. Somit erhält man in *PathFinder* folgende Gesamtfunktion:

$$t(s) = \frac{w}{10} - \frac{w}{s_g \cdot 5000} \cdot s$$

## 6 Wegfindungs-Algorithmen

Nun folgt der eigentliche Hauptteil der Arbeit. Nachdem jetzt sowohl das Programm *PathFinder*, als auch die darin angewandten Methoden zur Generierung und Visualisierung von *Graphen* erläutert worden sind, wird nun auf die Wegfindung eingegangen.

## 6.1 Größte Züge von Intelligenz: Tiefensuche

---

**Alg. 2** *Tiefensuche*

---

**geg.:** Graph  $G = (V, E)$

**ges.:** Weg  $P_{ab}$  von  $n_a$  nach  $n_b$

```
1: prozedur TIEFENSUCHE( $n_a, n_b$ )
2:   markiere alle Knoten in  $G$  als unbesucht, außer  $n_a$ 
3:   sei  $n_x$  aktiver Knoten  $n_a$ 
4:   solange nicht alle Knoten besucht sind wiederhole
5:     wenn  $n_x$  ist  $n_b$  dann
6:       erschließe Weg  $P_{ab}$  durch Vorgänger und beende Suche
7:     ende wenn
8:     wenn  $n_x$  keine unbesuchten Nachbarn hat dann
9:       sei  $n_x$  Vorgänger von  $n_x$ 
10:    sonst
11:      sortiere unbesuchte Nachbarn von  $n_x$ 
12:      sei  $n_{next}$  unbesuchter Nachbar mit geringstem Kantengewicht
13:      setze  $n_x$  als Vorgänger von  $n_{next}$ 
14:      sei  $n_x$   $n_{next}$ 
15:    ende wenn
16:  ende solange
17: ende prozedur
```

---

## 6.2 Heuristik als Mittel zum Ziel: Der Dijkstra-Algorithmus

---

**Alg. 3** *Dijkstra-Algorithmus*

---

**geg.:** Graph  $G = (V, E)$

**ges.:** Weg  $P_{ab}$  von  $n_a$  nach  $n_b$

---

-

## 6.3 Der Allstar: Der A\*-Algorithmus

---

**Alg. 4** *A\*-Algorithmus*

---

**geg.:** Graph  $G = (V, E)$

**ges.:** Weg  $P_{ab}$  von  $n_a$  nach  $n_b$

---

-

## 7 Vergleichsstatistik und Fazit

## 8 Schluss



# Literatur

- [1] vgl. Stern.de: Kuriose Navi-Unfälle  
<http://www.stern.de/digital/technik/navi-missgeschicke-in-100-metern-fahren-sie—in-den-fluss-3087618.html>  
*zul. abgerufen am 27.10.15*
  
- [2] Java Zufalls-Funktion  
<http://docs.oracle.com/javase/7/docs/api/java/util/Random.html>  
*zul. abgerufen am 18.10.15*
  
- [3] JavaFX-Homepage  
<http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm>  
*zul. abgerufen am 14.10.15*
  
- [4] Stephen G. Kobourov  
„Spring embedders and force directed graph drawing algorithms.”, 2012  
<http://arxiv.org/pdf/1201.3011v1.pdf>  
*zul. abgerufen am 24.10.15*
  
- [5] Thomas M. J. Fruchterman and Edward M. Reingold  
„Graph drawing by force-directed placement”, 1991  
[ftp://ftp.mathe2.uni-bayreuth.de/axel/papers/reingold:graph\\_drawing\\_by\\_force\\_directed\\_placement.pdf](ftp://ftp.mathe2.uni-bayreuth.de/axel/papers/reingold:graph_drawing_by_force_directed_placement.pdf)  
*zul. abgerufen am 24.10.15*

## ***Erklärung***

***Ich erkläre hiermit, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.***

***Ich bin damit einverstanden, dass innerhalb der Schule von Dritten in diese Seminararbeit Einsicht genommen werden kann.***

....., den .....  
(Ort) (Datum)

.....  
(Unterschrift Schüler)