Balancing Performance and Visual Appeal in Dynamic Graph Labeling

Nils Henrik Seitz* (Mat.Nr. 218205308)

Faculty of Computer Science
University of Rostock

Abstract

To tackle the NP-hard problem of graph labeling, a greedy approach was used to generate label positions. Conflict Detection is done fastly with simple bounding boxes, so that valid labels can be stored quick and efficient in proper data structures like maps and quadtrees. Animations are used to ease changes of label positions or visibility on the eye. All these techniques combined make for a fluid and visually appealing experience, that does justice to dynamic labeling.

Um das NP-Harte Problem des Graph Labeling in dynamischen Graphen anzugehen, wird ein gieriger Ansatz genutzt, um Label Position zu generieren. Mit simplen Bounding Boxes wird schnelle Konfliktermittlung betrieben, sodass valide Positionen durch geeignete Datenstrukturen wie Maps und Quadtrees effizient und zügig gespeichert werden. Animationen runden Änderungen der Labelpositionen oder -sichtbarkeit visuell ab. Diese Maßnahmen führen im Gesamtbild zu einem flüssigen, visuell ansprechenden Ergebnis, das der Natur von dynamischem Labeling gerecht wird.

Keywords: automatic label placement, dynamic labeling, graph labeling, interactive labeling

Bemerkung: Dieses Projekt wurde in Kooperation mit Nico Trebbin erstellt. Der Bericht wurde als eigenständiges Werk von dem Autor angefertigt. Bilddateien haben gegebenenfalls einen gemeinsamen Ursprung als Material für die Präsentation im Rahmen des Projektes.

Viele der Kernbegriffe des Projektes werden in diesem Bericht im Original verwendet, um Konsistenz mit der Codebasis und den Präsentationen des Projektes zu erhalten.

*ns464@uni-rostock.de

1. Einleitung

Graphen sind wohl die mit am meisten genutzte Datenstruktur der Informatik. Durch sie lassen sich viele verschiedene Sachverhalte modellieren. Mit `iGraph.js` existiert als praktische Grundlage ein Tool zur interaktiven und dynamischen Visualisierung von Graphen.

Das Problem ist, dass ohne die Namen der Knoten und Kanten dem Graph recht wenig Bedeutung abgewonnen werden kann, es fehlt Kontext. Der Kontext kann durch ein Labeling des Graphen klarer werden. Dazu sind allerdings zwei Dinge zentral:

Erstens sollte das Labeling visuell ansprechend sein, das heißt, es muss intuitiv klar sein, welches Label zu welchem Knoten oder zu welcher Kante gehört und es sollten keine Überdeckungen entstehen.

Zweitens sollte das Labeling performant sein, da selbst die schönste Anordnung von Labels nicht nützlich ist in einem dynamischen und interaktiven Umfeld, wenn sie zu lange zum Errechnen braucht. Den bestmöglichen Kompromiss aus Labeling Qualität und Performance zu finden, ist die Kernaufgabe dieses Projektes.

Dazu haben wir eine Labeling Pipeline implementiert:

1. Koordinaten der Knoten/Kanten updaten (von `iGraph.js`)
2. Potentielle Labelpositionen generieren
3. Konfliktermittlung
4. Tatsächliche Labelposition festlegen und speichern
5. Visualisierung

Natürlich ist diese nicht ganz so statisch und sukzessiv wie hier vereinfacht beschrieben. Zum Beispiel finden Schritt 2 und 3 verschränkt statt, also werden nicht erst alle potentiellen Positionen generiert und dann nach Konflikten gesucht. Stattdessen finden sie abwechseln statt.

Dies spiegelt den gierigen Ansatz wieder, der dieser Implementation zu Grunde liegt. Dadurch soll die Kernaufgabe des Projektes umgesetzt werden, also schnell möglichst gute Labelpositionen zu finden. Dies schien die sinnvollste Option in Anbetracht der Tatsache, dass das Labeling Problem NP-Hard ist.

Es sei abschließend gesagt, dass diese Pipeline und die verwendeten Techniken und Methoden nicht ausschließlich für Graph Labeling genutzt werden müssen. Beispielsweise wäre auch eine Nutzung für Radare von etwa Schiffen oder Flugzeugen vorstellbar, bei denen andere Schiffe/Flugzeuge mit Namen versehen werden sollen. Formal genommen ist das ja ein Graph ohne Kanten - sollte nur darstellen, dass die Techniken und Algorithmen sehr generisch sind.

Als hauptsächliche theoretische Grundlage diente zur Spezifikation der Pipeline sowie der Labeling Verfahren und Konfliktermittlung eine Arbeit von Luboschik et al.[1], in der das Vorgehen beschrieben wird.

2. Labeling

Labeling bezeichnet den Prozess, ein Objekt sichtbar mit seinem Namen oder anderen Informationen zu versehen. Somit gilt es für einen Graphen, seine

Knoten und seine Kanten zu labeln.

In erster Linie werden hier die verschiedenen Verfahren erklärt, wie Labelpositionen abhängig von ihrem Objekt und seiner Position erzeugt werden. Oft wird im folgenden über "potentiellen"Labelpositionen geschrieben - der Grund dafür ist, dass, sobald mehrere Objekte gelabelt werden müssen, sich deren potentielle Labelpositionen gegenseitig überdecken können. Dieses Problem wird in Conflict Detection und Bounds aufgelöst. Somit wird sich hier auf die Verfahren für ein individuelles Objekt konzentriert.

2.1. Knotenlabeling

Beim Labeling der Knoten sind als geometrische Repräsentation ein Kreis, das heißt, x, y-Koordinaten und ein Radius, sowie der entsprechende Name des Knotens durch die Daten von `iGraph.js` gegeben.

Nun ist es Aufgabe des Labelings, den Text des Labels erkennbar und visuell ansprechend in Nähe des dazugehörigen Knotens zu platzieren. Die Breite und die Höhe eines Labels werden von der ausgewählten Schriftart (siehe Konfiguration) beeinflusst. Die Breite eines Labels hängt zusätzlich von dem entsprechenden Namen bzw. der Anzahl der Buchstaben ab.

So ergeben sich die Höhe und Breite des Labels (sowie der Text selbst) und es muss eine Position in x, y-Koordinaten in Abhängigkeit von dem entsprechenden Knoten gefunden werden.

Die Reihenfolge des Labelings ergibt sich aus einer einmaligen Sortierung. Diese ist absteigend und sortiert nach der Größe der Radien der Knoten. Ein größerer Radius steht für eine höhere Relevanz, denn er bedeutet mehr Nachbarn und daher sind diese Knoten vom größerem Interesse.

Zur Generierung der Position eines potentiellen Labels werden nacheinander verschiedene Verfahren verwendet, um möglichst effizient Labelpositionen zu generieren mit möglichst wenig Überdeckung der einzelnen potentiellen Positionen und so, dass niemals der dazugehörige Knoten überdeckt wird:

2.1.1. 4-Position-Model

Das 4-Position-Model positioniert die potentiellen Labels möglichst nahe an dem korrespondieren Knoten in den Himmelsrichtungen Nord-Ost, Nord-West, Süd-Ost und Süd-West. Die erste Position ist Nord-Ost, also rechts-oben, und von dort an wird entgegen x, y-Koordinaten ein weiteres Label generiert (wenn nötig).

Dieses und das nachfolgendene Verfahren sind durch die gängingen Standards der Kartografie inspiriert. Auch dort werden eben beschriebene Positionen generiert, allerdings in leicht veränderter Reihenfolge. ZITAT BITTE

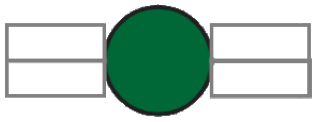

Abbildung 1: Labelpositionen im 4-Position-Model von ausgehenden vom dazugehörigen Knoten

2.1.2. 8-Position-Model

Im 8-Position-Model werden die potentiellen Labelpositionen in den Haupthimmelsrichtungen Ost, Nord, West, Süd generiert. Startpunkt ist Osten, also rechts, und wieder wird entgegen des Uhrzeigersinns ggf. ein weiteres Label generiert.

Weiterhin werden die Nord-/Süd-Positonen im Abstand einer Labelhöhe und die Ost-/West-Positonen im Abstand einer Labelbreite vom dazugehörigen Knoten positioniert. Dies wird gemacht, um Überdeckung mit den zuvor im 4-Position-Model generierten Positionen zu vermeiden, da, wenn zuvor das 4-Position-Model schon keine Position finden konnte, das 8-Position-Model ohne den Abstand dann wahrscheinlich ebenfalls keine Position finden würde.

Das führt dazu, dass vor allem die Ost-/West-Position eher weit entfernt von dem korrespondieren Knoten sind (da Labeltexte bzw. Namen in der Regel eher breit als hoch sind). Die Kartografie verwendet diese Positionen nicht. ZITAT BITTE

In der Implementation wurde dieses Problem dadurch behoben, dass eine "Hilfslinie", als visueller Indikator, das Label mit seinem entsprechenden Knoten verbindet, sodass die Zugehörigkeit schneller ersichtlich wird. Diese Technik wird im Spiral-Model nochmals angewandt, da hier dasselbe Problem besteht.

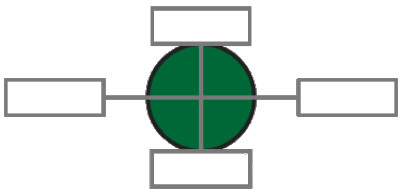

Abbildung 2: Labelpositionen im 8-Position-Model von ausgehenden vom dazugehörigen Knoten

2.1.3. Slider-Model

Das Slider-Model ist ein aufwendigeres Verfahren als die vorhergehenden beiden. Die Idee hier ist, die Eckpunkte des 4-Position-Model zu nehmen und dann Labelpostionen dazwischen zu generieren, in dem man das potentielle Label stückweise zum nächsten Eckpunkt verschiebt. Der Inkrement-Wert zum Verschieben ist konstant (siehe Magic Constants).

Wie auch beim 4-Position-Model in die erste Position die Nord-Ost-Ecke und von dort aus wird parallel zur y-Achse die Position verändert bis die Süd-Ost-Ecke erreicht wird. Da angekommen wird dann parallel zur x-Achse in Richtung der Süd-West-Ecke verschoben, usw.

Das Verschieben erfolgt hier offensichtlich im Uhrzeigersinn, um dem Trend der vorherigen beiden Verfahren entgegenzuwirken.

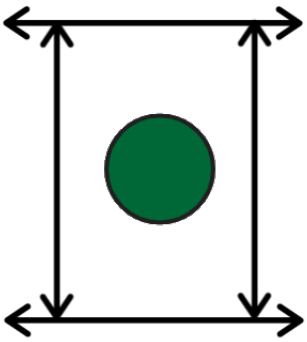

Abbildung 3: Labelpositionen im Slider-Model von ausgehenden vom dazugehörigen Knoten

2.1.4. Spiral-Model

Dieses Verfahren funktioniert gänzlich anders bisher genannten, da diese versuchen, Labelpositionen *karthesisch* zu finden, also durch Verschiebungen bezüglich der x, y-Koordinaten ausgehend vom Mittelpunkt des entsprechenden Knotens. Das Spiral-Model versucht, Labelpositonen *polar* zu finden:

$$s(m) = \begin{pmatrix} d \cdot \cos(2\pi\sqrt{\frac{m}{m_{max}}} \cdot c) \\ \sin(2\pi\sqrt{\frac{m}{m_{max}}} \cdot c) \end{pmatrix} \cdot \sqrt{\frac{m}{m_{max}}} \cdot r, \quad m \in \{1, \ldots, m_{max}\}$$

Das heißt, potentielle Labelposition werden hier spiralförmig generiert, also in ansteigendem Abstand vom dazugehörigen Knoten und in fairer Verteilung hinsichtlich der Richtung der Labels, da die Spirale auch den Winkel fortwährend inkrementiert. Auch ist die Idee wieder, den Trend von zuvor zu durchbrechen und auf eine unkonventiellere Art Labelposition zu suchen in Regionen, die zuvor noch nicht beachtet wurden.

Die Parameter der Gleichung beeinflussen die Form und Ausrichtung der Spirale:

- d: Orientierung der Spirale (im/gegen den Uhrzeigersinn), ($d \in \{-1, 1\}$)
- c: Krümmung bzw. Anzahl der Rotation der Spirale, ($c \in \mathbb{N}$)
- m_{max}: Maximalanzahl der Punkte in der Spirale, ($m_{max} \in \mathbb{N}$)
- m: Der jeweilige m-te Punkt der Spirale

Ergänzend zu den vorherigen Verfahren ist es sinnvoll, auch das Spiral-Model zu nutzen, da hier schnell Abstand vom dazugehörigen Knoten gewonnen werden kann. Dies ist wichtig, da die Verfahren zuvor vorrangig in der Nähe des Knoten versuchen, eine Position zu finden. Das Spiral-Model wird aber nur genutzt, wenn die Verfahren dabei erfolglos geblieben sind.

Wie auch beim 8-Position-Model wird, ob des erweiterten Abstands zum korrespondieren Knoten, eine Hilfslinie genutzt, um die Verbindung für gefundene Labelpositionen und ihre Knoten deutlich zu machen.

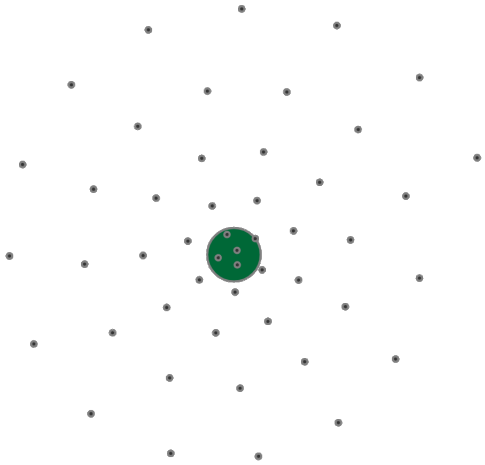

Abbildung 4: Labelpositionen im Spiral-Model von ausgehenden vom dazugehörigen Knoten. Parameter im Beispiel: $m_{max} = 50, d = 1, c = 6, r = 500$

2.2. Kantenlabeling

Im Gegensatz zum Labeling der Knoten werden beim Kantenlabeling einige Einsparung vorgenommen. Der Grund dafür liegt zum ersten in der Natur von Graphen selbst, da diese für n Knoten bis zu n^2 Kanten haben können. Bei einem großen und dichten Graphen wären das schlichtweg zu viele Labelpositonen, wenn man so aufwendige Verfahren wie bei den Knoten verwendet.

Ein weiterer Grund ist speziell für `iGraph.js`: Die Daten, die für Kanten zur Verfügung stehen, sind in der Regel nur Integer-Werte. Da diese Werte ohne Kontext relativ nichtssagend sind, werden Kanten stattdessen mit den Namen der durch sie verbundenen Knoten benannt. Somit gibt die Kante Auskunft über den Nachbar, selbst wenn dieser selbst nicht zu sehen ist, weil er z. B. außerhalb des Bildschirms ist oder keine Labelposition für diesen Knoten gefunden werden konnte.

Bei den Kanten wird auf eine Vorsortierung, wie es bei Knoten, verzichtet, vor allem aus Effizienzgründen, aber auch aus fehlender Notwendigkeit, da Kanten tendenziell ergänzend zum Knotenlabeling sind.

Kanten werden mit einer kleineren Schriftgröße gelabelt und erst, wenn das Knotenlabeling abgeschlossen ist und dann noch freie Kapazität vorhanden ist, das heißt, die festgelegte Maximalanzahl an gezeigten Labels noch nicht überschritten wurden (siehe Konfiguration) und der Zoom Mode nicht aktiv ist.

Das Kantenlabeling verzichtet zur Vereinfachung der Conflict Detection und Bounds) auf den gängigen Standard der Kartografie, die Labels parallel zu der Kante auszurichten. Aus den oben genannten Gründen ist das Kantenlabeling aber sowieso nebensächlich, weswegen diese Entscheidung für die Performance und gegen die Ästhetik in diesem Fall gerechtfertigt ist.

2.2.1. Midpoint

Der Startpunkt für eine potentielle Labelposition auf Kanten ist grundsätzlich der Mittelpunkt der Kante, das heißt, das Label befindet sich in gleichmäßigem Abstand zu seinen Knoten und der Mittelpunkt des Labels liegt auf dem der Kante. Die Ausrichtung ist, wie auch bei den Knoten, parallel zur x- bzw. y-Achse.

2.2.2. Interpolation

Weitere potentielle Labelpositionen werden entlang der Kante interpoliert bzw. das Label wird entlang der Kante in gleichmäßigem Abstand verschoben. Ob dieses Verfahren verwendet werden soll und wenn ja, wieviele Positionen auf der Kante interpoliert werden sollen, sind als Magic Constants in `consts.js` definiert. Falls der Mittelpunkt der Kante selbst wieder ein Ergebnis der Interpolation ist, wird er übersprungen, da er zuvor bereits erfolglos als Labelposition getestet wurde und nur dann die Interpolation überhaupt angewandt wird.

3. Conflict Detection und Bounds

Bevor Labels generiert werden, sollte sichergestellt sein, dass die Knoten und Kanten überhaupt sichtbar sind, um unnötige Berechnung zu ersparen. Wenn ein potentielles Label zu einem tatsächlichen Label wird, sollte sichergestellt sein, dass es sich nicht mit anderen tatsächlichen Labels oder Knoten überdeckt. Um diese Probleme zu lösen, werden Bounding Boxes genutzt:

3.1. Bounding Boxes

Eine Bounding Box ist ein Rechteck um die minimalen und maximalen Ausdehnungen eines Objekts in den x, y-Koordinaten, das somit Grenzen angibt, in dem das Objekt auf jeden Fall vollständig liegt.

Bounding Boxes wurden genutzt um die Bounds, also die Bildschirmgrenze, zu implementieren. Weiterhin fanden zu Anwendung als Repräsentation der Knoten und der Labels, sodass mit simplen Vergleichen eine Überdeckung ausgeschlossen werden kann. Eine weitere Nutzung findet sich im Zoom Mode zum Ersparen vieler unnötiger Berechnungen.

3.1.1. Bounds Checking

Die Bounds werden hier durch eine große Bounding Box repräsentiert. Für jeden Knoten und für jede Kante muss in jeder Iteration des Render-Loops kontrollieren werden, ob sie komplett sichtbar sind, das heißt, sich komplett in-bounds befinden. Außerdem muss dies für generierte Labels bzw. ihre Positionen kontrolliert werden. Das ist dann der Fall, wenn für das Objekt folgendes gilt:

Sei a das Objekt und b die Bounds. a, b sind Bounding Boxes:

$$in = a_{xmin} > b_{xmin} \wedge a_{ymin} > b_{ymin} \wedge a_{xmax} < b_{xmax} \wedge a_{ymax} < b_{ymax}$$

Für die Knoten bleibt die Sortierung nach der Größe der Radien auch nach dem Filtern via Bounds Check erhalten, sodass der wichtigste, derzeit sichtbare

Knoten als erstes versucht wird zu labeln. Um aus dem Versuch, als potenziellen Labelpositionen, tatsächliche Labelpositionen zu machen, muss überprüft werden, dass diese sich nicht mit Knoten schneiden (oder später mit schon gesetzten, also tatsächlichen, Labelpositionen).

3.1.2. Label Conflicts

Da ein Greedy-Ansatz genutzt wurde, wird, sobald eine potenzielle Labelposition konfliktfrei (cf) ist, diese auch sofort genutzt - auf die Gefahr hin, dass dies Labelpositionen von anderen Objekten blockiert und gegebenenfalls ein besseres Labeling nicht erreicht wird.

Für eine generierte Position, wie in Labeling beschrieben, wird nun zur Konfliktermittlung eine Bounding Box erzeugt, also das Label ëingerahmt". Für diese Bounding Box wird jetzt kontrolliert, ob sie sich mit keiner Bounding Box von allen sichtbaren Knoten und keiner Bounding Box von gegebenenfalls schon erzeugten Labels überdeckt. Das heißt, für alle Vergleich muss gelten: Die Bounding Box des potenziellen Labels ist total links von, rechts von, über oder unter der liegen mit der sie verglichen wird (also des Knotens oder tatsächlichen Labels):

Sei a die potenzielle Bounding Box und B die Menge der Bounding Boxes von Knoten und tatsächlichen Labels:

$$cf \leftrightarrow \forall b \in B : a_{xmax} < b_{xmin} \vee a_{xmin} > b_{xmax} \vee a_{ymax} < b_{ymin} \vee a_{ymin} > b_{ymax}$$

So können sukzessive potenzielle Labelpositionen überprüft und, falls konfliktfrei, zu tatsächlichen Labels umgewandelt werden, die dann im weiteren Verlauf selbst Berücksichtigung bei der Konfliktermittlung finden.

3.2. Quadtree

Da sich während des Prozess der Labelgeneration und Konfliktermittlung die Positionen der Knoten nicht verändern, wäre es sehr ineffizient, immer wieder alle Knoten (und deren Labels, sofern generiert) abzufragen.

Da die Aufteilung der Knoten im Raum gleich bleibt (innerhalb einer Iteration des Render-Loops), ist es sinnvoller, diese einmalig zu Beginn in einem Quadtree zu sortieren bzw. zu strukturieren.

Im Vergleich zu der naiven Methode mit einer Komplexität von $O(n^2)$ (da n-mal Knoten mit $n-1$ anderen Knoten verglichen werden) befindet man sich mit dem Quadtree in der Komplexitätsklasse $O(n \log n)$, vorausgesetzt der Baum ist gut balanciert. ZITAT BiTTE

Das Vorgehen des Quadtrees ist simpel: Initial startet man mit dem gesamten Bildschirm und allen Knoten. Nun ermittelt man den Mittelwert der x-Werte der Knoten und dasselbe analog für die y-Werte. Diese beiden Mittelwerte teilen nun als Geraden den Bildschirm in vier nicht zwangsläufig gleich große Teilbereiche.

Diese Teilbereiche haben annähernd die gleiche Anzahl an Knoten, im Idealfall $\frac{n}{4}$ der Knoten des Gesamtbereiches. Nun wird für diese Teilbereiche rekursiv das gleiche Verfahren angewandt wie eben für den gesamten Bildschirm beschrieben. Rekursionsabbruch ist, wenn die Teilbereiche weniger als zwei Knoten enthalten oder die maximale Rekursionstiefe erreicht ist (siehe Magic Constants).

Ein Sonderfall stellen Knoten dar, durch die die Trenngerade der Bereiche verläuft (da es sich hier nicht um Punkte handelt). Diese müssen dann in beiden Teilbereichen, auf jeder Seite der Trenngerade, als enthalten gezählt werden.

Das ist eine Schwachstelle des Quadtree in Situation, in denen die Knoten sehr stark aufeinander sind, z. B. bei starkem Zoom Out oder wenn die Lens Funktion von `iGraph.js` verwendet wird.

Da die Knoten dann fast identische Positionen haben, gelingt keine Aufteilung der Knoten in Teilregionen und alle Knoten sind in nahezu allen Regionen. Das führt den logarithmischen Charakter des Baums ad absurdum.

Der Quadtree erkennt selbst anhand eines Grenzwertes (Magic Constants), ob er ineffizient ist. Eine Lösung dafür wird ist ein Zoom Mode, der nachfolgend erklärt wird. Sollte auch das nicht möglich sein, werden die Knoten in den ineffizienten Teilregionen (also die den Grenzwerte von x Elementen pro Teilregion überschreiten) selbst als ineffizient markiert und werden beim Labeling übersprungen (obwohl sie sichtbar sind). Die Chance, die freie Kapazität bis zur Maximalanzahl an gezeigten Labels zu nutzen, wird dann anderen, weniger relevanten (laut Vorsortierung), dafür effizienten Knoten gelassen.

In den meisten Fällen ist der Quadtree aber effizient und das Ergebnis sind viele kleine, unabhängige Teilbereiche, in denen nur sehr wenige Knoten enthalten sind. Wenn nun ein potenzielles Label überprüft werden soll, so wird geguckt, welche Teilregionen des Quadtrees es überdeckt.

Mit allen Knoten, die in diesen überdeckten Regionen enthalten sind, wird dann eine individuelle Konfliktermittlung durchgeführt. Das heißt, es werden nur Konflikte in der Nähe der potenziellen Labelposition direkt kontrolliert und der Rest wurde eliminiert durch den rekursiven Abstieg (mit Zugriffszeit $O(\log n)$) im Quadtree mit Grenzen, also der Bounding Box, des Labels.

Bleibt die potenzielle Labelposition (nach der Beschreibung aus Label Conflicts mit allen Elementen der überdecken Teilbereiche) konfliktfrei, so kann es zu einem tatsächlichen Label gemacht werden. Dieses wird als neues Element in die eben überprüften Teilregionen übernommen, sodass es zukünftig auch im Quadtree repräsentiert ist und bei der Konfliktermittlung berücksichtigt wird.

4. Adaptation/Optimierung

Mit dem Quadtree wurde schon eine erste Form der Optimierung in diesem Projekt vorgestellt. Diese ist allerdings sehr generisch und in der Computergrafik vielseitig eingesetzt.

Die Idee, aufwendige Berechnungen vorher zu erkennen und einzusparen bleibt auch bei nächsten Optimierungen, jedoch sind diese spezieller auf das Projekt bzw. Graph Labeling zugeschnitten und es geht hier um die Balance zwischen visuell ästhetischem und trotzdem performantem Labeling, also einen Trade-Off zwischen diesen beiden Faktoren. Ein Quadtree ist immer besser als naive, da besteht kein Trade-Off. Insofern sind die

Tendenziell gilt: Je mehr Element potentiell zu labeln sind und je dichter diese Elemente beieinander sind, desto aufwendiger wird das Labeling und sollte zu Gunsten der Performance vereinfacht werden.

4.1. Zoom Mode

Wenn die Element zu dicht beieinander oder sogar aufeinander bzw.überdeckend sind, kann es sind, dass der Quadtree ineffizient wird. Dies ist vor allem der Fall, wenn man weit rauszoomt. Dann ist der Graph quasi auf einem Punkt und um ihn herum ist viel ungenutzter Raum.

Ob der Zoom Mode nutzbar ist, wird kontrolliert, wenn der Quadtree sich als ineffizient deklariert. Hierfür werden von allen Knoten minimalen und maximalen x- und y Werte ermittelt und aus diesen eine große Bounding Box generiert, die dann den Graph vollständig enthält.

Diese Bounding Box des Graphen wird mit den Bounds verglichen und es wird überprüft, ob oberhalb oder unterhalb der Graph-Bounding Box noch wenigstens eine Labelhöhe Platz ist oder ob sich links oder rechts davon noch wenigstens eine Labelbreite ungenutzter Raum befindet.

Findet sich an wenigstens einer Seite solch ungenutzter Raum, dann wird mittels Spiral Model versucht, eine Position in diesem Raum zu finden. Wird eine solche Position gefunden, ist sie in-bounds und hat keine Überdeckung mit der Bounding-Box des Graphen.

Spiral Model wird verwendet, da dies der einzige Algorithmus ist, der sich inkrementweise von seinem Ursprung (also dem korrespondieren Knoten) entfernt und man mit der Position des Labels außerhalb der großen Bounding Box des Graphen landen muss. Die wäre mit den anderen Algorithmen nicht zuverlässig möglich. Wie bei Spiral Model üblich werden die tatsächlichen Labels mit einer Hilfslinie zum Knoten verbunden.

Tatsächliche Labels werden hier nicht in den Quadtree einsortiert, da er im Zoom Mode nicht genutzt wird. Stattdessen werden sie sich direkt gemerkt. Der Performancegewinn hier ist, dass alle Knoten als eine Bounding Box dargestellt werden, und man somit im Prinzip nur Labelpositionen miteinander, aber nicht mit Knoten vergleichen muss.

In der Praxis ist die Anzahl der gelabelten Knoten konstant (siehe Magic Constants) und nur ein Bruchteil der zu sehenden Knoten, sodass die quadratische Komplexität hier noch nicht zu Performanceeinbußen führt.

Kantenlabeling findet im Zoom-Mode nicht statt, da die Kanten praktisch von den Knoten im Zoom überdeckt werden und hier auch Berechnungen eingespart werden können.

4.2. Performance Modes

Performance Modes bilden ein Zusammenspiel aus der Anzahl der sichtbaren Knoten, also derjenigen, die in-bounds sind, und den genutzten Labeling Verfahren. Allgemein gesagt: Je mehr Knoten sichtbar sind, desto weniger Verfahren werden genutzt - und die genutzten Verfahren sind die einfacheren Verfahren.

Dies wird durch Grenzwerte realisiert (siehe Magic Constants). Die Grenzwerte (siehe Figure 5) wurden vor allem durch manuelles Testen ermittelt. Ziel war es das ein Labelingdurchlauf im Render-Loop möglichst unter 50 ms bleibt.

Minimum	Maximum	Verfahren
1	99	4-Position-Model, 8-Position-Model, Slider-Model, Spiral-Model
100	499	4-Position-Model, 8-Position-Model, Slider-Model
500	999	4-Position-Model, 8-Position-Model
500	∞	4-Position-Model

Abbildung 5: Intervalle wie standardmäßig im Projekt festgelegt (siehe Magic Constants) mit dazugehörigen Labeling Verfahren

Hier ist der Zweck, dass man bei sehr vielen sichtbaren Knoten durch das Labeling eine schnelle, grobe Orientierung bekommen kann. Danach kann man via Zoom-In den gewünschten Bereich genauer explorieren, wofür dann auch ein aufwendiges und somit detailliertes Labeling wünschenswert wäre. Da dann auch weniger Knoten zu sehen sind, was zu weniger Vergleichen führt, ist hier genug Zeit für solche komplizierten Berechnung.

5. Visualisierung

Wenn ein potentielles Label nach Feststellen von Konfliktfreiheit zu einem tatsächlichen Label wird, so ist es für das Backend (wie in Conflict Detection und Bounds beschrieben) relevant, dieses Label von nun an in der Konfliktermittlung zu berücksichtigen. Für das Frontend gilt es dann, das Label an der gefundenen, freien Position zu zeichnen und animieren, sodass es für den Nutzer angenehmer anzuschauen ist.

5.1. Drawing

Wenn einem für eine Labelposition zugesichert werden kann, dass diese konfliktfrei ist, dann wird der Text bzw. Name des Labels ausgelesen und an eben diese Position in dem 2D-Canvas zu dem Graphen und ggf. anderen Labels dazugezeichnet.

Ein Sonderfall stellen Label dar, die mit 8-Position-Model oder Spiral-Model generiert wurden, da hier eine Hilfslinie zur Kenntlichmachung der Zusammengehörigkeit von Knoten und Label zusätzlich gezeichnet wird.

Diese Labels enthalten daher weitere Informationen über die Linie und die Seite des Labels, zu der diese Linie verbindet. Die Seite ist immer entgegengesetzt zur Richtung in der das Label gefunden wurde, also wenn das Label unterhalb des Knoten (Süd-Position) ist, verbindet die Hilfslinie Knoten und Oberseite des Labels. Die Hilfslinie endet immer in der Mitte der Seite. Nur die Seite mit der die Linie verbindet, wird dann zur weiteren Verdeutlichung selbst auch als Linie gezeichnet.

5.2. Animation

Die Animation finden immer dann statt, wenn sich der Sichtbarkeitszustand von Labels im Vergleich zur vorherigen Iteration des Render Loops ändert oder es nach wie vor sichtbar ist, aber die Position des Labels sich geändert hat. Das passiert, wenn in der neuen Iteration für ein anderes, wichtigeres Label diese Position gefunden wurde, dann im Nachhinein aber noch für das unwichtigere Label eine andere Position gefunden werden kann. Wäre dies nicht der Fall, würde sich der Sichtbarkeitszustand zu nicht sichtbar ändern und das unwichtigere Label würde ausgeblendet werden (siehe Figure 6).

Hierfür wird die Klasse aus Animatable.js aus iGraph.js verwendet, die dort auch schon die Animation der Knoten händelt. Somit werden Knoten und Labels durch dieselben Funktionen animiert.

	sichtbar$_t$	nicht sichtbar$_t$
sichtbar$_{t-1}$	animate	fade out
nicht sichtbar$_{t-1}$	fade in	–

Abbildung 6: Zustände von Labelsichtbarkeit vor $(t-1)$ und nach (t) dem Update der Positionen und die entsprechenden Animationsfunktionen

5.2.1. Fade In / Fade Out

Diese Funktion verändern lediglich die Alpha-Werte, also die Transparenz, der Labels und ggf. der dazugehörigen Linien. Für das Ausblenden werden sukzessive die Alpha-Werte dekrementiert, zum Einblenden werden sie sukzessive erhöht.

5.2.2. Animate

Da hier vor und nach dem Update das Label zu sehen ist, muss es verschoben werden. Dabei kann es passieren, dass es kurzzeitig zu Überdeckung des von Labels und Knoten kommt. In der finalen Position, nach der Animation, allerdings nicht mehr, da diese Position konfliktfrei ist.

Die Bewegungen bzw. die Animationen sind nicht rein linear, was das Labeling flüssiger, interaktiver und "lebendigerërscheinen lässt, auch wenn es zusätzliche Berechnungen erfordert. Diese können angestellt werden, da zuvor auf Effizienz geachtet wurde.

6. Konfiguration

Die Konfiguration des Labeling kann über zwei Wege erfolgen:

Der erste ist der Üser Modeünd erfolgt über ein minimales GUI, das bereits durch `iGraph.js` vorhanden war. Neue Funktionalität wurden einfach in zusätzliche Buttons implementiert.

Der zweite Weg ist der Äuthor Mode". Es existiert ein File `consts.js`, in denen alle im Projekt genutzten Magic Constants in einem Objekt zentral definiert

sind und somit auch dort abgeändert werden könnten. Die Standardeinstellungen sind allerdings getestet, auf `iGraph.js` zugeschnitten und haben sich über Monate etabliert. Bei Änderung könnten ungewollte Konsequenzen und Bugs auftreten.

6.1. GUI

Im Gegensatz zu Änderungen der Magic Constants ist das GUI sicher. Es beachtet festgelegte Minimal- und Maximalwerte und sorgt dafür, dass diese nicht unterschritten bzw. überschritten werden (sofern die Standardwerte in `consts.js` verwendet werden).

Die durch das GUI zu ändernden Parameter sind:

- Label anzeigen (Toggle)
- Anzahl angezeigte Labels erhöhen
- Anzahl angezeigte Labels reduzieren
- Schriftgröße erhöhen
- Schriftgröße reduzieren

Weiterhin gilt zu sagen, dass für die Anzahl der gezeigten Labels im Zoom Mode weniger Labels angezeigt werden als sonst. Um hier filigraner einstellen zu können, erfolgt das De-/Inkrementieren der Anzahl der gezeigten Labels in Einserschritten. Für den normalen Modus werden mehrere Labels auf einmal hinzugefügt bzw. gelöscht.

Um Berechnungen für große Graphen zu reduzieren sind geringe Labelanzahl und Schriftgröße eines der besten Werkzeuge, sollten allerdings vom User eingestellt werden können - daher ein GUI.

6.2. Magic Constants

Ursprünglich zum vereinfachten internen Testen und zur Strukturierung des Codes gedacht, ist eine Sammlung der Magic Constants gegebenenfalls auch für externe Nutzer interessant. Viele Parameter, die über das GUI nicht verstellt werden können sind hier konfigurierbar, beispielweise:

- Schriftart und -farbe
- Interpolationsschritt für Kanten (siehe Kantenlabeling)
- Ineffizienzgrenzen der Teilregionen des Quadtree (siehe Quadtree).
- Parameter der Spirale (siehe Spiral-Model)
- Grenzwerte für die Performance Modes (siehe Figure 5)

Änderung an den Werten können immer dazu führen, dass das Programm gar nicht funktioniert oder zumindest nicht wie erwartet. Für den Nutzung von gebräuchlichen Werten sollten allerdings maximal die Performance schlechter werden, Funktionalität aber erhalten bleiben.

Möglicherweise gibt es auch noch bessere Einstellungen hinsichtlich der Performance, die visuell gleichermaßen ansprechend sind wie die Standardeinstellungen.

7. Über das Projekt

7.1. Technische Details und Umsetzung

Das gesamte Projekt wurde in JavaScript umgesetzt, da die Basis, `iGraph.js`, schon in JavaScript geschrieben war. Es ist maximal modular zur Basis, das heißt, bis auf eine zusätzliche Zeile im Update-Loop (und die GUI-Elemente) wurden am Quellcode der Basis keine Veränderungen vorgenommen.

Der Code Style ist ES6-konform. Es wurden Klassen genutzt, um die einzelnen, logischen Bereiche auch im Code entsprechend zu trennen. Manche Klassen, wie z. B. `BBox.js`, sind eher Namespaces für Funktionen mit entsprechendem Zuständigkeitsbereich. Daher sind diese Funktionen dann statisch.

Ausnutzung von sprachlichen Besonderheiten oder Datenstrukturen wurden in folgenden Bereichen vorgenommen:

7.1.1. Maps

Die Knoten und Kanten werden in Maps gespeichert. Man kann sie eindeutig durch eine ID zuordnen (die von `iGraph.js` selbst kommt, im Falle von Kanten) oder einfach durch den Namen des Labels (im Falle von Knoten). So erhält man eine schnelle Zugriffszeit von $O(1)$ für das Updaten der Werte. Bei den Knoten wird in der Map die Sortierung erhalten, sodass die Map im weiteren Verlauf einfach zum Array konviertert werden kann.

7.1.2. Generator Functions

Generator Functions werden genutzt, um den Greedy Approach des Projektes zu realisieren. Die Elemente werden einzeln "yieldedünd weiterverarbeitet. Zum Beispiel kann so eine Funktion für alle potentiellen Labelpositionen angebeben werden, wenn sie aber als Generator Function gekennzeichnet ist, kann man Finden einer tatsächlicher Labelposition einfach returnen und die restlichen Positionen werden nicht berechnet.

7.2. Async Functions

Wo möglich werden asynchrone Funktionen benutzt. Zum Beispiel ist das Zeichen und Animieren der Labels unabhängigen vorherigen Schritt und kann somit asynchron erledigt werden, sobald die Labelposition konfliktfrei ist.

7.3. Bekannte Bugs

Beim Testen wurden die meisten Bugs gefixed - ein selten auftretender Bug ist übrig geblieben: "Flying Labels".

Manchmal kann es vorkommen, dass beim Fade Out von Labels diese von ihrer Position "wegfliegenïn eine Ecke des Bildschirms. Dies fällt nur auf, wenn enorm viele Berechnungen stattfinden, also die Animationen langsamer sind.

Eine weitere ungünstige Konsequenz die aus der Erpichtheit auf Performance entstand: Wenn alle sichtbaren Knoten als ineffizient markiert werden (und somit nicht gelabelt werden) und kein Zoom Mode möglich ist, z. B. wenn man Radar und Lens Funktion von `iGraph.js` nutzt, dann wird gar nichts gelabelt.

Diese Situation sind aber prakisch unmöglich, man muss sie sehr gezielt erzwingen.

7.4. Verbesserungsvorschläge und Anregungen

Die Grundlage des Labeling wurde durch dieses Projekt erstellt. Auf der Seite der Performance wurden viele Techniken umgesetzt und Möglichkeit genutzt um unnötige Berechnungen zu ersparen. Hinsicht der Visualisierung können aber vor allem noch Verbesserungen vorgenommen werden:

7.4.1. Verbessertes Kantenlabeling

Die Kantenlabels sollten ebenfalls nach kartografischen Standard gezeichnet werden. Das heißt, parallel zur entsprechenden Kante. Dafür wären zum einen verbesserte Animationen nötig, um die Labels zu rotieren. Des Weiteren eine verbesserte Konfliktermittlung, da dann nicht mehr garantiert werden kann, dass die Kantenlabels parallel zur x- und y-Achse sind.

7.4.2. Verbesserte Repräsentation der Knoten als Kreis

Das würde die Labelqualität verbessern, da die Labels noch näher an den Knoten wären und dementsprechend auch weniger Platz an jedem Knoten verschwenden (die Ecken der Bounding Box, die nicht Teil der Knotens sind).

Auch hier wäre eine verbesserte Konfliktermittlung, da die Knoten dann als Kreis statt als Bounding Box repräsentiert würden. Zusammen mit dem ersten Verbesserungsvorschlag müsste man dann nicht Rechteckee parallel zur x- und y-Achse, sondern stattdessen beliebig orientierte Rechtecke und Kreise vergleichen, was wesentlich aufwendiger ist.

7.4.3. Konservativere Animationen

Zuweilen sind die Animationen sehr aktiv, was zu viel Unruhe im Labeling führt. Gegebenfalls sollte man eine Toleranzzeit einführen, sodass sich Labels kurzzeitig überlappen dürfen, da viele Animationen durch minimale Überdeckungen während der Bewegungen entstehen. Eine solche Toleranzzeit würde das Gesamtbild etwas statischer machen und einfach bei den Magic Constants hinzugefügt werden.

Danksagung

Ich möchte meinem Projektpartner Nico Trebbin für die Zusammenarbeit danken. Wir haben uns viel vorgenommen und viel davon umgesetzt. Weiterhin danke ich Prof. Christian Tominski für das Semester, die Möglichkeit, auf seiner Codebasis aufzubauen und die Tipps, Anregungen und das Feedback während der Meetings. Zu guter letzt möchte ich meiner Freundin Emily Fuhrmann für ihre Unterstützung in jeder Lage danken.

Literatur

[1] Martin Luboschik, Heidrun Schumann, and Hilko Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 14 (6):1237–1244, 2008. ISSN 1077-2626. doi:10.1109/tvcg.2008.152. URL https://doi.org/10.1109/TVCG.2008.152.