Eine graphbasierte Sicht auf Eye-Tracking Daten

Tobias Meisel, Tobias Schneider und Safak Mumin

Abstract—In diesem Paper werden Daten aus Eye-Tracking Studien verarbeitet und visualisiert. Die Daten werden mit Hilfe eines Skripts für die vorgestellte, in Java implementierte Software aufbereitet. Die durch die Blickpunkte entstandenen Daten werden ins Transitionsmatrizen überführt, wobei hier die Granularität der Daten variiert werden kann. Der dadurch entstandene Graph kann mit verschiedenen Layoutalgorithmen in unterschiedlichen Formen dargestellt werden, um eine genaue Analyse der gegebenen Daten zu ermöglichen und um diese interaktiv explorierbar zu machen.

Index Terms—Graph Visualization, Eye-Tracking Data, Layout Algorithms

1 Einführung

Ein Graph ist eine abstrakte Struktur, mit deren Hilfe eine bestimmte Menge von Objekten oder Daten mit bestehenden Verbindungen zwischen diesen Objekten oder Daten anschaulich repräsentiert werden kann. Durch Kanten zwischen einzelnen Knoten können Abhängigkeiten und Verbindungen dargestellt werden.
Ziel dieser Ausarbeitung ist es, Daten einer Eye-Tracking Studie möglichst anschaulich zu repräsentieren um die entstandenen Graphen visuell zu analysieren. Mit Hilfe eines Skripts werden die Rohdaten für das vorgestellte Programm aufbereitet. Die eingelesenen Daten werden in Transitionsmatrizen überführt, in der die Granularität der Daten noch variiert werden kann. Der dadurch entstandene Graph kann mit verschiedenen Layoutalgorithmen visualisiert werden. Die verwendeten Algorithmen implementieren das Force Directed Layout, das Hierarchical Layout und das Circular Layout. Alle Graphvisualisierungen sind beliebig skalierbar, selektierbar und interaktiv gekoppelt um einzelne Punkte gleichzeitig in verschiedenen Layouts zu betrachten.

2 Verwandte Arbeiten

TODO: hier sollte noch related work rein

Einführung + Definition	0.5	ok
Verwandte Arbeiten	1	Safak
Datenverarbeitung	1	bisher 0.5
Visualisierung + GUI	1	VIEL zu wenig
Layoutalgorithmen	1.5	noch zu wenig

3 Definition

Die in der Arbeit verwendeten Datenstrukturen werden im Folgenden genauer definiert:
Ein Graph G ist ein geordnetes Paar (V, E). V bezeichnet eine Menge von Knoten v und $E \subseteq V \times V$ eine Menge von Kanten e. Jeder Kante e aus der Menge E wird ein Gewicht $w \in \mathbb{R}^+$ zugeordnet. Die hier betrachteten Graphen bestehen stets aus gerichteten, gewichteten Kanten.
Aufgebaut wird dieser Graph durch eine Transitionsmatrix. Eine Transitionsmatrix M ist eine $n \times n$ Matrix, deren Werte für Verbindungen zwischen zwei Knoten v_i und v_j stehen. Ein Eintrag an der Stelle (i, j) mit dem Wert 0 impliziert keine Verbindung zwischen den Knoten v_i und v_j. Ein Eintrag mit einem nichtnegativen Wert ungleich 0, impliziert eine Kante von v_i nach v_j. Entlang der Diagonalen werden die Einträge stets mit 0 aufgefüllt. Abhängig von der gewählten Granularität, wird auch die Größe der Transitionsmatrix geändert.

4 Datenverarbeitung

Die zur Verfügung gestellten Daten der Eye-Tracking Studie sind bisher in einzelnen Dateien gespeichert. Die Daten durchlaufen mehrere Schritte zur Aufbereitung bis sie in der vorgestellten Software verwendet werden können. Der Grund für diesen Vorgang ist, dass in der Software die Daten nicht jedes mal neu berechnet werden müssen, sondern in einem sogenannten Preprocessing soweit aufbereitet werden, dass sie im Tool nur noch eingelesen werden müssen.

4.1 Eye-Tracking Daten

TODO: mehr/bessere/korrekte Infos zu den Daten
In der Eye-Tracking Studie wird den Probanden eine bestimmte Karte (hier Antwerpen) vorgelegt. Je nach Aufgabenstellung ist das Ziel, ein bestimmtes Ziel zu finden oder eine Linie zu verfolgen. Parallel dazu werden die Augenbewegungen der Probanden verfolgt und in einer $.tsv$ Datei gespeichert.

4.2 Preprocessing im Python Skript

Die in der $.tsv$ Datei gespeicherten Daten werden mit Hilfe eines in Python geschriebenen Skripts $preprocess.py$ aufbereitet. Das Skript wird in der Konsole mit

```
python preprocess.py path/to/my/data
```

aufgerufen. Es wird der als Parameter übergebene Ordner samt Unterordnern nach $.tsv$ Dateien durchsucht. Aus diesen Dateien werden die Eye-Tracking Daten extrahiert, aufbereitet und in eine effektiv einlesbare Form gebracht. Vor Beenden des Skripts wird noch ein neuer Ordner mit dem Namen $ProcessedData$ erstellt. Dieser beinhaltet eine $.points$ Datei in der die Blickpunkte der Probanden gespeichert werden. Die Datei enthält folgende Attribute:

Attribut	Typ	Definiton
filename	String	Dateiname des verwendeten Stimuli
proband	String	Proband wird anhand einer Nummer identifiziert (proband01)
timestamp	Integer	exakter Zeitpunkt (in ms) an dem der Punkt (x, y) betrachtet wird
x	Integer	die x-Koordinate des Blickpunktes
y	Integer	die y-Koordinate des Blickpunktes
fixation duration	Integer	Dauer (in ms) wie lange der Proband den Blickpunkt fixiert

Zusätzlich werden in dem erstellten Ordner auch alle verwendeten Stimuli abgelegt. Dieser Vorgang muss für die Daten einer Eye-Tracking Studie nur einmal durchgeführt werden.

4.3 Einlesen und Verarbeiten von Daten

TODO: einlesen und verarbeiten von daten im tool, matrix reinschreiben und vllt auch beim nächsten kapitel

4.4 Skalierung der Daten

TODO: frame zum rastern der blickpunkte beschreiben

5 VISUALISIERUNG

Bei der Visualisierung von Graphen steht das Ziel, die Informationen möglichst anschaulich darzustellen, im Vordergrund. Eine genaue visuelle Analyse der Eye-Tracking Daten kann nur gewährleistet werden, wenn die Daten entsprechend aufbereitet werden.

5.1 GUI

TODO: paar details zur implementierung, welche sprache, frameworks usw. was kann man in der GUI machen, schön viele bilder usw

5.2 Layoutalgorithmen

Für die Darstellung eines Graphen ist die Wahl des entsprechenden Layoutalgorithmus ausschlaggebend. Ein Betrachter sollte auf keinen Fall mit einem Layout konfrontiert werden, was keinerlei Rückschlüsse auf die visualisierten Daten ermöglicht. Ziel der Visualisierung ist eine Betonung der Eigenschaften des Graphen bzw. der zugrundeliegenden Informationen. Welches Layout für einen Betrachter am besten geeignet ist, muss individuell entschieden werden und hängt auch vom Zweck der Visualisierung ab. Dennoch gibt es einige Kriterien, welche ein bestimmtes Layout geeigneter machen [1] [2] :

- wenige kreuzende Kanten
- eine korrekte Darstellung (bezogen auf den Abstand) der Knoten abhängig von der Gerätegröße
- wenig knickende Kanten (z.B. bei Hierarchical Layout)
- möglichst ästethischer Abstand benachbarter Knoten im Verhältnis zur freien Fläche

Zu denselben Ergebnissen führten auch mehrere empirische Studien von Purchase [3] hinsichtlich der Ästhetik von Graphlayouts. Hauptsächlich die drei Kriterien Symmetrie, Kantenübergänge und Krümmung wurden bezüglich Benutzerpräferenzen untersucht. Ergebnis der Studie war, dass die Minimierung von Krümmung und Kantenübergängen die Leistung der Probanden in verschiedenen Aufgaben erheblich verbesserte [4].
Im Folgenden werden die verwendeten Layoutalgorithmen vorgestellt und die charakteristischen Eigenschaften näher betrachtet.

5.2.1 Force Directed Layout

Das Force Directed Layout beschreibt ein Layout, welches eine Menge von Knoten mithilfe An- und Abstoßungskräfte ausrichtet. Anziehende Kräfte werden wie bei einer Feder dazu genutzt, um Paare von Endknoten an einer Kante anzuziehen. Abstoßende Kräfte werden wie beim Coloumb-Gesetz abgestoßen, um alle Paare von Knoten zu separieren. Der Algorithmus bietet sich besonders dann an, wenn die Zugehörigkeit einzelner Gruppen in einem Graphen visualisiert werden sollen. Durch die Anziehungskräfte werden zusammengehörige Knoten angezogen und später als eine Gruppe wahrgenommen, wohingegen der Abstand nicht zusammengehörige Knoten durch die Abstoßungskräfte vergrößert wird.
Den Autoren des im Folgenden vorgestellten Algorithmus waren fünf Punkte besonders wichtig [5]:

1) Gleichmäßiges Verteilen der Knoten im Frame
2) Minimieren der sich kreuzenden Kanten
3) Kantenlänge soll einheitlich lang sein
4) Darstellung vererbter Symmetrie
5) Anpassen an das gegebene Frame

Der Algorithmus wird nun in vereinfachter Form als Pseudocode dargestellt. Jeder Knoten besitzt zwei Vektoren $.pos$ und $.disp$:

```
area = W * L                                                    1
G = (V, E)                                                      2
k = sqrt(area/|V|)                                              3
f_a(x) = x^2/k                                                  4
f_r(x) = k^2/x                                                  5
                                                                6
for i=1 to ITERATIONS:                                          7
  for v in V:                                                   8
    v.disp = 0                                                  9
    for u in V:                                                 10
      if u != V:                                                11
        Δ = v.pos − u.pos                                       12
        v.disp = v.disp + (Δ/|Δ|)*f_r(|Δ|)                      13
                                                                14
  for e in E:                                                   15
    Δ = e.v.pos − e.u.pos                                       16
    e.v.disp = e.v.disp − (Δ/|Δ|)*f_a(|Δ|)                      17
    e.u.disp = e.u.disp + (Δ/|Δ|)*f_a(|Δ|)                      18
                                                                19
  for v in V:                                                   20
    v.pos = v.pos + (v.disp/|v.disp|)*min(v.disp, t)            21
```

```
    v.pos.x = min(W/2, max(-W/2, v.pos.x))  22
    v.pos.y = min(L/2, max(-L/2, v.pos.y))  23
                                            24
  t = cool(t)                               25
```

In Zeile 1 wird die Fläche des zu Verfügung stehenden Fensters berechnet. Die darauffolgenden vier Zeilen sind Konstanten beziehungsweise übergebene Parameter. Der Wert der Konstanten $iterations$ in Zeile 7 gibt an, wie oft der Algorithmus ausgeführt wird. Ziel ist es, ein Kräftegleichgewicht zu erhalten. Umso öfter der Algorithmus diese Schleife ausführt, umso besser wird das Ergebnis. In Zeile 8-13 werden die Abstoßungskräfte berechnet. Δ steht in diesem Fall für den Unterschied zweier Vektoren. In Zeile 15-18 werden die Anziehungskräfte berechnet. Jede Kante besteht aus einem geordneten Paar von Knoten $.v$ und $.u$. In Zeile 20-23 wird die maximale Verschiebung der Knoten abhängig von der Temperatur t begrenzt. Außerdem wird darauf geachtet, dass kein Knoten außerhalb des Fensters platziert wird. In der letzten Zeile wird die Temperatur durch eine entsprechende Kühlfunktion $cool(t)$ reduziert.
Vorteile dieses Algorithmus sind vor allem im Bezug auf Finden eines Pfades und Erkennung von Subgraphen deutlich. Dieses Resultat wurde in einer Studie [6] im Vergleich mit anderen Layoutalgorithmen bestätigt.
Ein Nachteil ist eine Laufzeit von $\mathcal{O}(n^3)$. Die äußere Schleife, welche die Anzahl der durchgeführten Iterationen bestimmt, benötigt $\mathcal{O}(n)$ und zusätzlich muss innerhalb dieser Schleife jedes Paar von Knoten miteinander verglichen werden. Allerdings kann die Laufzeit durch die beliebige Wahl der Konstanten $iterations$ verringert oder vergrößert werden. Jedoch kann bei einer zu klein gewählten Zahl das gewünschte Resultat nicht zufriedenstellend sein.
In unserer Implementierung wurde der Wert von $iterations$ auf 1000 gesetzt.

5.2.2 Hierarchical Layout

TODO: beschreibe hierarchical layout und bild

5.2.3 Circular Layout

Das Circular Layout beschreibt ein Layout auf dem alle Knoten kreisförmig angeordnet werden. So entsteht ein Polygon n-ten Grades, wobei n der Anzahl der Knoten entspricht. Alle Knoten haben zu ihren Nachbarn denselben Abstand. Somit kann kein Knoten besonders hervorgehoben oder vernachlässigt werden.
Die Algorithmen in dem Paper [7] gelten nur für einen $biconnected$ Graphen. Da wir diese Eigenschaft nicht garantieren können, wurde der Algorithmus etwas abgewandelt um trotzdem ein entsprechendes Ergebnis zu erzielen. Das Ziel der Kantenminimierung steht in diesem Fall im Vordergrund.
Der Algorithmus wird nun im Folgenden in vereinfachter Form als Pseudocode angegeben. Für einen gegeben Graph G wird das Circular Layout angewendet. G verfügt außerdem über die Methoden $getAllNodes$, $mapNodes$ und $getStartNodes$:

```
nodes = G.getAllNodes()                  1
startNodes = G.getStartNodes()           2
sortedNodes = G.mapNodes()               3
                                         4
for node in startNodes:                  5
  if depthSearch(node):                  6
    return                               7
                                         8
umfang = maxRadius * 2 * nodes.size      9
radius = (umfang / 2 / Π) * 1.1          10
sliceSize = 2 * Π / num                  11
                                         12
for i=0; i < nodes.size; i++:            13
  node = sortedNodes.get(i)              14
  x = cosinus(sliceSize * i - Π / 2) * radius15
  y = sinus(sliceSize * i - Π / 2) * radius16
  node.setPosition(x, y)                 17
```

In Zeile 1-3 werden drei Listen initialisiert, welche im weiteren Verlauf benötigt werden. In Zeile 5-7 wird die hierarchische Anordnung der Knoten ermittelt, indem zuerst über alle Startknoten iteriert wird und dann für jeden Startknoten eine Tiefensuche gestartet wird. War die Tiefensuche erfolgreich, ist die Anordnung der Knoten bekannt und es kann gezeichnet werden. In Zeile 9-11 werden Umfang, Radius und die Größe des Winkels zwischen zwei Knoten (betrachtet vom Mittelpunkt aus) berechnet. In Zeile 13-17 werden die Knoten letztendlich gezeichnet.
Bei der Zeichnung des Circular Layout spielen sich kreuzende Kanten eine große Rolle. Es gibt verschiedene Möglichkeiten diese zu reduzieren. Ein in [7] vorgestellter Algorithmus zählt die Anzahl der sich kreuzenden Kanten und läuft solange bis keine Verbesserung mehr erreicht. Aufgrund des Umfangs des Algorithmus wurde dieser in unserer Implementierung nicht berücksichtigt.
Der hier vorgestellte Algorithmus hat eine Laufzeit von $\mathcal{O}(n^2)$.

6 Schlussfolgerung

TODO: vllt nicht gerade Schlussfolgerung siehe andere projekt inf

References

[1] Diel, Stepahn and Görg, Carsten and Kerren, Andreas Preserving the Mental Map using Foresighted Layout *Eurographics / IEEE VGTC Symposium on Visualization*, 2001.

[2] Beck, Fabian and Burch, Michael and Diehl, Stephan and Weiskopf, Daniel The State of the Art in Visualizing Dynamic Graphs *The Eurographics Association*, 2014.

[3] Helen C Purchase, David Carrington, and Jo-Anne Allder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):233–255, 2002.

[4] Guy Melancon and Ivan Herman. Circular drawings of rooted trees. In *IN REPORTS OF THE CENTRE FOR MATHEMATICS AND COMPUTER SCIENCES*. Citeseer, 1998.

[5] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[6] Mathias Pohl, Markus Schmitt, and Stephan Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. In *Computational Aesthetics*, pages 49–56, 2009.

[7] Janet M. Six and Ioannis G. Tollis A framework and algorithms for circular drawings of graphs *Journal of Discrete Algorithms*, 4(1):25-50, 2006.