

Bauhaus-Universität Weimar  
Fakultät Medien  
Studiengang Medieninformatik

# Ein Framework zur Entwicklung von Visualisierungen für den Vergleich von Fließtexten mit Differenzalgorithmen

## Bachelorarbeit

Dean Jürges  
geb. am: 25.11.1992 in Salzgitter

Matrikelnummer 110662

1. Gutachter: Prof. Dr. Bernd Fröhlich
2. Gutachter: Prof. Dr. Benno Stein

Datum der Abgabe: 24. März 2016

# **Erklärung**

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, 24. März 2016

.....  
Dean Jürges

## **Kurzfassung**

In dieser Arbeit wurde ein Framework erstellt, mithilfe dessen Visualisierungen für Vergleiche zweier Texte oder mehrerer Revisionen entwickelt und getestet werden können. Dazu stehen verschiedene Textkorpora mit insgesamt über 16.000 Textstellen zur Verfügung, die mit Differenzalgorithmen verglichen werden. Das Framework wurde auf Basis moderner Webtechnologien implementiert. Sowohl Visualisierungen als auch Textstellen sind zur Laufzeit beliebig austauschbar, um so Visualisierungen miteinander zu vergleichen oder sie für unterschiedliche Texte zu testen. Mit einem Parameter-Widget können alle implementierten Visualisierungen im Detail angepasst werden, um die Grenzen und Darstellungsmöglichkeiten zu erproben. Das Framework und alle Visualisierungen wurden als Webkomponenten mithilfe des Google-Projekts Polymer erstellt. Die komponentenbasierte Entwicklung erlaubt es, die Visualisierungen in jeder beliebigen Webumgebung wie HTML-Elemente wiederzuverwenden. Außerdem kann das Framework dadurch sehr leicht durch neue Visualisierungen, Differenzalgorithmen und anderen Funktionen erweitert werden und bietet so eine Umgebung zur Entwicklung und Erforschung von bekannten und neuen Visualisierungen für Textvergleiche.

Ein weiterer wichtiger Beitrag dieser Arbeit ist eine Nutzerstudie, bei der Schüler Texte manuell vergleichen und eigene Visualisierungen zur Präsentation der Gemeinsamkeiten und Unterschiede erstellen sollten. Die Ergebnisse variierten stark und zeigten, dass es keine eindeutige Lösung des Problems der Visualisierung von Textvergleichen gibt. Trotzdem erstellten einige Schüler interessante Konzepte, die zum Teil bereits bekannten Visualisierungen ähnlich sind.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Aufgabenstellung und Zielsetzung . . . . .	3
1.3 Aufbau der Arbeit . . . . .	3
<b>2 Verwandte Arbeiten</b>	<b>4</b>
2.1 Differenzalgorithmen . . . . .	4
2.2 Visualisierungen für Differenzalgorithmen . . . . .	6
2.2.1 Quellcode . . . . .	6
2.2.2 Text . . . . .	8
2.3 Visualisierungen für Textwiederverwendung . . . . .	8
2.3.1 CitePlag . . . . .	9
2.3.2 PlagVis . . . . .	9
2.3.3 Matrixvisualisierungen . . . . .	12
2.3.4 Koordinatensystemvisualisierungen . . . . .	14
2.3.5 Barcodevisualisierung . . . . .	15
2.4 Visualisierungen für Änderungsverfolgungen . . . . .	16
2.5 Abstrakte Visualisierungen - Graphvisualisierungen . . . . .	17
<b>3 Nutzerstudie</b>	<b>20</b>
3.1 Aufbau . . . . .	20
3.2 Ergebnisse . . . . .	21
<b>4 Textvergleichsvisualisierungen</b>	<b>26</b>
4.1 Side By Side . . . . .	26
4.2 Vertical Aligned . . . . .	27
4.3 Classic Diff . . . . .	28
4.4 Two Lines Diff . . . . .	28
4.5 Animation Diff . . . . .	30
4.5.1 Quelle-Plagiat Animationen . . . . .	30
4.5.2 Revision Animationen . . . . .	34

## ***INHALTSVERZEICHNIS***

---

4.6	Multi Revision Browser . . . . .	34
<b>5</b>	<b>Daten und ihre Aufbereitung</b>	<b>38</b>
5.1	Vorverarbeitung . . . . .	38
5.1.1	Tokenisierung mit UIMA . . . . .	39
5.1.2	Sortierung nach Textlänge/ Anzahl Änderungen . . . . .	39
<b>6</b>	<b>Entwicklungsframework für Textvergleichsvisualisierungen</b>	<b>40</b>
6.1	Polymer-Elemente . . . . .	40
6.2	Das Framework . . . . .	42
6.2.1	HTML-Abschnitt . . . . .	42
6.2.2	Data Handling . . . . .	44
6.2.3	Diff-Klasse . . . . .	45
6.2.4	URL . . . . .	46
6.3	Parameter-Widget . . . . .	46
6.4	Visualisierungen . . . . .	47
6.4.1	Vertical Aligned . . . . .	48
6.4.2	Two Lines Diff . . . . .	48
6.4.3	Multi Revision Browser . . . . .	50
6.4.4	Animatoren . . . . .	53
<b>7</b>	<b>Zusammenfassung und Ausblick</b>	<b>57</b>
	<b>Literaturverzeichnis</b>	<b>59</b>

# Kapitel 1

## Einleitung

In diesem Kapitel werden die Motivation und Ziele der Arbeit beschrieben. Anschließend wird der gewählte Aufbau erklärt.

### 1.1 Motivation

Der Vergleich zweier oder mehrerer Fließtexte, um inhaltliche Ähnlichkeiten festzustellen, ist eine Aufgabe, die in zwei Hauptbereichen Anwendung findet. Beim kollaborativen Schreiben arbeiten mehrere Personen an dem gleichen Text. Beispielsweise verändern Lektoren und Autoren eines Artikels diesen mehrfach, um Fehler zu korrigieren, Abschnitte hinzuzufügen und zu löschen. Dadurch entstehen mehrere Versionen desselben Textes, die inhaltlich verglichen werden müssen, um eine vereinigte Version zu erzeugen. Interessant in diesem Zusammenhang ist die Beobachtung von Artikeln auf Nachrichtenwebsites, die nach der erstmaligen Veröffentlichung oft noch bearbeitet werden. Ein Vergleich aller Versionen (im Folgenden auch „Revisionen“) eines Artikels kann beispielsweise zeigen, ob der eigentliche Sinn des Artikels über die Zeit verändert wurde. Aber auch Korrekturen und Verschiebungen im Text können interessante Aufschlüsse liefern.

Ein weiterer Anwendungsfall für Textvergleiche ist die Erkennung von Textwiederverwendung. Ein Beispiel dafür ist die Suche nach Plagiarismus in wissenschaftlichen Arbeiten. In den letzten Jahren wurden immer mehr Fälle, wie die von Karl-Theodor zu Guttenberg<sup>1</sup> oder Annette Schavan<sup>2</sup>, öffentlich gemacht. Meistens sind es freiwillige Plagiatsjäger, die mehrere Hundert Sei-

---

<sup>1</sup>Karl-Theodor zu Guttenberg verlor 2011 seinen Doktortitel nachdem seine Dissertation als Plagiat erkannt wurde ([https://de.wikipedia.org/wiki/Plagiatsaffäre\\_Guttenberg](https://de.wikipedia.org/wiki/Plagiatsaffäre_Guttenberg))

<sup>2</sup>Seit 2012 wurde die Dissertation von Annette Schavan öffentlich diskutiert. 2014 wurde ihr Doktorgrad aberkannt. (<https://schavanplag.wordpress.com/>)

ten durchforsten und manuell mit möglichen Originaltexten vergleichen. Eine automatische Erkennung von Ähnlichkeiten und eine übersichtliche Darstellung der Fundstellen können diesen Prozess unterstützen. Auch im Lobbyismus spielt die Wiederverwendungserkennung eine Rolle. So kommt es vor, dass Änderungsvorschläge von Parlamentariern mit Lobbypapieren teilweise übereinstimmen. Durch Textvergleiche kann festgestellt werden, welche Lobbys wie viel Einfluss auf welche Politiker haben. Ein wichtiger Teil der Erkennung von Textwiederverwendung ist die Präsentation der Ergebnisse. Die Sammlung aller Fundstellen muss so aufbereitet und dargestellt werden, dass Dritte sie effizient verifizieren können.

Der Vergleich zweier Texte beinhaltet eine umfangreiche und anstrengende kognitive Leistung. Beide Texte müssen gelesen und inhaltlich verstanden werden. Außerdem muss sich der Betrachter merken, an welchen Stellen im Text bestimmte Sätze stehen, um sie eindeutig denen des anderen Textes zuordnen zu können. Das Finden von Gemeinsamkeiten erweist sich besonders dann als schwierig, wenn sie an unterschiedlichen Stellen im Text stehen oder die Texte verschieden strukturiert sind, durch anders gesetzte Absätze beispielsweise. Diese Aufgabe wird umso schwieriger, je länger die zu vergleichenden Texte sind. Auch eine höhere Komplexität der Unterschiede erschwert das Problem. So ist es zum Beispiel schwierig, kleine Unterschiede in sonst gleichen Abschnitten zu finden oder Gemeinsamkeiten zu identifizieren, die in langen Textabschnitten untergehen. Außerdem wird die Aufgabe mit einer steigenden Anzahl von miteinander zu vergleichenden Texten sehr viel umfangreicher. Die klassischen Hilfsmittel, derer sich meist bedient wird, sind Textmarker und Stifte, um sowohl Unterschiede zu markieren als auch Gemeinsamkeiten zwischen den Texten zuzuordnen.

Überraschenderweise gibt es nur wenige digitale Hilfsmittel, um Menschen bei Textvergleichen zu unterstützen. Die meisten verwenden sogenannte Differenzalgorithmen, um zwei Texte miteinander zu vergleichen und visualisieren deren Ergebnisse. Diese Algorithmen sind jedoch optimiert für den Vergleich von Quellcode. Das bedeutet, sie erwarten eine zeilenweise Organisation der Texte, wobei jede Zeile relativ kurz ist. Die Ausdrucksmöglichkeiten von Quellcode gegenüber Fließtexten sind außerdem sehr eingeschränkt, was den Vergleich vereinfacht, da die Zeilen einer relativ eindeutigen Struktur folgen. Weiterhin werden solche Vergleiche meist ausgeführt, um wenige Änderungen zwischen zwei Versionen des gleichen Quellcodes darzustellen. Daher ist es nur bedingt möglich, diese Algorithmen für natürliche Sprache einzusetzen. Fließtexte bestehen zum einen meist aus langen Zeilen, zum anderen können die Veränderungen zwischen zwei Texten schnell drastisch werden. Da die Struktur sich sehr stark ändern kann, ist es schwieriger Verschiebungen im Text zu erkennen. Darüber hinaus erkennen Differenzalgorithmen keine Paraphrasie-

rungen, also wenn das gleiche mit anderen Worten gesagt wird, da der Vergleich nur auf lexikalischer Ebene erfolgt.

## **1.2 Aufgabenstellung und Zielsetzung**

Ziel dieser Bachelorarbeit ist es, die Anwendung von Differenzalgorithmen auf natürliche Sprache zu analysieren. Ein besonderer Fokus liegt dabei auf der Visualisierung der Ergebnisse. Um zu erforschen, welche Grenzen der manuelle Vergleich von Fließtexten und die anschließende Visualisierung der Ergebnisse haben, wurde eine Nutzerstudie mit Schülern der 9. bis 11. Klasse durchgeführt. Anschließend wurde ein Framework entwickelt, mit dem verschiedene Techniken zur Visualisierung von Textvergleichen erforscht werden können. In dieses Framework wurden bereits bekannte Visualisierungen eingefügt. Gleichzeitig wurden diese optimiert und neue Techniken gefunden und erprobt. Die Visualisierungen können mit Texten mehrerer Korpora umfangreich getestet werden. Zwei davon enthalten revisionierte Texte, zwei weitere enthalten Plagiatsstellen und deren Quellen.

## **1.3 Aufbau der Arbeit**

Im folgenden Kapitel 2 werden themenverwandte Arbeiten und Tools besprochen. Anschließend wird in Kapitel 3 eine Nutzerstudie zum Vergleich von drei Revisionen eines Textes beschrieben. In Kapitel 4 werden die Konzepte der hier erstellten Visualisierungen vorgestellt. Kapitel 5 erklärt, welche Textkorpora verwendet wurden und wie die Vorverarbeitung ablief. Kapitel 6 beschäftigt sich mit dem für Browser entwickelten Framework zum Erforschen der Textvergleichsvisualisierungen und deren Implementierung. Kapitel 7 fasst die Arbeit zusammen und gibt einen Ausblick für die Zukunft.

# Kapitel 2

## Verwandte Arbeiten

Dieses Kapitel stellt verwandte Arbeiten vor. Zuerst werden Arbeiten zum Thema Differenzalgorithmen beschrieben, anschließend Visualisierungen derer Ergebnisse. Dann werden Visualisierungen sowohl zur Textwiederverwendung als auch zur Änderungsverfolgung erklärt. Zum Schluss werden noch abstrakte Graphvisualisierungen vorgestellt.

### 2.1 Differenzalgorithmen

Das Problem der Bestimmung der längsten gemeinsamen Teilfolge (engl. Longest Common Subsequence) zweier Folgen bietet die algorithmische Grundlage für Textvergleiche. In der Arbeit von Hunt and Szymanski [1977] wird das Problem beschrieben und ein Algorithmus dazu vorgestellt.

Gegeben sei beispielsweise eine Folge  $A$  bestehend aus Zeichen  
 $A=\{X, M, B, A, C\}$ . Dann ist

$B=\{M, A, C\}$

eine Teilfolge von  $A$ , da alle Zeichen von  $B$  in der gleichen Reihenfolge in  $A$  vorkommen. Dabei müssen die Zeichen der Teilfolge nicht nacheinanderfolgend in der Folge vorkommen, aber die Reihenfolge muss übereinstimmen.

Bei der Suche der längsten gemeinsamen Teilfolge geht man von zwei Folgen aus, beispielsweise

$A=\{Z, U, K, Z, A, P, L, V\}$  und

$B=\{U, B, C, Z, S, P, Y, T, L\}$

dann ist eine gemeinsame Teilfolge

$C=\{U, P\}$ , da sie sowohl eine Teilfolge von  $A$  als auch eine Teilfolge von  $B$  ist.

Die längste gemeinsame Teilfolge wäre in diesem Fall

$D=\{U, Z, P, L\}$ .

In dem Artikel von Myers [1986] wird gezeigt, dass das Problem der längsten gemeinsamen Teilfolge zweier Folgen  $A$  und  $B$  äquivalent zu dem Problem der kürzesten Transformation von  $A$  zu  $B$  ist. Eine Transformation erfolgt, indem Zeichen von  $A$  gelöscht und zu  $A$  hinzugefügt werden, um als Ergebnis  $B$  zu erhalten. Die kürzeste Transformation entspricht der, die am wenigsten dieser Lösch- und Hinzufügeoperationen benötigt.

Der dazu entwickelte Algorithmus wird mit „edit graphs“ beschrieben. In Abbildung 2.1 sieht man das dazugehörige Beispiel für die Folgen  $A=\{A, B, C, A, B, B, A\}$  und  $B=\{C, B, A, B, A, C\}$ .

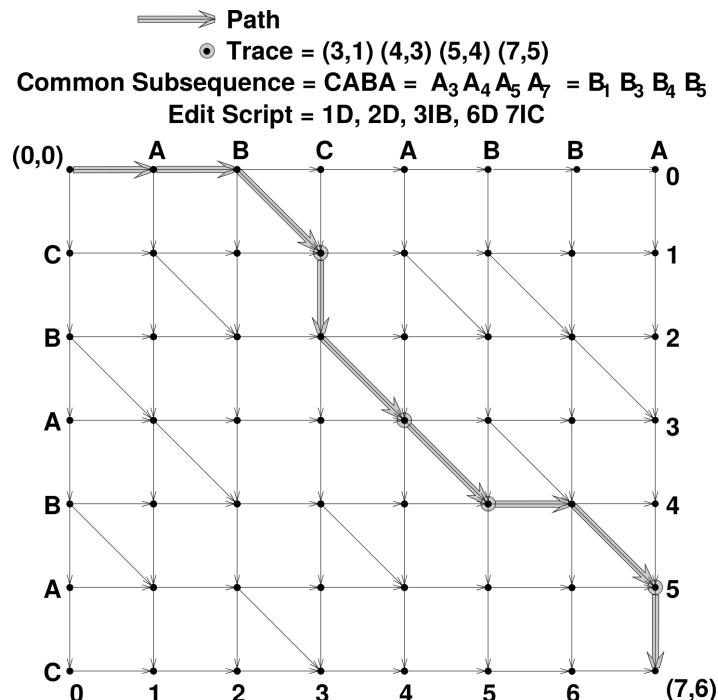


Abbildung 2.1: „Edit graph“ aus der Arbeit von Myers [1986]

Ein „edit graph“ wird vom Prinzip wie eine Matrix aufgebaut, sodass für jedes Zeichen aus  $A$  eine Spalte und für jedes Zeichen aus  $B$  eine Zeile erstellt wird. Alle Knoten des dadurch entstehenden Rasters werden durch horizontale und vertikale Kanten mit ihren Nachbarn verbunden. Zusätzlich werden diagonale Kanten an Knoten gezeichnet, an denen die Zeichen von  $A$  und  $B$  gleich sind. Ein Weg (engl. Path) von oben links  $(0,0)$  nach unten rechts  $(7,6)$  beschreibt die Transformation. Eine horizontale Kante ist äquivalent zum Löschen des Zeichens an der Stelle  $x$  ( $xD$ ). Eine vertikale Kante entspricht dem Einfügen eines Zeichens ( $b_i$ ) der Folge  $B$  an der Stelle  $x$  ( $xIb_i$ ). Da diagonale Kanten nur

an gemeinsamen Zeichen existieren, muss an dieser Stelle keine Bearbeitung stattfinden. Die Transformation von Folge  $A$  im Beispielbild läuft wie folgt ab. Die Zeichen an der ersten und der zweiten Stelle werden gelöscht (1D, 2D). An der dritten Stelle wird ein „B“ eingefügt (3IB). Das Zeichen an der sechsten Stelle wird gelöscht (6D). An der siebten Stelle wird ein „C“ eingefügt (7IC). So ergibt sich die Transformation (in der Abbildung „Edit Script“) 1D, 2D, 3IB, 6D, 7IC. Die gemeinsame Teilfolge ergibt sich durch die gewählten diagonalen Kanten (C, A, B, A).

Die kürzeste Transformation entspricht dem Weg mit der minimalen Anzahl von nicht-diagonalen Kanten. Die längste gemeinsame Teilfolge ist gegeben durch den Weg mit maximaler Anzahl von diagonalen Kanten. Somit sind die beiden Probleme äquivalent zueinander. Die in dieser Arbeit verwendeten Differenzalgorithmen nehmen als Eingabe Strings entgegen, tokenisieren diese anhand von Leerzeichen und weisen jedem Token einen Typen zu, der sich durch die Transformation ergibt. Tokens, die von Folge  $A$  zu  $B$  gelöscht wurden, sind „Deletions“. Tokens, die in  $B$  eingefügt wurden, sind „Insertions“. Und Tokens, die in Folge  $A$  und  $B$  übereinstimmen, sind „Equals“. Anhand dieser Einteilung können die Ergebnisse der Differenzalgorithmen visualisiert werden.

## 2.2 Visualisierungen für Differenzalgorithmen

Die Ergebnisse von Differenzalgorithmen wurden bereits auf unterschiedliche Weise visualisiert. Dabei gibt es Visualisierung, die entweder für den Vergleich von Quellcode oder für den Vergleich von Text ausgelegt sind.

### 2.2.1 Quellcode

Ein häufiger Anwendungsfall ist der Vergleich von Quellcode in Versionsverwaltungen. Um verschiedene Versionen eines Programms zusammenzuführen, wurden eine Reihe von Tools entwickelt, die das Erkennen von Unterschieden beider Versionen vereinfachen sollen. Ein Tool dafür ist Diffuse<sup>1</sup>, das beide Versionen nebeneinanderstellt und die Unterschiede hervorhebt. Die Zeilen sind so ausgerichtet, dass Gleiche nebeneinander steht. Scrollt man in einer Datei, scrollt die andere automatisch mit. Eine Erweiterung dieses Prinzips ist P4Merge<sup>2</sup>, welches drei Versionen der Datei nebeneinander anzeigt und anhand der Gemeinsamkeiten ausrichtet. Dargestellt ist die Basisdatei, die lokale Datei und die Remote-Datei. Diff und Merge Tools dieser Art gibt es viele.

---

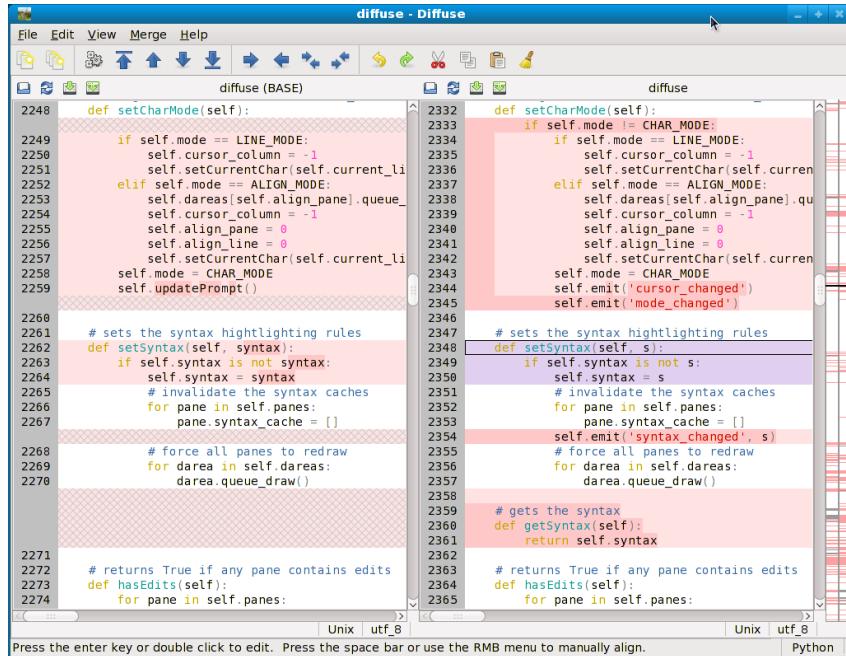
<sup>1</sup><http://diffuse.sourceforge.net/screenshots.html>

<sup>2</sup><https://www.perforce.com/product/components/perforce-visual-merge-and-diff-tools>

## KAPITEL 2. VERWANDTE ARBEITEN

---

Da es für den Nutzer wichtig zu sehen ist, was sich zwischen den Revisionen verändert hat, liegt der Fokus immer auf den Unterschieden.



**Abbildung 2.2:** Diffuse Screenshot: <http://diffuse.sourceforge.net/screenshot.png>

Auf der Website von Github<sup>3</sup> werden ebenfalls die Unterschiede zwischen zwei Revisionen visualisiert. Beide werden in einer Ansicht zusammengefasst.

2	2	-
		+var count = 0;
3	3	for (var i = 0; i != array.length; i++) {
4	4	count++;
5	5	
6	6	- someFunction();
		+ someNewFunction();
7	7	}

**Abbildung 2.3:** Github Screenshot

Zeilen, die aus der früheren Revision gelöscht wurden, sind rot markiert. Hinzugefügte Zeilen sind grün markiert. Finden Veränderung innerhalb einer Zeile statt, steht die alte Version der Zeile direkt über der neuen. Zusätzlich sind die

<sup>3</sup><https://www.github.com>

Änderungen mit einem dunkleren Grün beziehungsweise Rot markiert (siehe Abbildung 2.3).

### 2.2.2 Text

In einer Webapplikation von Weber-Wulff<sup>4</sup> können Texte miteinander verglichen werden. Die Applikation erlaubt es lokale Dateien hochzuladen oder Text direkt einzufügen. Gleiche Abschnitte werden in beiden Texten, die nebeneinander angezeigt werden, mit individuellen Farben markiert (siehe Abbildung 2.4).

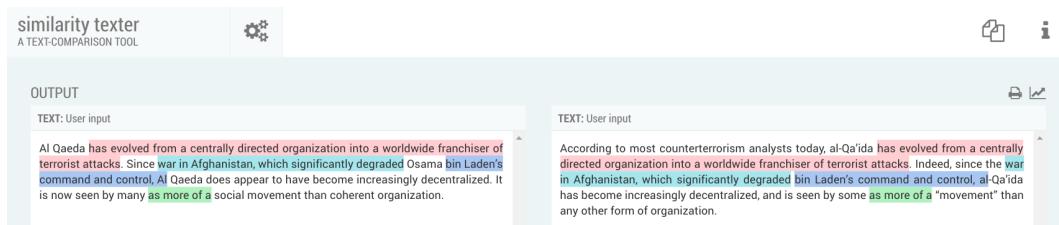


Abbildung 2.4: similarity texter von Weber-Wulff

Webapplikationen dieser Art gibt es viele. Diffchecker<sup>5</sup> zum Beispiel vergleicht zwei Texte mit Fokus auf den Unterschieden. In Abbildung 2.5 sieht man, dass aus dem linken Text gelöschte Wörter rot markiert werden und im rechten Text hinzugefügte Wörter grün markiert werden.

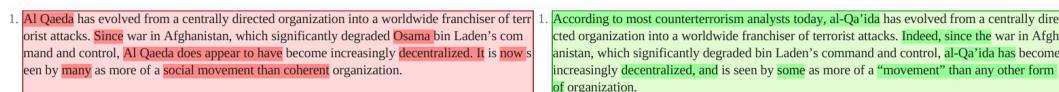


Abbildung 2.5: Diffchecker

## 2.3 Visualisierungen für Textwiederverwendung

Ziel der Visualisierungen für Textwiederverwendungen ist nicht nur die Identifizierung der Gemeinsamkeiten, sondern auch eine übersichtliche Darstellung der Ergebnisse.

<sup>4</sup><http://people.f4.htw-berlin.de/~weberwu/simtexter/app.html>

<sup>5</sup><https://www.diffchecker.com/>

### 2.3.1 CitePlag

Ähnlich wie in der Webapplikation von Weber-Wulf visualisieren Gipp et al. [2014] den Vergleich von zwei Texten. Jede Gemeinsamkeit kann mit einer eindeutigen Farbe zugeordnet werden. Außerdem bietet diese Visualisierung eine Übersicht über die gesamte Länge der verglichenen Dokumente. In der Mitte zwischen beiden Texten werden Zitate durch Punkte auf beiden Seiten dargestellt, die durch gekrümmte Linien verbunden sind. So lassen sich auch Gemeinsamkeiten zuordnen, die an unterschiedlichen Stellen in beiden Texten stehen.

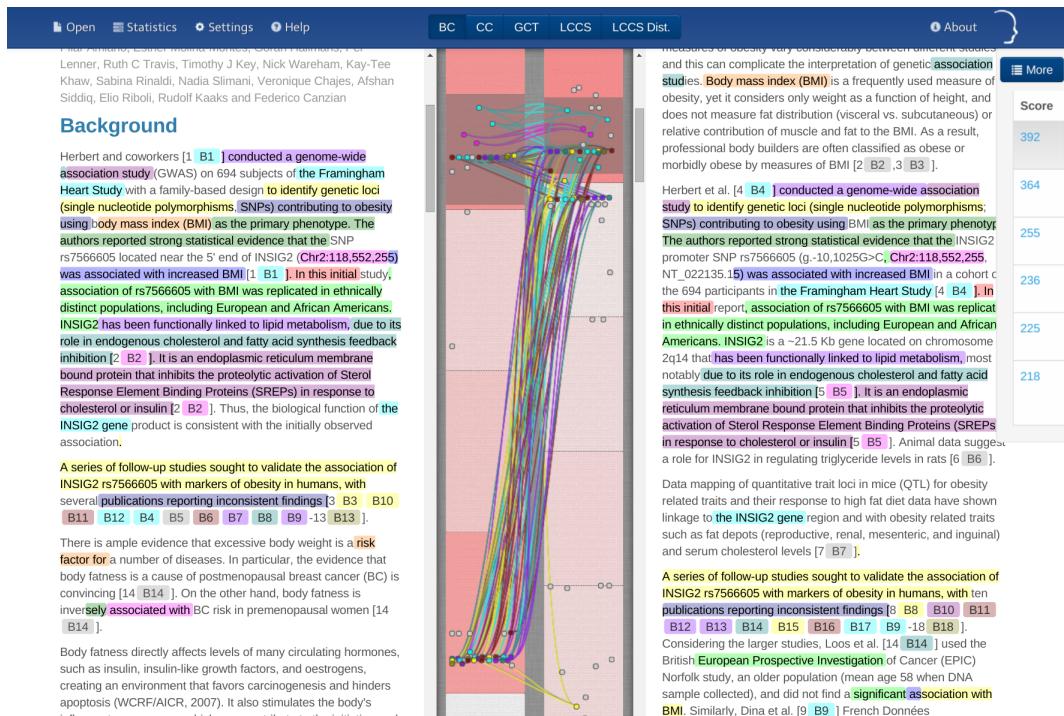


Abbildung 2.6: CitePlag von Gipp et al. [2014]

### 2.3.2 PlagVis

Riehmann et al. [2015] entwickelten mit PlagVis ein Tool, welches sowohl eine Übersicht aller Fundstellen von Plagiat in einer wissenschaftlichen Arbeit, als auch eine Visualisierung dieser Stellen im Detail liefert. Dazu wurden die Daten von VroniPlag verwendet, welche auch unter anderen in dieser Arbeit visualisiert werden. In Abbildung 2.7 sieht man das dazu entwickelte visuelle Toll. Eine Spalte am linken Rand zeigt alle Quellen, aus denen plagierte

wurde (a). In der Spalte rechts daneben sind alle Seiten der Arbeit eingezeichnet (g). Seiten, bei denen Plagiat gefunden wurde, werden durch Linien mit den entsprechenden Quellen verbunden. Bei Auswahl einer Quelle werden alle ausgehenden Kanten hervorgehoben. Rechts neben dieser Übersicht ist eine Listenansicht aller Fundstellen (b), die ebenfalls mit den korrespondierenden Seiten verbunden sind (h).



Abbildung 2.7: PlagVis von Riehmann et al. [2015]

Jede Fundstelle wird außerdem durch ein Glyph repräsentiert (d), welches eine Übersicht über die Bearbeitung von Originaltext zu Plagiat gibt. Diese „DiffLine“ repräsentieren beide Textabschnitte als längliches Rechteck und visualisiert sowohl Abschnitte, die vom Originaltext übernommen oder gelöscht wurden, als auch Abschnitte, die im plagiarierten Text hinzugefügt wurden. Diese Visualisierung basiert auf dem „Wordgraph“ von Riehmann et al. [2012], bei der Sätze auf Wortbasis verglichen werden. Die Sätze werden übereinander dargestellt, wobei Wörter als Knoten in einem Graph betrachtet werden können. Zwischen aufeinander folgenden Wörtern wird eine Kante gezeichnet. Werden beim Vergleich von übereinander stehenden Wörtern Übereinstimmungen gefunden, können diese Knoten zu einem zusammengefasst werden, von dem mehrere Kanten aus und ein gehen. In Abbildung 2.8 sieht man dazu ein Beispiel. Eine Version dieser Visualisierung wurde auch in dieser Arbeit erstellt und wird in Kapitel 4.4 beschrieben.

## KAPITEL 2. VERWANDTE ARBEITEN

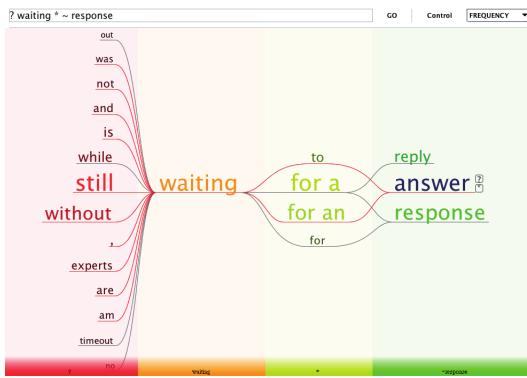


Abbildung 2.8: „Wordgraph“ von Riehmann et al. [2012]

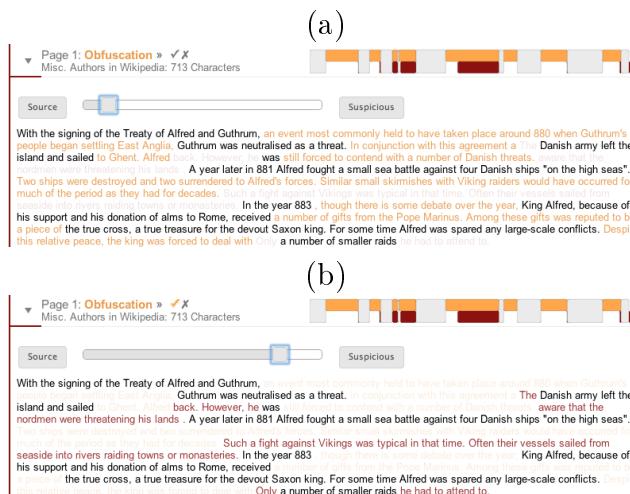
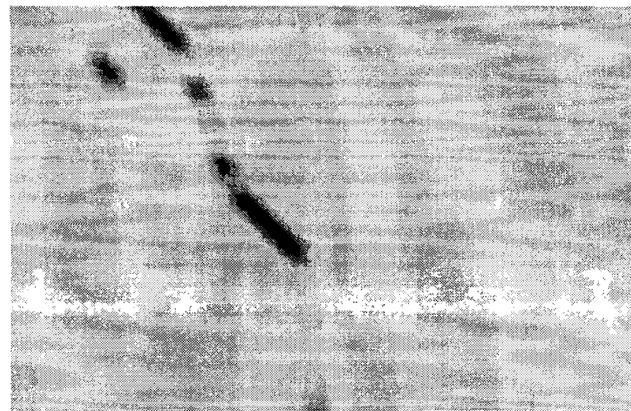


Abbildung 2.9: „Diff Blending“ von Riehmann et al. [2015]

Wählt man eine Fundstelle aus, können Quelle und Plagiat visuell verglichen werden. Dazu können hier drei unterschiedlichen Techniken ausgewählt werden. Die Visualisierungen „Vertical Aligned“ und „Classic Diff“ finden auch in dieser Arbeit Anwendung und werden in Kapitel 4 beschrieben. Beim „Diff Blending“ kann der originale Text, mithilfe eines Schiebereglers, zum plagierten Text umgewandelt werden. Dafür werden die Texte übereinander gelegt und die Transparenz der Wörter, die sich unterscheiden, mit dem Schieberegler gesteuert. In Abbildung 2.9 (a) sieht man den Originaltext, da der Schieberegler sich auf der linken Seite befindet. Schiebt man ihn nach rechts, blenden die gelben Wörter aus und die roten Wörter ein (b).

### 2.3.3 Matrixvisualisierungen

Bei einem Vergleich von zwei Texten wird oft eine Matrixdarstellung gewählt, bei der ein Text auf der x-Achse und der andere Text auf der y-Achse dargestellt wird. Culwin and Lancaster [2001] untersuchten studentische Arbeiten auf Plagiate. Um den Vergleich von zwei Texten zu visualisieren, wird ein Ähnlichkeitswert auf Satzbasis berechnet und in der Matrix dargestellt. Jeder Satz des ersten Textes wird mit jedem Satz des zweiten Textes verglichen. Die Ähnlichkeit wird per Grauwert an der Stelle des Rasters angegeben. Schwarz entspricht zwei gleichen Sätzen, weiß zwei völlig unterschiedlichen. Dadurch ergeben sich Muster, die über die strukturelle Ähnlichkeit der Texte Aufschluss geben.



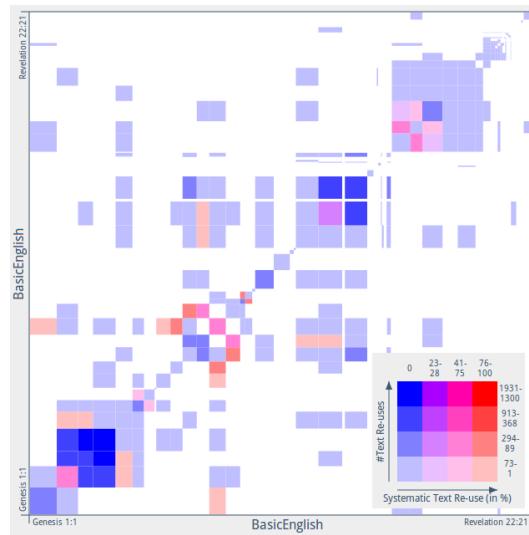
**Abbildung 2.10:** Matrixvisualisierung von Culwin and Lancaster [2001]

In Abbildung 2.10 sieht man beispielsweise mehrere schwarze Stellen auf einem sonst hellgrauem Hintergrund. Bei den Textstellen, die an diesen Stellen miteinander verglichen wurden, scheint eine höhere Ähnlichkeit im Vergleich zum Rest der beiden Texte zu existieren.

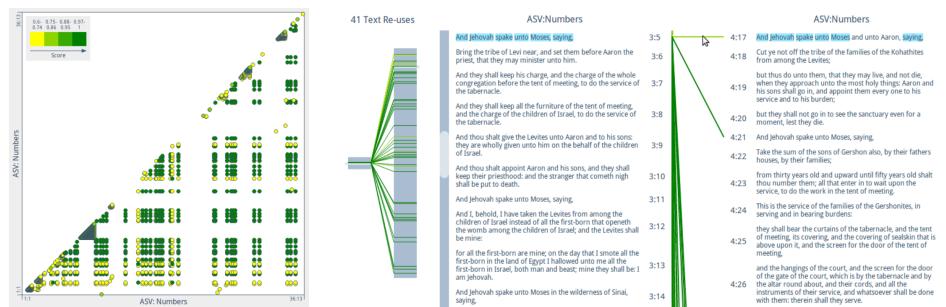
Jänicke et al. [2014] entwickelten zwei Matrixvisualisierungen, die jeweils einen Vergleich in unterschiedlichem Detailgrad darstellen. Hierzu vergleichen sie verschiedene Versionen der Bibel miteinander. Für die Übersicht des Vergleiches zweier Bücher wurde das „Text-Reuse-Grid“ erstellt. Die Bücher werden kapitelweise auf den Achsen aufgetragen. Die Anzahl der Textwiederverwendungen und der Prozentsatz der systematischen Textwiederverwendungen werden berechnet und an der entsprechenden Stelle des Rasters mit Farbe und Sättigung abgebildet. In Abbildung 2.11 sieht man, dass Rot einer höheren und Blau einer niedrigeren systematischen Textwiederverwendung entspricht, während die Anzahl der Wiederverwendungen durch die Sättigung angegeben wird.

## KAPITEL 2. VERWANDTE ARBEITEN

---



**Abbildung 2.11:** „Text-Reuse-Grid“ von Jänicke et al. [2014]

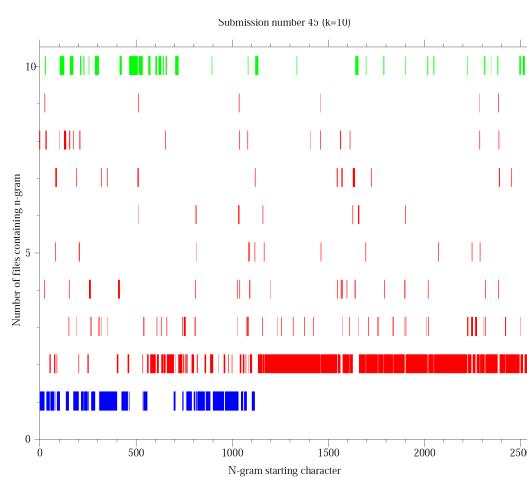


**Abbildung 2.12:** „Dot-Plot-View“ von Jänicke et al. [2014]

Verglichen wurde hier die „Basic English“ Version der Bibel mit sich selbst, wodurch eine symmetrische Matrix entsteht. Um zwei Kapitel im Detail zu vergleichen, wird in dem „Dot-Plot-View“ ebenfalls eine Matrix erstellt. Für jede Textwiederverwendung zwischen zwei Sätzen wird ein Punkt gezeichnet. Die Farbe gibt den Ähnlichkeitswert an. An die „Dot-Plot-View“ ist ein „Text-Reuse-Reader“ gekoppelt, der beide Texte nebeneinander anzeigt. Gibt es Textwiederverwendung zwischen zwei Sätzen, wird dies durch eine horizontale Linie zwischen den beiden Sätzen angezeigt.

### 2.3.4 Koordinatensystemvisualisierungen

Mit „Categorical Patterngrams“ verglichen Ribler und Abrams [2000] einen Basistext zu unterschiedlichen Texten. Ziel der Visualisierung ist es, darzustellen in wie vielen unterschiedlichen Texten die N-Gramme des Basistextes vorkommen. Dazu wird ein Graph erstellt mit den N-Grammen auf der x-Achse und die Anzahl des Auftretens in allen Texten auf der y-Achse. N-Gramme, die einzigartig für den Basistext sind, befinden sich auf der y-Koordinate 1. N-Gramme, die in der gesamten Sammlung an Texten häufig vorkommen, befinden sich auf sehr hohen y-Koordinaten.



**Figure 8. Patterngram of Submission 45**

**Abbildung 2.13:** „Categorical Patterngram“ von Ribler and Abrams [2000]

Durch die Clusterbildungen gibt die Visualisierung eine gute Übersicht darüber, ob Textabschnitte von der Wortwahl her besonders einzigartig oder allgemein sind. Für Plagiatserkennung sind Abschnitte interessant, in denen N-Gramme mehr als einmal, aber nicht zu häufig in anderen Texten vorkommen. In dem Beispiel in Abbildung 2.13 wurde nach 10-Grammen gesucht. 10-Gramme, die 10 mal oder häufiger in anderen Texten vorkommen, sind für diese Betrachtung eher unwichtig und werden, grün markiert, auf der zehnten Zeile dargestellt. 10-Gramme, die nur im Basistext vorkommen, sind ebenfalls uninteressant und wurden blau markiert. Interessant ist die zweite Zeile, in der ab ungefähr  $x=1500$  viele 10-Gramme hintereinander in einem anderen Text vorkommen. Dieser Abschnitt könnte aus dem Basistext kopiert worden sein.

Einen ähnlichen Ansatz wählten Monroy et al. [2002], die verschiedene Ver-

sionen von Don Quixote zu einem Basistext verglichen. Bei ihrer Timeline-Visualisierung, repräsentiert die x-Achse die Seiten der Bücher. Auf der y-Achse werden ebenfalls die verschiedenen Versionen eingetragen, sodass eine Reihe jeweils ein Buch repräsentiert. Für jede Seite, auf jeder Reihe, wird ein Balken gezeichnet, dessen Höhe die Unterschiede dieser Version zum Basistext auf dieser Seite zeigt. Die Balken können ausgewählt werden, um Informationen zu dieser Version abzurufen und die dargestellte Textstelle nachzulesen.

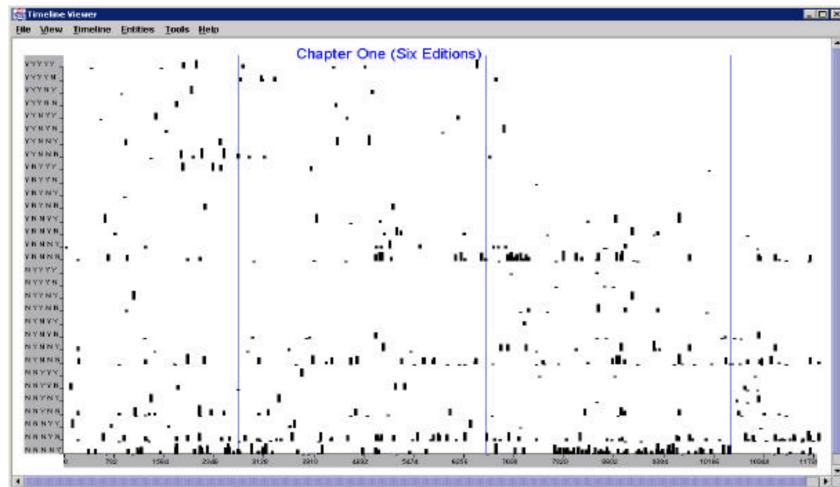


Abbildung 2.14: Timeline-Visualisierung von Monroy et al. [2002]

### 2.3.5 Barcodelvisualisierung

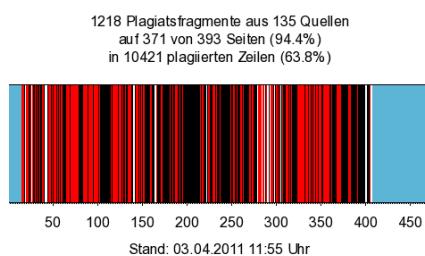


Abbildung 2.15: Barcodelvisualisierung von GutenPlag

Auf den Websites<sup>6</sup> der Plagiatsjäger von VroniPlag und GutenPlag wird eine Barcodelvisualisierung verwendet, die eine Übersicht über Plagiatsfragmente in

---

<sup>6</sup><http://de.vroniplag.wikia.com/> und <http://de.guttenplag.wikia.com/>

der gesamten Arbeit gibt. Jede Seite wird als vertikale Linie dargestellt, deren Farbe angibt ob kein Plagiat gefunden wurde (weiß), Plagiat gefunden wurde (schwarz) oder Plagiat aus mehreren Quellen gefunden wurde (rot).

## 2.4 Visualisierungen für Änderungsverfolgungen

Bei der Änderungsverfolgung werden Unterschiede oder Gemeinsamkeiten von mehreren Versionen des gleichen Textes visualisiert.

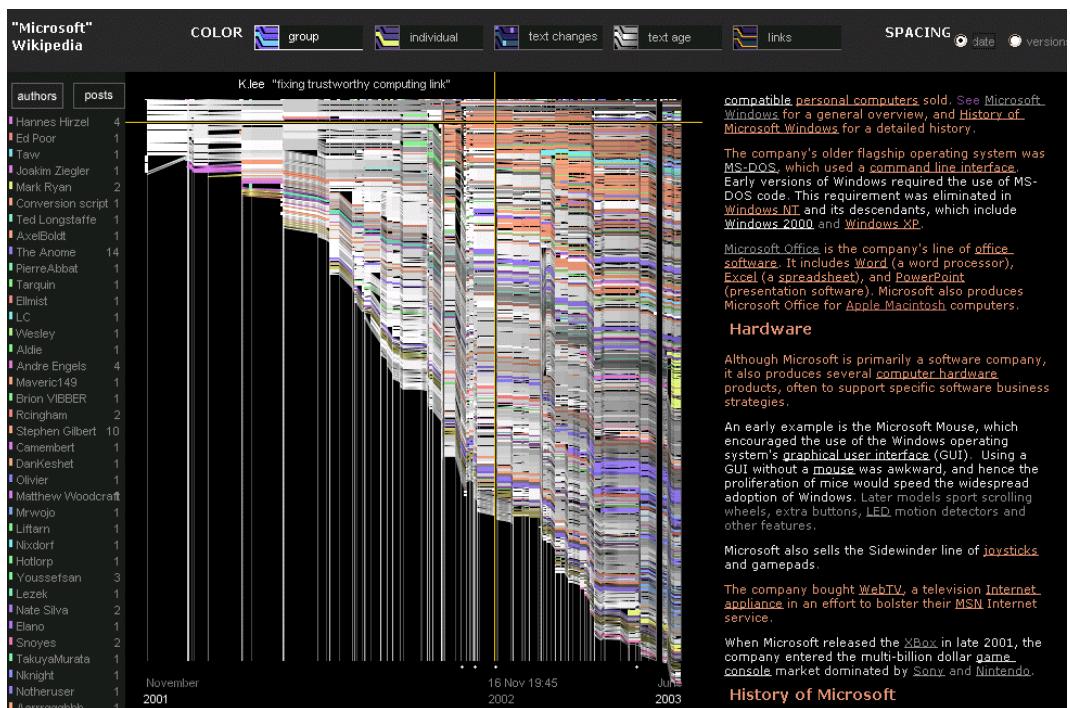


Abbildung 2.16: „History Flow“ von Viégas et al. [2004]

Eine Übersicht über die Bearbeitungshistorie von Wikipedia Artikeln entwickelten Viégas et al. [2004] mit „History Flow“. Dort werden alle Versionen eines Artikels als vertikale Linien nebeneinander angeordnet. Wikipedia Artikel werden von mehreren Autoren bearbeitet. Der prozentuale Anteil des Textes wird farblich dem Autor zugeordnet. Zwischen korrespondierenden Segmenten werden Schattierungen gezeichnet, sodass gleiche Abschnitte zwischen den Versionen zugeordnet werden können. Außerdem wird der Abstand dieser Linien an den realen zeitlichen Abstand angepasst. In Abbildung 2.16 sieht man die Bearbeitungen der Wikipedia Seite von Microsoft.

Shannon et al. [2010] entwickelten mit „Deep Diff“ eine Visualisierung um mehrere Revisionen eines Textes zu vergleichen. Dazu werden zuerst die ersten beiden Revisionen miteinander verglichen. Bei den Vergleichen von Revision 2 mit 3, 3 mit 4, und so fortlaufend, werden die Ergebnisse der vorherigen Vergleiche berücksichtigt. Die Visualisierung geht dann von der letzten Revision aus und markiert alle Textabschnitte, die in vorherigen Revisionen hinzugefügt wurden. Die Farbe gibt an, wie aktuell diese Bearbeitung ist.

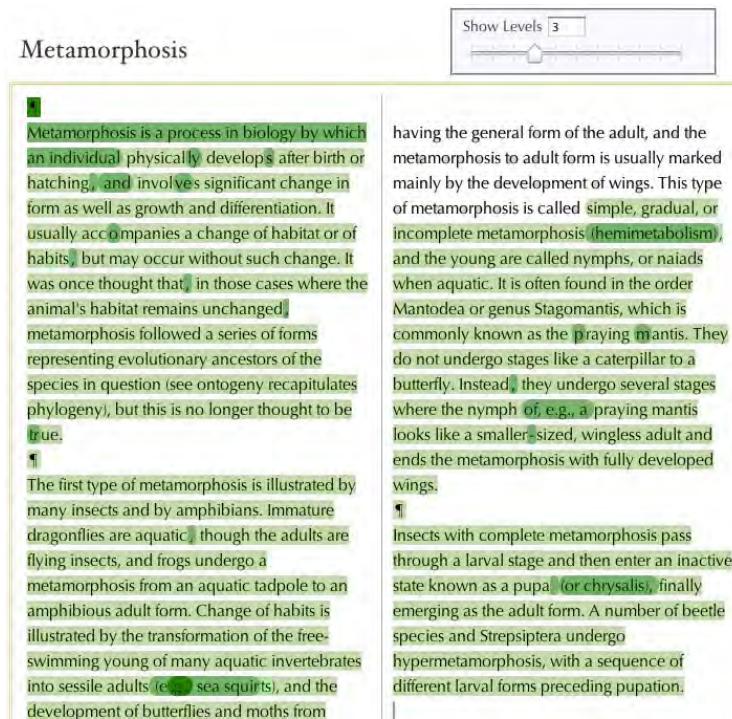


Abbildung 2.17: „Deep Diff“ von Shannon et al. [2010]

## 2.5 Abstrakte Visualisierungen - Graphvisualisierungen

Es gibt Visualisierungen, deren Ziel es ist, Ähnlichkeitswerte für Texte eines ausgewählten Korpus zu berechnen, um die Ergebnisse und daraus folgenden Beziehungen übersichtlich darzustellen. Von Freire [2008] wurde beispielsweise ein graphbasierter Ansatz gewählt, um Programmcode von Studenten miteinander zu vergleichen. Dabei werden die Programme als Knoten dargestellt. Die Kanten zwischen den Knoten geben über ihre Farbe und Dicke den entspre-

chenden Ähnlichkeitswert an.

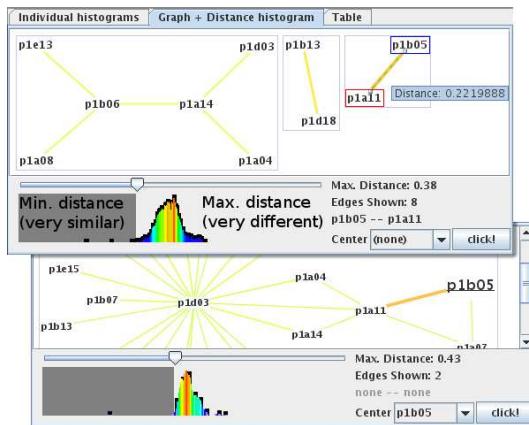
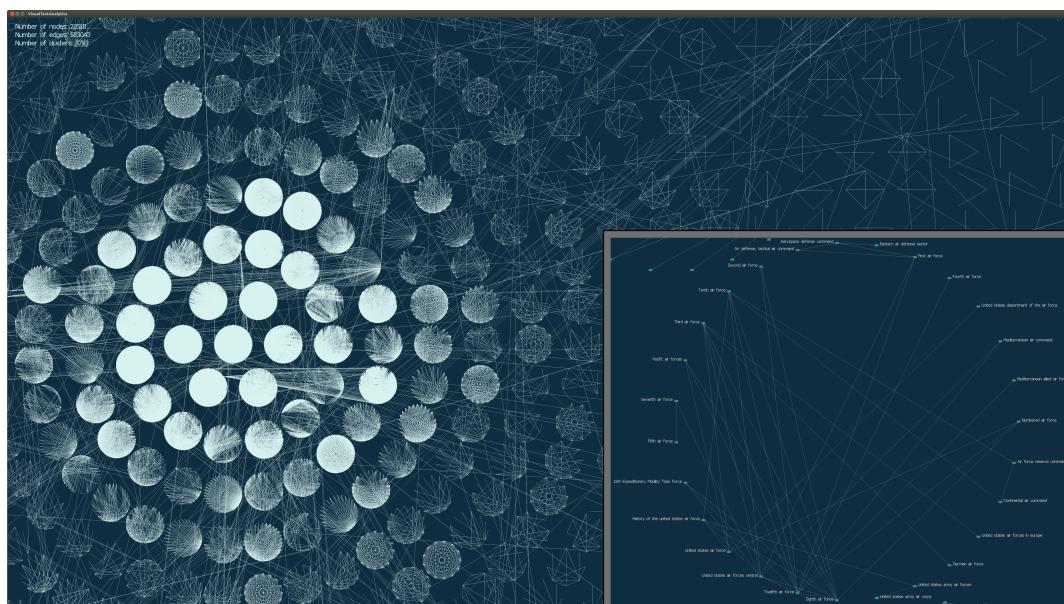


Abbildung 2.18: Graphvisualisierungen von Freire [2008]

In dem Semesterprojekt „Visual Text Analysis“ des Wintersemesters 2014/15 verglichen wir alle Wikipedia Artikel miteinander und erstellten mit den Ergebnissen eine ähnliche Visualisierung, wobei ein Knoten einen Artikel repräsentiert. Die Ähnlichkeit zwischen zwei Artikeln lässt sich an der Transparenz der Kante ablesen. Zusätzlich werden Kanten nur gezeichnet, wenn der Ähnlichkeitswert über einem, vom Nutzer gewählten, Schwellwert liegt. So ergeben sich Cluster, die durch verschiedene Layoutalgorithmen übersichtlich angeordnet werden können. In Abbildung 2.19 sieht man radial angeordnete Cluster, sortiert nach der Anzahl der Knoten. Das Cluster mit den meisten Knoten befindet sich in der Mitte. Wählt man ein Cluster aus, kann es im Detail betrachtet werden (siehe unten rechts). Außerdem stellt die Visualisierung Funktionalitäten zur Verfügung, mit denen Cluster verschoben, markiert und vereint werden können.

## KAPITEL 2. VERWANDTE ARBEITEN

---



**Abbildung 2.19:** Visualisierung aus dem Projekt „Visual Text Analysis“

# Kapitel 3

## Nutzerstudie

In einer Nutzerstudie sollten 34 Schüler (9. bis 11. Klasse) eigene Visualisierungen zum Vergleich von drei Revisionen eines Nachrichtenartikels entwickeln. Als Probanden wurden Schüler gewählt, weil bei Erwachsenen eher die Möglichkeit der Voreingenommenheit durch bereits bekannte Textvergleichsvisualisierungen besteht.

### 3.1 Aufbau

Die Studie war in zwei Aufgaben geteilt. In der ersten, der Analyse, sollten die Schüler die Unterschiede und Gemeinsamkeiten der Texte erkennen und markieren. Die zweite Aufgabe war es, eine Übersicht zu zeichnen, um einer imaginären dritten Person, die keine Zeit hat alles zu lesen, zu zeigen, wie sich der Text in den drei Revisionen verändert hat. Speziell sollten Änderungen und Verschiebungen im Text oder gleich gebliebene Abschnitte dargestellt werden.

Beide Aufgaben dauerten jeweils ungefähr 15 Minuten, dabei ging es weniger um eine vollständige Übersicht aller Fundstellen, als darum das erdachte Konzept zur Visualisierung deutlich zu machen. Die Schüler durften sich in Gruppen von zwei bis zu drei zusammenfinden. Jede Gruppe bekam die drei ausgedruckten Revisionen und Material zum Bearbeiten, darunter leere Blätter, Scheren und Kleber. Weitere Materialien wurden unterschiedlich auf die Gruppen verteilt, um Diversität zu sichern. Dadurch entstehen drei Obergruppen. Die Erste arbeitete mit Textmarkern in vier Farben, die Zweite bekam Buntstifte und die Dritte lediglich Bleistifte.

## 3.2 Ergebnisse

Die erstellten Übersichten waren sehr unterschiedlich und zeigten keine Anzeichen für ein gemeinsames visuelles Konzept. Um sie kategorisieren zu können, wurde zwischen Übersichten unterschieden, die als Einzel- oder Referenzdokument dargestellt werden, die alle Dokumente gleichberechtigt darstellen, die eine abstrakte Visualisierung sind und Übersichten, bei denen eine Fragmentierung in Textabschnitte und anschließende Darstellung in einer einzelnen Ansicht vorgenommen wurde.

In Abbildung 3.1 sieht man eine Visualisierung, die Revision 1 als Referenzdokument verwendet. Alle Paragraphen der jeweiligen Revision werden als horizontale Balken untereinander dargestellt. Die Paragraphen der ersten Revision sind grün markiert. So kann in der zweiten Revision dargestellt werden, welche Paragraphen übernommen wurden, indem sie ebenfalls grün gezeichnet werden. Sind Paragraphen nur teilweise gleich, kann das mit teilweise grün markierten Balken dargestellt werden. Das Prinzip wird für die zweite Revision fortgesetzt, indem in dieser Revision hinzugefügte Paragraphen rot markiert werden. Außerdem werden durch horizontale Linien zwischen den Revisionen Textverschiebungen angezeigt.

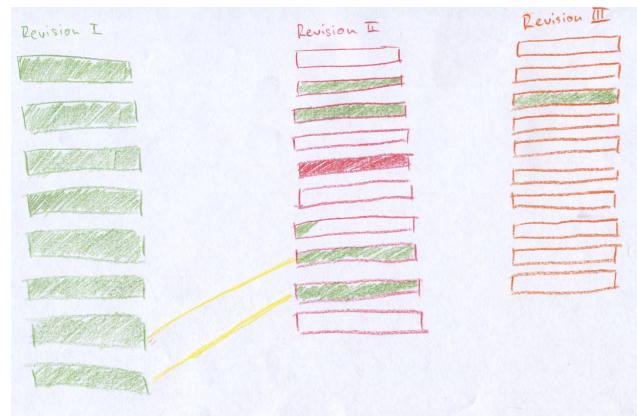


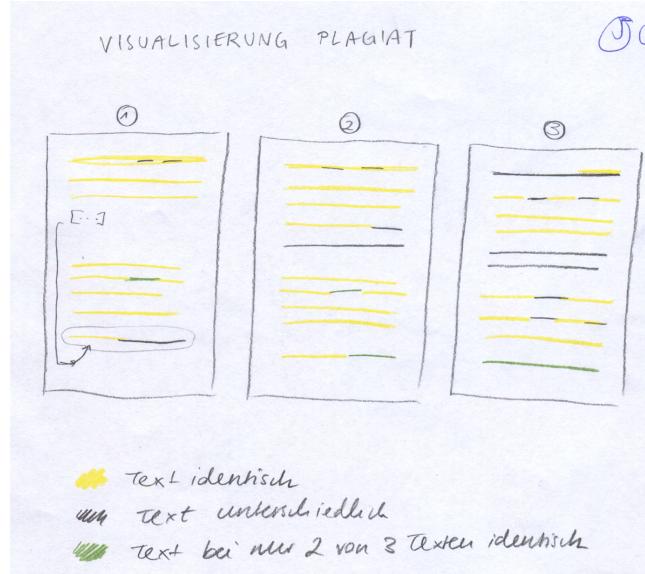
Abbildung 3.1: Erstes Beispiel für eine Visualisierung mit Referenzdokument

Auch die Visualisierung in Abbildung 3.2 nimmt Revision 1 als Referenzdokument und zeigt eine Übersicht darüber, welche Textabschnitte in den anderen Revisionen übernommen wurden. Dafür werden farbige, horizontale Linien eingezeichnet, die grob Textzeilen repräsentieren. Gelb steht für identischen Text, schwarz für unterschiedlichen und grün für Text, der in zwei von drei

## KAPITEL 3. NUTZERSTUDIE

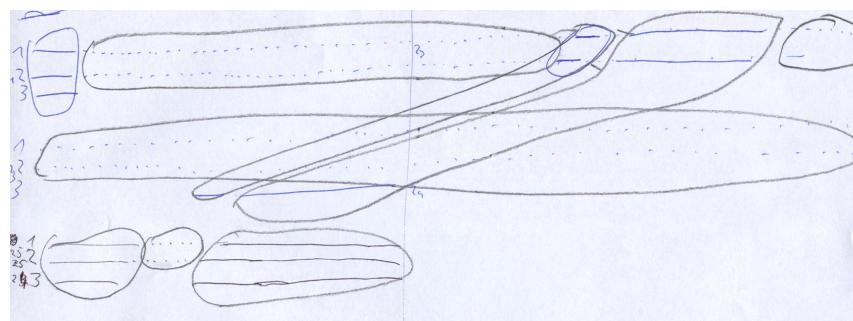
---

Revisionen identisch ist. Außerdem werden Textverschiebungen durch Pfeile angezeigt.



**Abbildung 3.2:** Zweites Beispiel für eine Visualisierung mit Referenzdokument

Abbildung 3.3 zeigt eine Visualisierung, die alle Revisionen gleichberechtigt und zeilenbasiert darstellt. Dabei werden alle Revisionen als horizontale Linien dargestellt und untereinander platziert. Die Art der Linie gibt an, ob und welche Revisionen in diesem Abschnitt der Zeile übereinstimmen. Die drei durchgezogenen Linien zu Beginn zeigen, dass alle Revisionen gleich beginnen. Danach stimmen Revision 1 und Revision 2 überein, während dieser Abschnitt in Revision 3 fehlt.

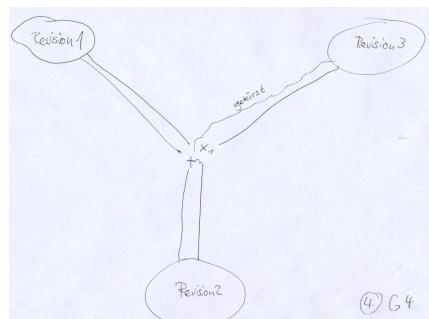


**Abbildung 3.3:** Beispiel für eine gleichberechtigte Darstellung aller Revisionen

Die gepunkteten Linien existieren nur in den ersten beiden Revisionen,

die durchgezogenen in allen. Außerdem sind Textverschiebungen durch ovale Umrandungen eingezeichnet. Diese Darstellung ähnelt der Diffline, welche in Kapitel 2.3.2 vorgestellt wurde, aber auch in leicht abgewandelter Version in dieser Arbeit erstellt wurde (siehe Kapitel 4.4).

In Abbildung 3.4 sieht man eine abstrakte Visualisierung, bei der die Revisionen radial angeordnet sind. Alle Paragraphen der Revisionen wurden mit  $X_x$  benannt und werden in der Mitte angeordnet. Von den Revisionen zeigen Linien zu den Paragraphen, deren Form Aufschluss darüber gibt, ob der Paragraph gleich ist (gerade Linie) oder gekürzt wurde (gewellte Linie). Weitere Linienformen könnten verschiedene Bearbeitungsarten darstellen. Eine graphenbasierte Visualisierung von Textvergleichen, bei denen Texte als Knoten repräsentiert werden, ist nicht unüblich. Beispiele dazu wurden in Kapitel 2.5 vorgestellt.



**Abbildung 3.4:** Eine abstrakte Graphenvisualisierung

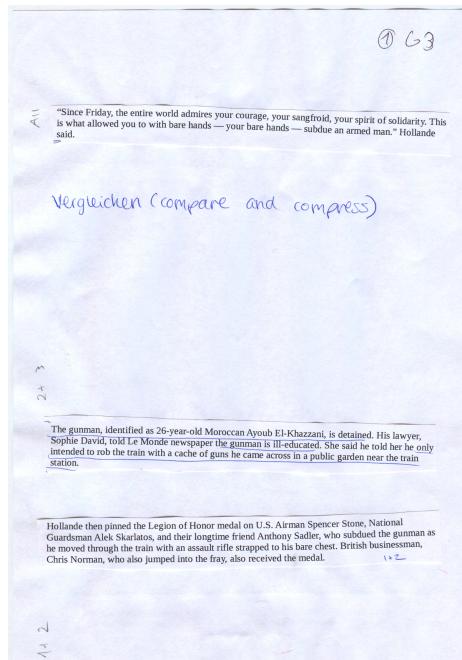


**Abbildung 3.5:** Eine abstrakte Koordinatensystemvisualisierung

## KAPITEL 3. NUTZERSTUDIE

---

Eine weitere abstrakte Visualisierung zeigt Abbildung 3.5, bei der alle Paragraphen aller Revisionen auf der x-Achse eines Graphen aufgetragen wurden. Die Anzahl der Unterschiede der Paragraphen sind auf der y-Achse angegeben. Die Revisionen können durch eine eindeutige Farbe zugeordnet werden. So hat Revision 3 im ersten Paragraph 10 Unterschiede zu den anderen Revisionen. Außerdem war eine Aufteilung in inhaltliche und sprachliche Unterschiede geplant, die durch Glyphen (Dreieck und Kreis) angegeben werden sollte. Ähnliche, mit Koordinatensystemen umgesetzte Visualisierungen, wurden in Kapitel 2.3.4 vorgestellt.



**Abbildung 3.6:** Beispiel für eine Fragmentierung in Textabschnitte und Darstellung in einer neuen Ansicht

Abbildung 3.6 zeigt eine mit Schere und Kleber erstellte Visualisierung, bei der Paragraphen ausgeschnitten und auf einem neuen Blatt Papier, geordnet danach, in welchen Revisionen sie vorkommen, aufgeklebt wurden. Oben sollten alle Paragraphen stehen, die in allen Revisionen vorkommen. Darunter Paragraphen, die nur in Revision 2 und Revision 3 und darunter Paragraphen, die nur in Revision 1 und Revision 2 vorkommen.

Die Ergebnisse zeigen, dass es keinen eindeutigen und intuitiv gewählten Weg gibt, Textvergleiche zu visualisieren. Es ist eine klare Tendenz zum Abbilden von Texttypen durch Farben vorhanden. Wenn den Gruppen Farben

### *KAPITEL 3. NUTZERSTUDIE*

---

zur Verfügung standen, wurden diese auch benutzt. Sogar halfen sich einige Gruppen, die eigentlich nur Bleistifte bekamen, mit selbst mitgebrachten Kugelschreibern aus, um mit zwei verschiedenen Farben zu arbeiten. Einige Visualisierungen zeigten Ähnlichkeit mit bereits bekannten Konzepten.

# Kapitel 4

## Textvergleichsvisualisierungen

Im folgenden Kapitel werden die Konzepte der entwickelten Textvergleichsvisualisierungen erklärt. Visualisiert werden die Ergebnisse tokenbasierter Vergleiche. Die Texte werden anhand von Leer- und Satzzeichen zu Listen von Zeichenfolgen (engl. Tokens) zerlegt. Durch die Differenzalgorithmen ist gegeben, dass jedes Token entweder in beiden Texten existiert (**equal**), nur im ersten Text vorhanden ist (**deletion**) oder nur im zweiten Text vorkommt (**insertion**). Bei einem Vergleich von zwei Texten wird im Folgenden immer von Quelle und Paraphrase, oder originalem Text und paraphrasiertem Text, gesprochen. Selbst wenn es sich bei dem Vergleich nicht um Texte handelt, die eine Quelle-Plagiat Beziehung haben, werden der Einfachheit halber ein originaler und ein paraphrasiertes Text definiert. Bei revisionierten Texten ist in der Regel die frühere der beiden Revisionen der Originaltext.

Teil des Konzepts dieser Visualisierungen ist, dass sie im Detail noch anpassbar sind. Es gibt also einige veränderbare Parameter, mit denen die Visualisierungen an die jeweiligen Texte angepasst werden können, umso die Grenzen und Möglichkeiten zu erproben. Genauer wird dies in Kapitel 6 erklärt.

### 4.1 Side By Side

Die wohl einfachste Visualisierungstechnik ist, beide Texte nebeneinander zu platzieren. Die Quelle steht dabei links und die Paraphrase rechts. Diese Darstellung scheint erst einmal trivial, ist allerdings die Ausgangslage für komplexe Visualisierungen. Im Prinzip bildet sie die Realität ab, in der man sich zwei Texte ausdrückt und nebeneinander legt. Das Erkennen von Gemeinsamkeiten und Unterschieden zeigt sich hier als schwierig, da beide Texte nicht ausgerichtet oder markiert sind. Deswegen wurde die „Vertical Aligned“ Visualisierung erstellt.

## KAPITEL 4. TEXTVERGLEICHSVISUALISIERUNGEN

---

Denn es gab militärische Auseinandersetzungen ohne Kriegszustand, wie z.B. im Mandschureikonflikt zwischen China und Japan 1931 und 1937, als trotz ausgedehnter Militäroperationen beide Regierungen darauf beharrten, daß kein Kriegszustand bestünde, da sie ihre diplomatischen Beziehungen weiterhin aufrecht erhielten.[FN 134] [FN 134] Langer, S. 63,

Das entmutigendste Beispiel dieser Art war das Verhalten von China und Japan 1931 und 1937, als diese in ausgedehnte Militäroperationen gegeneinander verwickelt waren mit schweren Verlusten und Zerstörung von Eigentum.[FN 16] Beide Regierungen beharrten jedoch darauf, daß zwischen ihren Staaten kein Kriegszustand bestünde, was durch ihre andauernden diplomatischen Beziehungen bewiesen wäre. [FN 16] Zur Entwicklung des Mandschureikonflikts s. die Darstellung bei Langer, R., Seizure of Territory (1947), 50-66, 123-131.

Abbildung 4.1: Side By Side Beispiel

## 4.2 Vertical Aligned

Vertical Aligned ist eine Visualisierung, die bereits in PlagVis Anwendung gefunden hat und in dem Paper von Riehmann et al. [2015] beschrieben wird. Die Texte werden auf ihre Gemeinsamkeiten geprüft und anhand dessen Zeilenumbrüche eingefügt, sodass gleiche Wortsequenzen nebeneinander stehen. Gleiche Wörter beider Texte werden rot markiert. So können Gemeinsamkeiten im Text schnell erkannt werden.

Denn es gab militärische  
Auseinandersetzungen ohne  
Kriegszustand , wie z . B . im  
Mandschureikonflikt zwischen China  
und Japan 1931 und 1937 , als  
trotz ausgedehnter  
Militäroperationen beide

Regierungen darauf beharrten , daß  
kein  
  
Kriegszustand bestünde , da sie  
ihre  
diplomatischen  
Beziehungen weiterhin aufrecht  
erhielten . [ FN 134 ] [ FN 134 ]  
Langer , S. 63 ,

Das entmutigendste Beispiel dieser  
Art war das Verhalten von China  
und Japan 1931 und 1937 , als  
diese in ausgedehnte

Militäroperationen gegeneinander  
verwickelt waren mit schweren  
Verlusten und Zerstörung von  
Eigentum . [ FN 16 ] Beide  
Regierungen beharrten jedoch darauf  
, daß zwischen ihren Staaten  
kein  
Kriegszustand bestünde , was durch  
ihre andauernden  
diplomatischen  
Beziehungen bewiesen wäre . [ FN  
16 ] Zur Entwicklung des  
Mandschureikonflikts s. die  
Darstellung bei Langer , R. ,  
Seizure of Territory ( 1947 ) , 50  
- 66 , 123 - 131 .

Abbildung 4.2: Vertical Aligned Beispiel

### 4.3 Classic Diff

Die Classic Diff Visualisierung findet bereits in verschiedenen visuellen Tools Anwendung. Sie funktioniert in der Art, dass beide verglichenen Texte zu einem zusammengefasst werden. Wörter, die aus dem originalen Text gelöscht wurden, werden rot markiert und durchgestrichen. Wörter, die im paraphrasierten Text hinzugefügt wurden, werden grün markiert und unterstrichen. Wörter, die in beiden Texten gleich sind, werden nicht markiert. Der Fokus liegt hier also auf den Veränderungen von Quelle zu Paraphrase.

In dieser Arbeit wurde eine Erweiterung für den Fall erstellt, wenn direkt hinter einem gelöschten Wort ein hinzugefügtes Wort steht. Daran erkennt man, dass das erste Wort durch das zweite ersetzt wurde. Nun bietet es sich an, die Ähnlichkeit dieser beiden Wörter zu berechnen. Liegt der berechnete Ähnlichkeitswert über einem Schwellwert, können die Wörter farblich markiert werden, um deutlich zu machen, dass hier eine Bearbeitung des Wortes vorliegt. In Abbildung 4.3 wurden beispielsweise „integrierter“ zu „integrierten“ und „innovativer“ zu „innovativen“ geändert. Die Farben für die Markierungen werden zufällig generiert. Eine Erweiterung dessen ist es, alle hinzugefügten Wörter mit allen gelöschten Wörtern zu vergleichen. So können Verschiebungen im Text erkannt und markiert werden.

Für Da weniger Länder die Versorgung übernehmen, erfordert eine „vernünftige Strategie für Energiesicherheit“ seien ein integrierter integrierten und innovativer innovativen Ansatz – netwennig. Ein Schlüsselfaktor für das künftige Wachstum ist Indiens Fähigkeit, den explodierenden Energiebedarf zu decken. Indien leidet an ernsten Engpässen und ist starkübermäßig von Erdöl und Kohle abhängig. [...] Indien hat reiche Öl- und Gasvorkommen, derendoch bei Erschließung der jedoch Förderung, die unter der Federführung des öffentlichen Sektors erfolgt, fehlt es an Unternehmensgeist.}

Abbildung 4.3: Classic Diff Beispiel

### 4.4 Two Lines Diff

Bei der Two Lines Diff Visualisierung werden ebenfalls beide Texte zu einem zusammengefasst. Am besten lässt sich die Visualisierung beschreiben, wenn man beide Texte als jeweils eine ununterbrochene Zeile betrachtet, die übereinander stehen. Der originale Text steht oben und der paraphrasierte Text darunter. Da es unpraktisch ist, einen Text zu lesen, der nur als eine ununterbrochene Zeile dargestellt wird, werden automatische Zeilenumbrüche eingefügt. Richtet man den Text anhand der Gemeinsamkeiten aus, müssen vier unterschiedliche Fälle berücksichtigt werden.

Für die französische Währung war das Jahr 1851 wesentlich <sup>b</sup>bedeutungsvoll gewesen , weil in diesem Jahr die am Edelmetallmarkt bestehende Relation das dem französischen <sup>(a)</sup>  
 Doppelwährungssystem <sup>zugrunde liegenden</sup> Gold / Silber - Wertverhältnis <sup>von 1 zu 15</sup> unterschritt . <sup>(c)</sup>

Abbildung 4.4: Two Lines Diff Beispiel

- (a) **equal:** Da sowohl in der oberen Zeile als auch in der unteren Zeile das Gleiche stehen würde, können die beiden Zeilen zu einer zusammengefasst werden. Diese Zeile wird im folgenden als *shared-line* bezeichnet, da sie sozusagen von beiden Texten geteilt wird.
- (b) **deletion + insertion:** In diesem Fall stehen zwei Zeilen übereinander, die obere Zeile repräsentiert den originalen Text, die untere den paraphrasierten. Das bedeutet der Text wurde in diesem Abschnitt von Original zu Paraphrase verändert. Diese doppelte Zeile wird im Folgenden als *double-line* bezeichnet.

Da die Texte an den Gemeinsamkeiten ausgerichtet sind, entstehen für *double-lines* zwangsläufig Lücken. Diese lassen sich in zwei Fälle kategorisieren.

- (c) **deletion:** Die untere Zeile bleibt leer, also wurden die Wörter der oberen Zeile aus dem originalen Text gelöscht.
- (d) **insertion:** Bleibt die obere Zeile leer, wurden die Wörter der unteren Zeile im paraphrasierten Text hinzugefügt.

So entsteht eine Visualisierung, die beide Texte zusammenfasst und sowohl Gemeinsamkeiten als auch Unterschiede darstellt. Das Layout kann im Detail angepasst werden. Bei den Zeilenumbrüchen kann entschieden werden, ob die Zeile nach ganzen Wörtern umgebrochen wird, oder zeichenbasiert. Durch diese Zeilenumbrüche entstehen globale Zeilen, die aus lokalen Zeilen (*shared-lines* und *double-lines*) bestehen.

Der Abstand zwischen den globalen Zeilen ist ebenfalls eines der veränderbaren Parameter. Bei Texten, die wenige Veränderungen aufweisen, kann es sinnvoll sein die globalen Zeilen näher zusammenzuschieben und so eine Übersicht über alle Unterschiede der Texte zu bekommen. Will man die Unterschiede und den Textfluss im Detail betrachten, empfiehlt sich eine Ansicht mit weiter auseinander geschobenen globalen Zeilen. Außerdem kann eingestellt werden, ob es zwischen den globalen Zeilen Trennlinien gibt und welcher Art diese sind (Typ, Farbe, Dicke). Weiterhin kann ein sogenanntes Zebra-Striping

eingestellt werden, bei dem die globalen Zeilen abwechselnd mit verschiedenen Farben hinterlegt werden. Die Schriftgröße der lokalen Zeilen ist ebenfalls einstellbar. Um dem Betrachter deutlich zu machen, dass die *shared-lines* in beiden Texten existieren, bietet es sich an, die Schriftgröße der *shared-lines* in etwa so groß zu machen wie beide Teile der *double-lines* zusammen.

Im Fall (**b**) **deletion + insertion** wurde noch eine Erweiterung entwickelt, mit der getestet werden kann, welcher Art die Veränderung ist. Dafür wird für alle Wortpaare von Quelle zu Paraphrase ein Ähnlichkeitswert berechnet. Liegt dieser über einem Schwellwert, kann man davon ausgehen, dass das Wort des Originaltextes nur abgeändert wurde. Diese Wörter können wieder durch automatisch generierte Farben markiert werden.

## 4.5 Animation Diff

Einen dynamischeren Ansatz bietet das Animation Diff. Die Visualisierung setzt sich ebenfalls aus generischen Komponenten zusammen. Dazu wurden Animatoren erstellt, die Animationen ausführen und so die HTML-Elemente der Seite transformieren. Ein Animator braucht bestimmte Voraussetzungen, um zu funktionieren. Sind diese erfüllt, können Animatoren problemlos ausgetauscht werden. So kann man beliebig Animatoren kombinieren, um die beste Visualisierung für den jeweiligen Text zu schaffen. Visualisierungen für Quelle-Plagiat Texte können selbstverständlich immer auch für zwei Revisionen ausgeführt werden. Allerdings wurde hier zusätzlich eine Visualisierung für revisionierte Texte erstellt, die in 4.5.2 vorgestellt wird.

### 4.5.1 Quelle-Plagiat Animationen

Die Idee bei Quelle-Plagiat Animationen ist es, die Arbeitsweise des Autors zu imitieren. So wird zuerst nur der Originaltext auf der linken Seite angezeigt. Die rechte Seite repräsentiert ein leeres Blatt. Nun werden Animationen gestartet, um die linke Seite zu kopieren und zu bearbeiten, um so den paraphasierten Text zu erstellen. Die initialen Animationen lassen sich in drei Arten einteilen. Hier sei noch einmal angemerkt, dass dieselben Animatoren in verschiedenen Visualisierungen verwendet werden können. Außerdem gibt es Animatoren, die nur eine Kombination von anderen Animatoren sind.

#### Alles kopieren

Hier wird der gesamte Originaltext markiert und eine Kopie dessen auf die rechte Seite transformiert. Anschließend wird er so bearbeitet, dass der paraphasierte Text entsteht. Für das Kopieren des Originaltextes gibt es verschie-

dene Animatoren. Der Einfachste ist *translate*, der eine Kopie des gesamten Textes, in einer variablen Zeit, auf die rechte Seite verschiebt. Diese Animation macht recht deutlich was passiert, spiegelt allerdings eher weniger das tatsächliche Arbeiten mit Textprogrammen wieder, und ist daher eher unverständlich.

Um eine intuitivere Kopieranimation zu erstellen, wurden Copy-Animatoren entwickelt. Diese stellen eine Animation dar, bei der der Text markiert und auf die rechte Seite kopiert wird. Sie setzen sich wie folgt zusammen.

- Der gewünschte Text wird markiert (Abbildung 4.5 (a))
- Die Markierung wird ausgeblendet
- Der Text erscheint auf der rechten Seite, gleichzeitig wird die Markierung wieder eingeblendet (Abbildung 4.5 (b))
- Die Markierung blendet endgültig aus

Während die letzten drei Punkte zusammen eine Art Aufleuchten-Animation ergeben und unverändert bleiben, gibt es für die initiale Markierung mehrere Möglichkeiten.

*Array-Highlight* markiert die gegebenen Wörter nacheinander in konstanter Zeit. Besonders über mehrere Zeilen wirkt diese Animation unnatürlich. Deswegen wurden Animatoren entwickelt, welcher das Markieren von Absätzen mit Maus oder Tastatur imitiert.

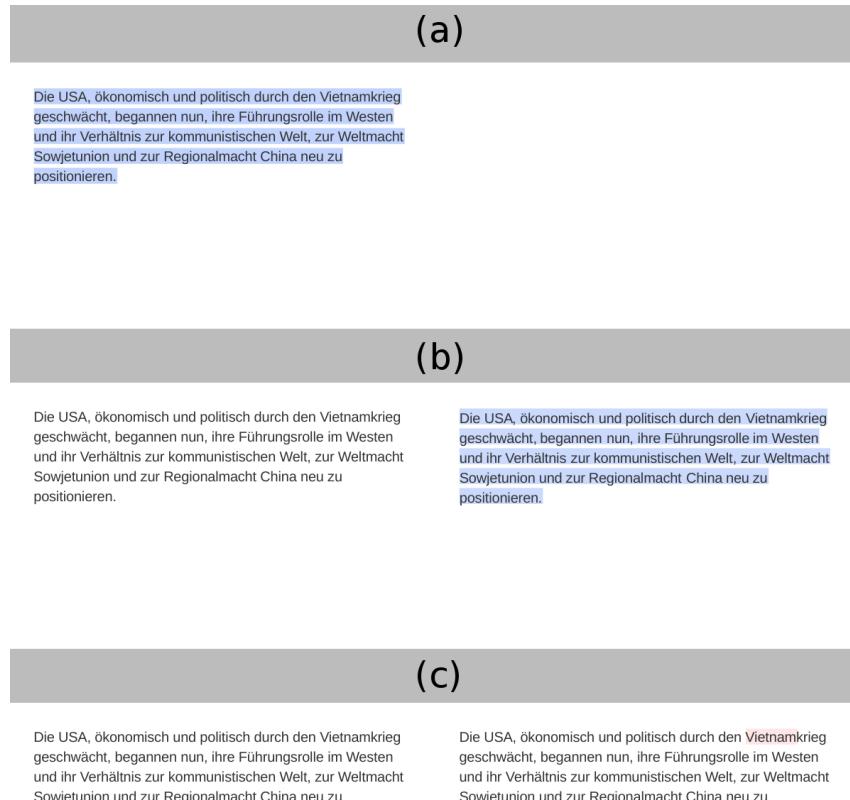
*Natural-Highlight* markiert den gesamten Text. Dabei startet die Markierung bei dem ersten Wort und setzt sich dann Diagonal nach unten rechts fort. Dies imitiert das Markieren eines Absatzes einer Website oder innerhalb eines Textbearbeitungsprogramms mit der Maus.

*Double-Click-Highlight* markiert zuerst ein Wort in der Mitte des Textes, anschließend den gesamten Text. Dies soll Doppelklick-Markieren von Absätzen in Textbearbeitungsprogrammen oder Websites nachbilden.

*Ctrl+A Highlight* blendet die Tastenkombination ein, mit der am PC der gesamte Text des Fensters markiert werden kann. Dieser Animator kann noch in der Art erweitert werden, dass die Tastenkombination für Kopieren (Ctrl+C) und die für Einfügen (Ctrl+V) eingeblendet wird, um so das Kopieren auf die rechte Seite zu visualisieren.

Die Art der Transformation, sowie die Dauer der Animationen, kann angepasst werden. Ist der Text auf der rechten Seite, beginnt die Bearbeitung (Abbildung 4.5 (c)). Der Text wird sequenziell vom ersten bis zum letzten Wort bearbeitet. Es kann ein Caret aktiviert werden, das in etwa dem eines

## KAPITEL 4. TEXTVERGLEICHSVISUALISIERUNGEN



**Abbildung 4.5:** Copy & Paste Animation

herkömmlichen Textbearbeitungsprogramms entspricht und angibt, an welcher Stelle sich die Animation befindet. Wie schon bei vorherigen Visualisierung muss zwischen drei Fällen unterschieden werden.

Muss das Wort **gelöscht** werden, wird es Zeichen für Zeichen rot markiert. Es kann ausgewählt werden, ob das Wort anschließend ausblendet und der folgende Text nachrückt, so als würde es tatsächlich gelöscht werden. Die andere Möglichkeit ist, das Wort rot-markiert stehen zu lassen und mit dem nächsten Wort fortzufahren.

Ist das Wort **gleich** muss keine Bearbeitung stattfinden. Das Wort kann, wenn gewünscht, markiert werden. Ist der Caret aktiv, wandert er sequenziell über alle Zeichen des Wortes.

Muss ein neues Wort **hinzugefügt** werden, gibt es zwei Möglichkeiten.

*Fade-In* verschiebt den folgenden Text, um eine Lücke zu schaffen. Anschließen wird das neue Wort dort eingeblendet. Alternativ kann *Typing* das Wort durch eine Tippanimation Zeichen für Zeichen hinzufügen. Um eine natürlichere Tippanimation zu schaffen, gibt es *Natural-Typing*. Hier variiert die Tippgeschwindigkeit für jedes Zeichen je nach Wortlänge, um das Schreiben eines Autors zu imitieren. Wurde das Wort hinzugefügt, gibt es, wie bei gelöschten Wörtern, die Möglichkeit eine Markierung beizubehalten, um auch nach Ende der Animation hinzugefügte Wörter von dem Rest unterscheiden zu können.

Da es öfter vorkommt, dass mehrere Wörter hintereinander der gleichen Art sind (**gelöscht**, **gleich**, **hinzugefügt**), kann ausgewählt werden, Animationen zusammenzufassen. Die Animationen werden dann für mehrere Wörter ausgeführt. Dadurch sind sie schneller und wirken flüssiger. Steht ein hinzugefügtes Wort direkt hinter einem gelöschten Wort, kann man davon ausgehen, dass das Erste durch das Zweite ersetzt wurde. Eine Erweiterung hierfür ist der *Editing Animator*, welcher den Caret Zeichen für Zeichen über das gelöschte Wort schiebt und bei Unterschieden zum hinzugefügten Wort, das Zeichen ändert. Diese Animation ist jedoch nur sinnvoll, wenn beide Wörter bis auf wenige Zeichen gleich sind.

### Gleiches kopieren

Bei dieser Visualisierung werden direkt zu Anfang alle **gleichen** Wörter im Originaltext markiert und auf die rechte Seite kopiert. Beim Markieren und Kopieren, beziehungsweise Translatieren können die gleichen Animatoren gewählt werden wie bei der **Alles Kopieren** Visualisierung. Allerdings würden hier die Animatoren *Double-Click-Highlight* und *Ctrl+A Highlight* eher missverständlich wirken, da nicht alles markiert wird, sondern nur einzelne Wörter des Textes. Nachdem die gleichen Wörter auf die rechte Seite kopiert wurden, beginnt die Bearbeitung. Diese läuft ähnlich ab wie bei der vorherigen Visualisierung, allerdings spielen gelöschte Wörter hier keine Rolle mehr. Es werden lediglich die Lücken mit hinzugefügten Wörtern gefüllt.

### Sequenzielles Kopieren

Hier wird ein anderes Konzept angewandt, bei dem die Wörter des paraphrasierten Textes sequenziell bearbeitet werden. Wörter die **gleich** sind werden von der linken Seite kopiert. Animatoren können hier *highlight+copy* oder *translate* sein. **Hinzugefügte** Wörter werden mit *Fade-In* oder *Typing* hinzugefügt. Da gleiche und hinzugefügte Wörter gleichzeitig und sequenziell der rechten Seite hinzugefügt werden, spielen die gelöschten Wörter hier auch keine Rolle.

#### 4.5.2 Revision Animationen

Um für revisionierte Texte nicht nur die Veränderungen zwischen zwei Revisionen zu visualisieren, sondern auch die Veränderungen über die gesamte Bearbeitungszeit, wurde ein etwas anderer Ansatz gewählt. Ausgangspunkt ist der Vergleich zwischen der ersten und zweiten Revision. Beide Texte werden nebeneinander dargestellt. In der ersten Revision werden alle Wörter rot markiert, die in der zweiten Revision gelöscht wurden. In der zweiten Revision werden alle Wörter grün markiert, die hinzugefügt wurden. Man kann über einen Button zur nächsten Revision schalten. Dann startet eine Animation, die alle Veränderung von rechts nach links übernimmt. Dabei blenden die Wörter rechts sequenziell aus und links wieder ein. So werden beide Texte sozusagen zu einem Classic Diff vereint. Die Überbleibsel von Revision zwei blenden aus und Revision drei wird von rechts außerhalb des Fensters eingeschoben. Nun findet ein neuer Vergleich zwischen Revision zwei und Revision drei statt. Auf der rechten Seite werden wieder alle hinzugefügten Wörter grün markiert. Auf der linken Seite werden alle gelöschten Wörter wieder rot markiert. Da es aber bereits aus dem vorherigen Vergleich rot markierte Wörter gibt, wird die Farbe so angepasst, dass erkennbar ist, in welcher Revision die Wörter gelöscht wurden. Um so niedriger die Revision ist, in dem das Wort gelöscht wurde, desto transparenter ist die Markierung. Außerdem gibt es die Möglichkeit, längere, zusammenhängende Abschnitte gelöschter Wörter zu Glyphen zusammenzufassen. Fährt man mit der Maus über diese Glyphen, wird angezeigt, welcher Text verborgen wurde und in welcher Revision dieser gelöscht wurde. Die Animationsgeschwindigkeiten sind wieder individuell einstellbar. Außerdem lässt sich einstellen, gelöschte Wörter auf der linken Seite, nach einer bestimmten Anzahl von übernommenen Revisionen verschwinden zu lassen.

Ziel dieser Visualisierung war es, eine Übersicht, über alle Veränderungen zwischen den Revisionen, stückweise aufzubauen. Allerdings sind bereits einige Fehler aufgefallen, die diese Ansicht hat. Zum Beispiel entsteht bei Übernahme von rechts nach links, auf der rechten Seite ein Text, der in keiner Revision wirklich existiert. Man könnte die Wörter stehen lassen. Aber das Ausblenden und Einschieben der neuen Revision wirkt nicht intuitiv. Letztendlich wurde entschieden, dass es besser ist, eine statische Visualisierung für revisionierte Texte zu erstellen.

### 4.6 Multi Revision Browser

Der Multi Revision Browser wurde entwickelt, um mehrere Revisionen in einer statischen Darstellung zu visualisieren. Ähnlich wie beim Vertical Aligned werden Abschnitte, die in allen Revisionen gleich sind, nebeneinander angezeigt.

Die ganze Visualisierung kann als Tabelle angesehen werden, bei der eine Spalte eine Revision darstellt. Die zeilenweise Aufteilung ergibt sich durch Wörter, die in allen Revisionen gleich sind. Die ursprüngliche Reihenfolge der Wörter bleibt in allen Spalten erhalten, sodass jede Revision vollständig rekonstruiert werden kann. Die Visualisierung wird auf Wortbasis erstellt. Beginnen beispielsweise alle Revisionen mit den gleichen fünf Wörtern, wird die erste Zeile der Tabelle als *all-equal*-Zeile markiert, wie es in Abbildung 4.6 der Fall ist. In diesen Zeilen ist der Inhalt jeder Zelle in jeder Revision gleich. Sobald sich ein Wort von denen der anderen Revisionen unterscheidet, wird eine neue Zeile erstellt. Diese Zeilen haben dann in mindestens einer Revision Unterschiede zu den anderen Revisionen und sind somit nicht *all-equal*. In der vollständigen Visualisierung wechseln sich immer Zeilen, in denen alle Revisionen übereinstimmen, mit Zeilen, in denen es zwischen den Revisionen Unterschiede gibt, ab. Um die Übersichtlichkeit zu verbessern, werden *all-equal*-Zeilen farblich hinterlegt.

In den meisten Fällen sind nur kurze Abschnitte in allen Revisionen gleich. So entstehen lange Textzeilen, in denen sich die Revisionen unterscheiden. Um diese Abschnitte besser analysieren zu können, wird der oben beschriebene Algorithmus rekursiv auf diese Zeilen angewandt. Zuerst wird geprüft, ob sich zwei oder mehr Revisionen in einer Zeile ähnlich sind. Ist dies der Fall, werden die entsprechenden Zellen zusammengefasst und eine Subvisualisierung für den Vergleich dieser Revisionen erstellt. Da hier wieder Zeilen entstehen können, die sich in den Revisionen unterscheiden, könnte man eine weitere Rekursion durchführen.

Wieviele Rekursionen sinnvoll sind, ist von der Länge des Textes, der Anzahl der dargestellten Revisionen und der Anzahl der Unterschiede zwischen den Revisionen abhängig. Daher kann eingestellt werden, wie tief der Algorithmus vorgehen soll. Außerdem können Zellen manuell markiert und verglichen werden. Werden mehr als zwei Zellen verglichen, erstellt die Visualisierung einen weiteren Multi-Revision-Vergleich. Werden nur zwei Zellen zum Vergleich ausgewählt, kann auch eine der vorher vorgestellten Visualisierungen, zum Beispiel Classic Diff, in die Zelle eingefügt werden. In Abbildung 4.7 werden beispielsweise die dritte und vierte Revision des Textes innerhalb eines ungleichen Abschnitts im Detail verglichen. Die komponentenbasierte Entwicklung der Visualisierungen erlaubt es, innerhalb dieser Zelle jede beliebige Visualisierung einzufügen.

## KAPITEL 4. TEXTVERGLEICHSVISUALISIERUNGEN

---

'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With
No Letup in Volatility in China	China Joining the Rebound	China Joining the Rebound
By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —
Markets across Asia	Markets across Asia	Shanghai had its first positive day in a week and markets rallied Asia on Thursday after
opened higher	rose	
on Thursday following	on Thursday following	
Wall Street's strong rebound on Wednesday	Wall Street's strong rebound on Wednesday	Wall Street's strong rebound on Wednesday
, but stocks in China showed continued volatility,	, but stocks in China showed continued volatility,	. Stocks in China remained volatile, though the movements were muted compared with recent days. The Shanghai Composite Index dropped into negative territory in the last hour of trading, then shot from being down 0.65 percent to closing up 5.4 percent. Trading volumes were heavier than usual during that period – an indication of state intervention. The Shanghai and Shenzhen markets were the only major ones in the region to drop into negative territory on Thursday. The rest of the region appeared to have received a boost from hints
even in the early minutes of trading. The	although muted compared with recent days. Asia	
markets appeared to be headed toward their first broad rally	markets appeared to be headed toward their first broad rally	
— if a modest one — in Asia		
since last week, helped along by	since last week, helped along by	
soothing words	hints	
from the Federal Reserve	from the Federal Reserve	
suggesting		
that interest rate	that interest rate	that interest rate
increase	increases	increases
in the United States might not be as imminent as had been believed.	in the United States might not be as imminent as had been believed.	in the United States might not be as imminent as had been believed.

**Abbildung 4.6:** Multi Revision Browser Beispiel

## KAPITEL 4. TEXTVERGLEICHSVISUALISIERUNGEN

---

'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With	'Aug. 26, 2015 Asian Shares Follow Wall Street Higher, With
No Letup in Volatility in China	China Joining the Rebound	China Joining the Rebound	China Joining the Rebound	China Joining the Rebound
By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —	By AMIE TSANG HONG KONG —
Markets across Asia opened higher on Thursday following Wall Street's strong rebound on Wednesday, but stocks in China showed continued volatility, even in the early minutes of trading. The markets appeared to be headed toward their first broad rally — if a modest one — in Asia since last week, helped along by soothing words from the Federal Reserve suggesting	Markets across Asia rose on Thursday following Wall Street's strong rebound on Wednesday, but stocks in China showed continued volatility, although muted compared with recent days. Asia markets appeared to be headed toward their first broad rally since last week, helped along by hints from the Federal Reserve	<p>ShanghaiShares hadin itsChina firstrose positivefor daythe first time in a week and markets rallied throughoutAsia on Thursday after Wall Street's strong rebound on Wednesday. Stocks in China remained volatile, though the movementsheen wereof muted comparedWall withStreet recent days.rebound. The Shanghai Composite Index droppedwas intedown negative0.65 territorypercent in the last hour of trading, then shot fromup beingand down 0.65 percent to closingclosed up 5.4 percent. Trading volumes were heavier than usual during that period – an indication of state intervention. <u>But volatility remained muted compared with recent days.</u> The Shanghai and Shenzhen markets were the only major ones in the region to drop into negative territory on Thursday. The rest of the region appeared to have received a boost from hints</p>	Shares in China rose for the first time in a week and markets rallied throughout Asia on Thursday on the heels of a Wall Street rebound. The Shanghai Composite Index was down 0.65 percent in the last hour of trading, then shot up and closed up more than 5 percent. Trading volumes were heavier than usual during that period – an indication of state intervention. But volatility remained muted compared with recent days. Shanghai and Shenzhen were the only major markets in the region to drop into negative territory on Thursday, although both recorded gains for the day. The rest of Asia appeared to have been buoyed by hints	

**Abbildung 4.7:** Beispiel für das manuelle Vergleichen innerhalb des Multi Revision Browsers

# Kapitel 5

## Daten und ihre Aufbereitung

In dieser Arbeit wurden vier verschiedene Korpora verwendet, die insgesamt aus über 16.000 Textstellen (engl. Spots) bestehen.

**VroniPlag** Der Korpus von VroniPlag besteht aus 7547 Fundstellen von Plagiatsfällen in wissenschaftlichen Arbeiten. Die einzelnen Spots sind Abschnitte aus diesen Arbeiten und den plagierten Originaltexten.

**Webis-CPC** Der an der Professur „Web Technology and Information Systems“ (Webis) der Bauhaus-Universität Weimar entstandene Korpus enthält 7853 mögliche Fundstellen für Plagiatsfälle. Der Korpus wurde mit „Mechanical Turk Crowdsourcing“ erstellt und die Fundstellen bestehen ebenfalls aus Abschnitten des Originaltextes sowie des Plagiats.

**Newsdiffs** newsdiffs.org ist eine Website, die Veränderungen von Newsartikeln bekannter amerikanischer Nachrichtenseiten dokumentiert und darstellt. In dieser Arbeit wurde ein Auszug von 592 Artikeln und ihren Revisionen verwendet. Die Spots bestehen hier aus mindestens zwei und bis zu 34 Revisionen.

**Webis-TRC** Ebenfalls an der Webis Professur entstanden, bietet dieser Korpus 150 Spots, die aus mehreren hundert Revisionen bestehen. Die Spots sind Arbeiten von Autoren zu einem spezifischen Thema. Jeder Bearbeitungsschritt ist als Revision gespeichert.

### 5.1 Vorverarbeitung

Um das Einlesen der Daten in das Framework zu erleichtern, wurden alle Korpora in eine gleiche Datenstruktur gebracht. Dabei besteht jeder Korpus aus einem Ordner, der mit den Fundstellen als Unterordnern gefüllt ist. Jeder Spot

hat außerdem eine innerhalb des Korpus eindeutige ID. Die Texte in den Spots sind im JSON-Format gespeichert. Handelt es sich um Texte mit Quelle-Plagiat Beziehung, sind die JSON-Dateien auch immer als Quelle und Paraphrase benannt. Im Falle von revisionierten Texten haben die Revisionen wiederum eine eindeutige ID, die aufsteigend nach Aktualität vergeben ist.

### **5.1.1 Tokenisierung mit UIMA**

Teil der in Javascript implementierten Differenzalgorithmen, die in diesem Framework verwendet werden, ist eine Tokenisierung der eingegebenen Strings. Allerdings wird dabei nur anhand von Leerzeichen getrennt. In mehreren Visualisierungen war eine feinere Tokenisierung nötig. Dazu wurden in einem Vorberechnungsschritt JSON-Dateien mit Apache UIMA<sup>1</sup> erstellt, die eine Einteilung des Textes in Paragraphen, Sätze und Tokens vornimmt. Die UIMA Bibliothek ist in Java implementiert. Die Texte werden von Textdateien eingelesen, tokenisiert und als JSON-Dateien ausgegeben.

### **5.1.2 Sortierung nach Textlänge/ Anzahl Änderungen**

Da es sich um einen großen Umfang an Texten handelt, wurden, ebenfalls mit Java, einige Metainformationen erstellt um das Navigieren in den Korpora zu erleichtern. Eine Metainformation war zum Beispiel die Textlänge. Um die Spots eines Korpus danach sortieren zu können, wurde eine Liste erstellt, welche aus den Spot IDs aufsteigend sortiert nach Textlänge besteht. Eine weitere Liste, die nur für revisionierte Texte erstellt wird, ist die Anzahl der Revisionen aufsteigend sortiert.

---

<sup>1</sup><https://uima.apache.org/>

# Kapitel 6

## Entwicklungsframework für Textvergleichsvisualisierungen

Das Entwicklungsframework wurde für Google Chrome<sup>1</sup> entwickelt und läuft auf einem lokalen Apache<sup>2</sup> Host. Da das Framework als Polymer-Element erstellt wurde, muss es in der *index.html* des Hosts verankert werden. Außerdem muss man hier in einem kurzen Script die Callbacks für Tastatur und Maus sowie für Größenveränderung des Fensters, mit Funktionen des Frameworks verbinden (siehe Listing 6.1).

**Listing 6.1:** Weiterleitung des *Resize-Events* zum Framework

```
1 <main-visualisation id="mv"></main-visualisation>
2
3 <script>
4     // Entwicklungsframework
5     var mainVisu = document.getElementById('mv');
6
7     // Resize Event
8     window.onresize = function(event) {
9         mainVisu.resize(event);
10    }
11 </script>
```

Das Framework und alle Visualisierungen sind als Polymer-Elemente umgesetzt. Der grundsätzliche Aufbau dieser Elemente ist wie folgt.

### 6.1 Polymer-Elemente

Polymer ist eine von Google entwickelte Bibliothek, um Webkomponenten (engl. Web Components) zu erstellen. Webkomponenten ermöglichen eine kom-

---

<sup>1</sup>Version 48.0.2564.97 (64-bit)

<sup>2</sup><https://wiki.ubuntuusers.de/Apache/>

ponentenbasierte Entwicklung für Websites als W3C<sup>3</sup> Standard. So ist es möglich, eigene HTML-Elemente zu definieren.

Es bietet sich an, für jedes Polymer-Element eine eigene HTML-Datei anzulegen. In Listing 6.2 sieht man den allgemeinen Aufbau dieser Datei. Das Element selbst besteht aus drei Abschnitten. In dem **template**-Abschnitt (a) können weitere HTML- und Polymer-Elemente definiert werden. Diese sind dann Kinder des aktuellen Polymer-Elements. In dem **style**-Abschnitt (b) können CSS-Definitionen für das Polymer-Element und alle Kinder festgelegt werden. Und in dem **script**-Abschnitt (c) können Javascript Funktionen definiert werden. Polymer bietet einige Funktionen, die automatisch aufgerufen werden, wenn man das Element erstellt, anhängt oder löscht. Ein **dom-module** Tag (d), dessen ID mit der ID des im script erstellten Element übereinstimmen und ein Bindestrich in sich tragen muss, umschließt template, style und script. Am Ende des scripts registriert *registerElement* (e) das erstellte Element mit der ID beim Dokument. Wenn man nun die HTML-Datei importiert, kann das Polymer-Element als HTML-Element verwendet werden. Außerdem ist der Zugriff von außen auf alle Funktionen und Membervariablen möglich.

**Listing 6.2:** Template für Polymer-Elemente

```
1 <dom-module id="polymer-element">      <!--(d)-->
2
3   <template>
4     <!-- (a) HTML-Abschnitt -->
5   </template>
6
7   <style>
8     /* (b) CSS-Abschnitt */
9   </style>
10
11  <script>
12    // (c) Script Abschnitt
13    var PolymerElement = Polymer.Class({
14
15      is: "polymer-element",
16
17      properties: {
18        exampleString: {
19          type: String,
20          value: "someString"
21        }
22      },
23
24      // Konstruktor
25      factoryImpl: function() {
26        // ...
27      },
28
29
30  
```

---

<sup>3</sup>World Wide Web Consortium: [https://de.wikipedia.org/wiki/World\\_Wide\\_Web\\_Consortium](https://de.wikipedia.org/wiki/World_Wide_Web_Consortium)

```
31      // Ausgeführt wenn das Element angehängt wurde
32      attached: function() {
33          // ...
34      }
35  });
36
37
38      // Element bei Dokument registrieren (e)
39      document.registerElement("polymer-element", PolymerElement);
40
41  </script>
42
43  <dom-module>    <!--(d)-->
```

## 6.2 Das Framework

Das Framework versorgt die Visualisierungen mit Daten und ermöglicht eine beliebige Auswahl von Daten, Visualisierungen und Differenzalgorithmen zur Laufzeit. In Abbildung 6.1 sieht man das Layout. In der Mitte wird die ausgewählte Visualisierung in einem Fenster (a) dargestellt. In der oberen Zeile (b) erfolgt die Auswahl der Texte. Zuerst muss man einen Korpus (b1), anschließend einen Spot (b2), in den entsprechenden Dropdown-Listen selektieren. Enthält der Korpus revisionierte Texte, werden in der oberen grauen Zeile zwei weitere Dropdown-Listen eingeblendet, mit denen die zu vergleichenden Revisionen ausgewählt werden können. Der Random Data Button (b3) bestimmt zufällige Daten. Dabei kann die Auswahl eingeschränkt werden, zum Beispiel in Bezug auf die Textlänge oder Anzahl der Revisionen. Die Einschränkung bezieht sich auf den aktuellen Korpus. Man kann beispielsweise zufällige Texte langer, mittlerer oder kurzer Länge erhalten, wobei sich „kurz“ auf die 1/3 kürzesten Texte des gewählten Korpus bezieht.

Bei (c) kann der Differenzalgorithmus ausgewählt werden, mit dem der Vergleich der Texte erfolgt. In der linken Sidebar (d) selektiert man die Visualisierung, mit dem der Vergleich der Texte dargestellt wird. Der Random Everything Button (e) wählt sowohl einen zufälligen Spot als auch eine zufällige Visualisierung und einen zufälligen Differenzalgorithmus aus.

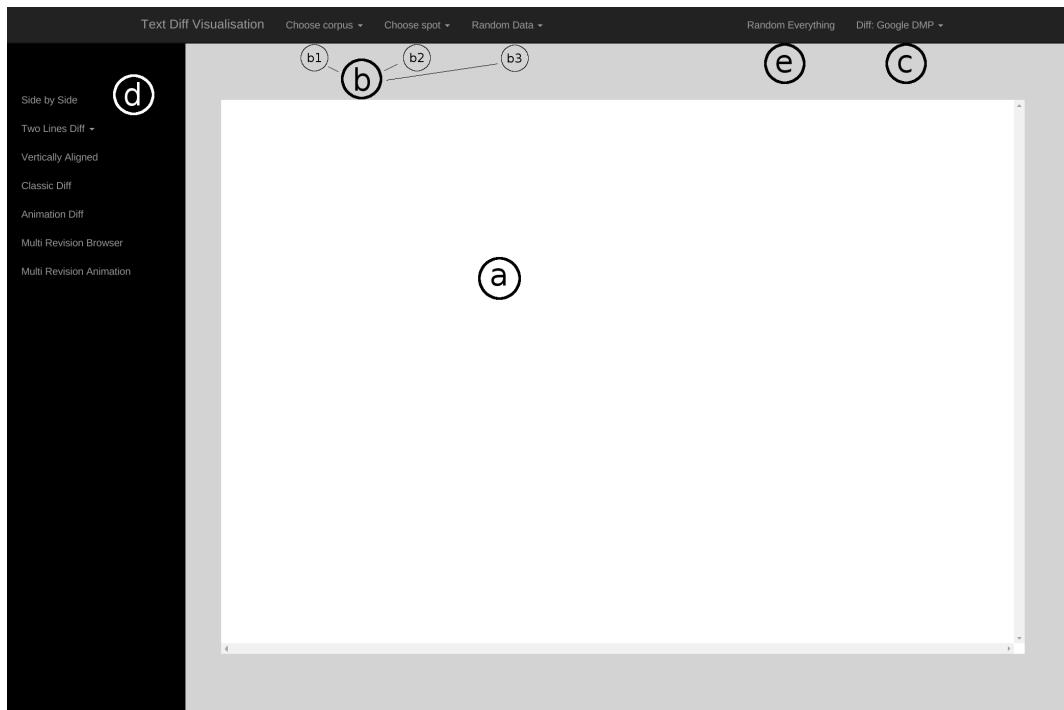
### 6.2.1 HTML-Abschnitt

Der HTML-Abschnitt des Frameworks ist relativ komplex. Buttons, Dropdown-Listen sowie Top- und Sidebar wurden mithilfe von Bootstrap<sup>4</sup> erstellt. Bootstrap ist ein HTML, CSS und Javascript Framework, welches unter anderem Templates für Buttons und Naviagationbars enthält. Die Templates der Top-

<sup>4</sup><http://getbootstrap.com/>

<sup>5</sup><http://getbootstrap.com/components/#navbar>

## KAPITEL 6. ENTWICKLUNGSFRAMEWORK FÜR TEXTVERGLEICHSVISUALISIERUNGEN



**Abbildung 6.1:** Framework Startbildschirm

und Sidebar<sup>6</sup> wurden leicht angepasst. Buttons und Dropdown-Menüs sind als Standardelemente implementiert, wie es in Listing 6.3 der Fall ist.

**Listing 6.3:** Bootstrap-Dropdown-Menü für die Auswahl des Korpus

```
1 <li class="dropdown">
2   <a class="dropdown-toggle" data-toggle="dropdown" role="button">
3     <span id="chosenCorpus">Choose corpus</span>
4     <span class="caret"></span>
5   </a>
6
7   <ul class="dropdown-menu dataset">
8     <li><a id="vroniplag" class="dropdown-link">VroniPlag</a></li>
9     <li><a id="webis-cpc" class="dropdown-link">Webis-CPC</a></li>
10    <li><a id="essays" class="dropdown-link">Essays</a></li>
11    <li><a id="newsdiffs" class="dropdown-link">Newsdiffs</a></li>
12  </ul>
13 </li>
```

Das Layout für die verwendeten Klassen ist im CSS-Abschnitt von Bootstrap definiert. Wichtig ist, dass alle dynamischen Elemente eine ID bekommen. Damit kann den Elementen der Liste jeweils eine Mausclick-Funktion hinzugefügt werden, die zum einen die HTML-Elemente des Frameworks so anpasst, dass

<sup>6</sup><http://startbootstrap.com/template-overviews/simple-sidebar/>

sie die aktuelle Auswahl von Korpus, Spot, usw. anzeigen, zum anderen Funktionen zum Data Handling aufrufen.

### 6.2.2 Data Handling

Wird ein Spot, eine Visualisierung oder ein Differenzalgorithmus ausgewählt, überprüft eine Funktion ob genug Auswahl getroffen wurde um eine Visualisierung zu erstellen (siehe Listing 6.4). Dazu hat das Framework Strings als Membervariablen, die angeben dass noch nichts ausgewählt wurde (z.B. *corpus*=„not-choose“), bzw. von den Callback-Funktionen der Buttons einen eindeutigen Wert zugewiesen bekommen (z.B. *corpus*=„vroniplag“).

Die lokale Datenstruktur ist so aufgebaut, dass jeder Korpus einen Ordner, mit den Spots als Unterordern besitzt. Der Name eines Spot-Ordners entspricht einer eindeutigen ID innerhalb dieses Korpus. Wurde ein revisionierter Spot ausgewählt, werden für alle Dateien, die in dem entsprechenden Ordner liegen, zwei Buttons erstellt, die in die Dropdown-Listen für Revision A und Revision B angehängt werden. Anschließend werden diese Dropdown-Listen eingeblendet, damit die zu vergleichenden Revisionen ausgewählt werden können. Im Falle eines Quelle-Plagiat Spots werden die Membervariablen für Revision A und B auf „no-revision“ gesetzt. Die Dateien in diesen Spot-Ordnern sind immer als Quelle und Paraphrase benannt und können ohne weiteres geladen werden. Ist die Auswahl vollständig, werden die Dateien mit jQuery<sup>7</sup> geladen. Dazu werden zweigetJSON Funktion ausgeführt und erst fortgefahren, wenn beide abgeschlossen sind.

**Listing 6.4:** Daten einlesen mit jQuery

```
1 | loadJSON: function(filepath) {
2 |   return $.getJSON(filepath, {format: 'json'}, function(data){});
3 | },
4 |
5 | createVisualisation: function() {
6 |
7 |   $("#visuContainer").html("");
8 |
9 |   // Return if spot or visualisation are not chosen
10 |   if (this.corpus == 'not-chosen' || this.visu == 'not-chosen' ||
11 |       this.spot == 'not-chosen' ||
12 |       this.revisionA == 'not-chosen' || this.revisionB == 'not-chosen')
13 |     return;
14 |
15 |   // Save choice in URL
16 |   this.updateUrl();
17 |
18 |   // Load original and paraphrase
19 |   $.when(this.loadJSON(this.originalPath), this.loadJSON(
    this.paraphasePath)).done(function(original, paraphrase){
      // Create Visualisation...
```

---

<sup>7</sup><https://jquery.com/>

Ist eine Visualisierung ausgewählt, die nur zwei Texte vergleicht, wird der Vergleich erstellt und an das Element der Visualisierung übergeben. Visualisierungen, die mehrere Texte vergleichen, bekommen als Übergabeparameter den Pfad zum ausgewählten Spot. Die Vergleiche finden erst später statt.

### 6.2.3 Diff-Klasse

Um eine universelle Repräsentation des Vergleiches zweier Texte zu schaffen, wurde eine Javascript Klasse für Vergleiche erstellt. Diese bekommt als Übergabeparameter das von jQuery eingelesene JSON-Objekt. In dem Objekt befinden sich sowohl eine tokenisierte Version der Texte als auch der zusammenhängende Text als String. Ein Beispiel für ein Token-Objekt ist in Abbildung 6.2. Die Tokens können dem Fließtext durch Start- ("begin") und Endindices ("end") eindeutig zugeordnet werden. Außerdem ist der Wortstamm ("stem") und die Wortart ("pos") gespeichert.

```
{
    "sofa":1,
    "begin":1840,
    "end":1848,
    "lemma":"umfasste",
    "stem":"umfasst",
    "pos":"NP"
},
```

Abbildung 6.2: Beispiel für ein Token-Objekt in JSON

Um die Tokens in den Javascript Differenzalgorithmen zu benutzen, wird jedem Token beider Texte eine eindeutige ID zugewiesen und in einer Map gespeichert. Beginnt der originale Text beispielsweise mit „Das“, wird dem Wort „Das“ die ID 0 zugewiesen. Anschließend werden zwei Strings erstellt, die Originaltext und Paraphrase repräsentieren. Diese bestehen aus den IDs der Wörter, getrennt durch Leerzeichen und in ihrer ursprünglichen Reihenfolge. Diese Strings können dann in die Differenzalgorithmen gegeben werden. Die Ausgabe des gewählten Algorithmus kann anschließend mit der Map wieder zu den ursprünglichen Tokens umgewandelt werden. Einige Visualisierungen benutzen HTML-Span Repräsentationen der Tokens, mit dem Typ (Equal, Deletion, Insertion) als Klasse. Dafür hat die Diff-Klasse eine Funktion, die für jeden Token einen Span erstellt und alle Spans als Array ausgibt.

### 6.2.4 URL

Ein Feature des Frameworks ist es, die aktuelle Auswahl von Visualisierung und Spot in der URL zu speichern. So können Beispiele einfacher gespeichert und geteilt werden. Durch Anhängen eines Fragezeichens, kann der URL ein Querystring hinzugefügt werden, auf den man in Javascript mit `location.search` zugreifen kann. Bei Auswahl von Korpus, Visualisierung und Spot werden dem Querystring Werte hinzugefügt, die mit „&“ voneinander getrennt werden. Für revisionierte Korpora werden ebenfalls die ausgewählten Revisionen in der URL gespeichert. So entsteht beispielsweise die URL in Abbildung 6.3.



Abbildung 6.3: Beispiel URL

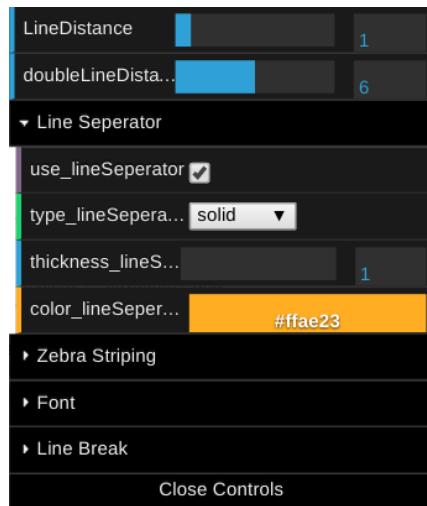
Wird die Website aufgerufen, überprüft das Framework die URL auf einen Querystring und aktualisiert die entsprechende Auswahl, falls eine vorhanden ist.

## 6.3 Parameter-Widget

Jede Visualisierung erstellt bei ihrer Initialisierung ein Parameter-Widget, mit der ihre eigenen Parameter verändert werden können. Das Widget wurde mit dat.gui<sup>8</sup> erstellt. Dat.gui ist eine Javascript Bibliothek, mit der einfache Graphical User Interfaces (GUI) in Webumgebungen erstellt werden können. Außerdem verfügt dat.gui über die Möglichkeit, die aktuelle Einstellung in einem JSON-String zu speichern. Um die gewählte Einstellung auch in der aktuellen Session beizubehalten, zum Beispiel wenn die Seite nur aktualisiert wird oder der Spot gewechselt wird, wird der Local Storage verwendet. Damit können Daten lokal gespeichert werden, die mit Beendigung der Session gelöscht werden.

Jedes Parameter-Widget ist so aufgebaut, dass es einen Konstruktor und eine Initialisierungsfunktion besitzt. In der Initialisierungsfunktion werden zuerst eine Klasse für die veränderbaren Daten erstellt. Anschließend wird eine Instanz der Daten und eine der dat.GUI erstellt. Der dat.GUI Instanz können GUI-Elemente und Ordner hinzugefügt werden. Den Ordner können weitere Unterordner und GUI-Elemente hinzugefügt werden. Verfügbare GUI-Elemente sind Nummern-Slider, Farben-Picker, Checkbox und Buttons. Jedes GUI-Element kann man durch Callback-Funktionen erweitern, die ausgeführt

<sup>8</sup><http://workshop.chromeexperiments.com/examples/gui>



**Abbildung 6.4:** Beispiel für ein dat.gui Parameter-Widget

werden, während der Wert sich ändert oder wenn die Veränderung beendet wurde. Will man den Local Storage mit den Daten des Parameter-Widgets füllen, bietet sich eine Funktion *onFinishChange* an, die unter dem gleichen Key wie in dem Widget, den Wert abspeichert. Bei Initialisierung des Widgets wird der Local Storage auf Keys überprüft, die mit denen des Widgets übereinstimmen. Wurden welche gefunden, wird der Wert aus dem Local Storage übernommen. Besonders zum Einsatz kommt dies bei den Animation Diffs, da die Seite neu geladen wird, wenn man die Animation neu startet.

Die Initialisierungsfunktion gibt die erstellte Instanz der Daten zurück, damit sie in der Visualisierung verwendet werden können. Alle Visualisierungen funktionieren auch ohne Parameter-Widget, dazu muss ein Element definiert werden, welches Default-Werte für alle, durch das Widget veränderbaren, Parameter bereitstellt. In der Arbeit mit diesem Framework wurde allerdings immer ein Widget benutzt. Das Framework stellt dazu einen globalen Container zur Verfügung, auf den alle Visualisierungen Zugriff haben, um ihr Widget dort an- bzw. abzuhängen.

## 6.4 Visualisierungen

Für jede Visualisierung wird ein Polymer-Element erstellt, welches das Parameter-Widget initialisiert und, wenn nötig, andere Polymer-Elemente importiert.

### 6.4.1 Vertical Aligned

Die Vertical Aligned Visualisierung besteht aus zwei gleich breiten, nebeneinander platzierten HTML-Divs, die mit den Spans des Vergleichs und Zeilenumbrüchen gefüllt werden. Dazu wurde eine Javascript Klasse erstellt, von der jeweils für Quelle und Paraphrase eine Instanz erstellt wird. Diese fügt sequentiell die Spans des Vergleichs hinzu, zählt die Länge der aktuellen Zeile und fügt einen Zeilenumbruch hinzu, falls diese die per Widget wählbare maximale Zeilenlänge überschreitet. Außerdem werden vor gleichen Spans Zeilenumbrüche eingefügt, sodass sie nebeneinander stehen.

### 6.4.2 Two Lines Diff

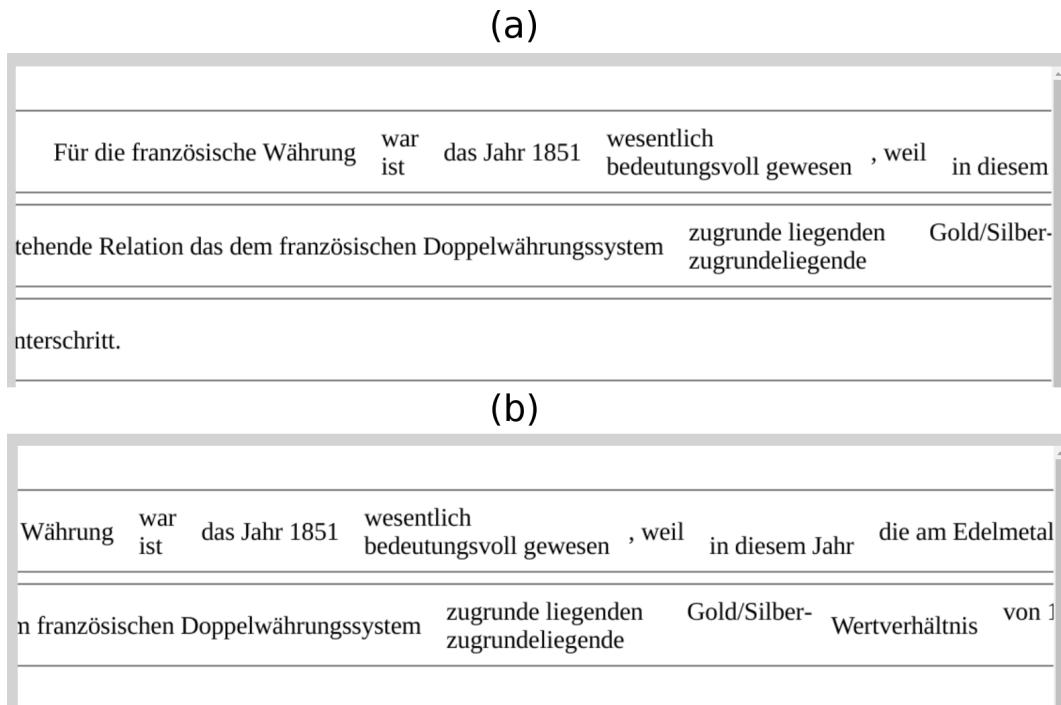
Für die Two Lines Diff Visualisierung wurden zwei Versionen erstellt. In der einen werden absolut positionierte HTML-Spans verwendet, in der anderen die Textfunktion von Canvas. Die Berechnung der Positionen ist dabei gleich. Die Ergebnisse des Vergleichs werden wieder sequentiell bearbeitet. Jede globale Zeile wird von einem Polymer-Element repräsentiert, welches Funktionen zum Hinzufügen und Entfernen von Spans in der mittleren (shared-line) oder oberen und unteren Zeile (double-line) besitzt. Bevor die Schleife über alle Spans beginnt, wird für die erste Zeile ein solches Objekt erstellt. Außerdem werden Variablen initialisiert, welche die Länge der äußeren und mittleren Zeilen angeben.

Während die y-Koordinate der Spans durch die Position der globalen Zeile und einem durch das Parameter-Widget festgelegten Offset für double-lines gegeben ist, muss die x-Koordinate pixelgenau berechnet werden. In der Hauptschleife wird das zu bearbeitende Span je nach Typ (equal, deletion, insertion) unterschieden und an den entsprechenden Teil der globalen Zeile angehängt. Nach jedem Anhängen wird die Länge der Zeile (Oben, Unten oder Mitte) aktualisiert, dabei wird vom linken Fensterrand bis zum Ende des letzten Wortes der Zeile gemessen. Equal-Spans werden der mittleren Zeile hinzugefügt. Ist die maximale Länge beider äußeren Zeilen länger als die mittlere Zeile, stehen also äußere Zeilen vor der aktuellen, wird die mittlere Zeile so verlängert, dass der Equal-Span nach dem Ende der längsten äußeren Zeile beginnt. Zwischen äußeren und mittleren Zeile wird ein durch das Widget einstellbares Padding gesetzt. Deletion-Spans werden in die obere Zeile, Insertion-Spans in die untere Zeile nach dem Ende des letzten Wortes der mittleren Zeile gesetzt.

Nach jedem Anhängen eines neuen Wortes, wird überprüft ob die aktuelle Zeile über die Fensterbreite hinaus geht. Ist dies der Fall, wird eine neue globale Zeile erstellt und unter der vorherigen positioniert. In dem Widget ist einstellbar, ob der Zeilenumbruch auf Wort- oder Zeichenbasis geschehen soll.

Bei Umbrüchen nach Wörtern, wird das als letztes angehängte Wort, welches also über die Fensterbreite hinaus geht, von der aktuellen Zeile entfernt und an die neu erstellte Zeile angehängt. Bei zeichenbasierten Umbrüchen, werden aus dem Span zwei neue erstellt, die auf beide Zeilen aufgeteilt werden. Anschließend werden Zeilenlängen wieder auf 0 gesetzt und die Schleife fährt mit den weiteren Spans des Vergleichs und der neuen Zeile fort.

Teil der Funktionalitäten der globalen Zeilen ist auch die Hintergrundfarbe zu ändern, Linienseparatoren zu setzen und die Schriftgröße zu verändern. Werden diese Parameter in dem Widget verändert, können in einer Schleife die entsprechenden Funktionen aller globalen Zeilen aufgerufen werden. Eine weitere Funktion ist es, alle Wörter der äußeren Zeilen miteinander zu vergleichen um Wortabänderungen zu erkennen. Dazu wird wieder die Levenshtein-Distanz aller Wörterpaare berechnet. Liegt diese über dem per Widget wählbaren Schwellwert, werden beide Wörter mit einer automatisch generierten Farbe markiert. Die Levenshtein-Distanz ergibt sich aus der Anzahl von Einfüge-, Lösch- und Ersetzoperationen, die gebraucht werden um das erste Worte zum zweiten zu Transformieren.



**Abbildung 6.5:** Horizontales Scrollen mit Canvas

Eine Erweiterung der Canvas Version ist das horizontale Scrollen. Anders als bei der HTML-Version wird hier der gesamte Text in einer Zeile dargestellt.

Globale Zeilenumbrüche ergeben sich durch mehrere Canvas Fenster, die mit einem horizontalen Offset versehen sind. Dieser Offset lässt sich durch Maus-Srollen manuell verändern, sodass in jedem Canvas Fenster der Text sich von rechts nach links (scrolldown) oder links nach rechts (scrollup) bewegt. In Abbildung 6.5 (a) sieht man die Darstellung zu Beginn in drei Canvas Fenstern. Beim Scrolling wird der Text in allen Fenstern nach links verschoben, wodurch die letzte Zeile leer ist und verschwindet (b).

### 6.4.3 Multi Revision Browser

Die Multi Revision Browser Visualisierung wird mit einer HTML-Tabelle aufgebaut. Dazu wurden drei Polymer-Elemente erstellt. Das Erste (*Multi-Revision-Browser*) baut die gesamte Visualisierung auf und kommuniziert mit dem Parameter-Widget. Da diese Visualisierung nur für revisionierte Texte konzipiert ist, werden erst hier die Daten eingelesen und Vergleiche ausgeführt. Die Visualisierung beginnt damit, alle Revisionen mit jQuery einzulesen. Die Fließtexte und Tokens werden in separaten Arrays gespeichert.

Anschließend werden die ersten beiden Revisionen miteinander verglichen. Für die Revisionen werden HTML-Divs erstellt, an die die Spans angehängt werden. Man könnte dafür auch Arrays verwenden, aber Divs eignen sich hier für die sequentielle Bearbeitung der Spans besser, da ein Span nur ein Parent-Element besitzen kann. Deletion- und Equal-Spans bilden die erste Revision, Insertion- und Equal-Spans die zweite. Für Equal-Spans müssen Kopien erstellt werden, weil sie in zwei verschiedenen Containern vorkommen. Da die gesamte Visualisierung an Spans ausgerichtet wird, die in allen Revisionen gleich sind, ist es wichtig solche zu erkennen und einen Zugriff auf die HTML-Elemente zu ermöglichen. Dazu bekommen Equal-Spans beim Anhängen an die Divs eine ID, die Aufschluss darüber gibt, in welchen Vergleich sie gleich sind. Im ersten Vergleich ist dies zwar trivial, trotzdem bekommen sie hier das Attribut id0 mit einer Laufvariable als ID.

**Listing 6.5:** Beispiel für einen Span der in den ersten beiden Vergleich vorkommt

```
1 | <span class="equal" id0="9" id1="9" all_equal="true">Virginia</span>
```

Nun wird der zweite Vergleich durchgeführt. Der Text der zweiten Revision wird mit dem Text der dritten Revision verglichen. Für die dritte Revision wird wieder ein Div erzeugt, an das Insertion- und Equal-Spans angehängt werden. Die zweite Revision existiert bereits als Div aus dem ersten Vergleich. Die Spans dieses Divs müssen jetzt mit den Equal- und Deletion-Spans des neu durchgeföhrten Vergleichs verglichen werden. Da Equal- und Insertion-Spans des ersten Vergleichs inhaltlich den Equal- und Deletion-Spans des zweiten Vergleichs entsprechen, können die Typen der Spans einfach sequentiell mit-

einander verglichen werden. Haben beide den Typ Equal, sind sie in allen drei Revisionen vorhanden und bekommen ebenfalls die id0 des ersten Vergleichs. Spans, die in dem zweiten Vergleich gleich sind, bekommen wieder eine ID, die auf den Vergleich schließen lässt, also id1. Dies wird für alle übrigen Revisionen fortgesetzt, wobei rekursiv alle vorherigen Revisionen auf gleiche Spans überprüft werden und gegebenenfalls mit weiteren IDs versehen werden. Nach Hinzufügen der letzten Revision, bekommen Spans die in allen Revisionen vorhanden sind das Attribut *all\_equal*.

In Listing 6.5 sieht man beispielsweise einen Span aus dem Div der ersten Revision. In diesem Beispiel wurden drei Revisionen miteinander verglichen und dieses Wort ist in allen gleich. Darum hat es das Attribut *all\_equal* und für jeden Vergleich eine ID. Die entsprechenden Spans in den Revisionen zwei und drei haben durch den Algorithmus die gleiche id1 zugewiesen bekommen. Obwohl es drei unabhängige Elemente sind, die an verschiedenen Positionen ihrer jeweiligen Parents stehen, können sie eindeutig einander zugeordnet werden. Damit kann jetzt eine Ausrichtung nach *all\_equal* Elementen erfolgen.

Dazu wird das Polymer-Element *Revision-Grid* initialisiert, welches eine HTML-Tabelle erstellt und mit Zeilen (*Revision-Lines*) füllt. Der Algorithmus bearbeitet die Spans des ersten Revision-Divs sequentiell und unterscheidet zwischen Spans mit und ohne *all\_equal* Attribut. In einer Boolean Variable wird gespeichert, ob der letzte Span *all\_equal* war oder nicht. Unterscheidet sich das *all\_equal* Attribut von dem des vorherigen Spans, wird eine neue *Revision-Line* erstellt, die eine Referenz auf die Tabelle als Übergabeparameter bekommt und so eine Zeile hinzufügen kann. *Revision-Lines* erstellen außerdem für jede Revision eine Zelle, die von dem *Revision-Grid* mit Inhalt gefüllt werden kann.

Die aktuelle *Revision-Line* ist als Membervariable immer bekannt. In einer Schleife werden alle Spans der ersten Revision bearbeitet. Ist der zu bearbeitende Span *all\_equal* wird er an die erste Revision (also Zelle) der aktuellen *Revision-Line* angehängt. Außerdem werden über die ID des letzten Vergleichs alle korrespondierenden Spans der anderen Revisionen den entsprechenden Zellen der aktuellen Zeile hinzugefügt. Ist der aktuelle Span nicht *all\_equal*, müssen die Revisionen unterschiedlich bearbeitet werden. Hier zeigt sich der zu Anfang erwähnte Vorteil der HTML-Divs gegenüber Arrays. Da Spans nur einen Parent haben, werden sie automatisch aus den Revisions-Divs entfernt, wenn sie einer Zeile hinzugefügt werden. So sind nur Spans in den Revisions-Divs die noch nicht zugeordnet wurden und der aktuelle Span steht immer an erster Stelle (*firstChild*). Also kann in einer Schleife solange das erste Kind jeder Revision der aktuellen Zeile hinzugefügt werden, bis ein *all\_equal*-Span vorkommt. Nach dem Hinzufügen aller nicht-*all\_equal*-Spans steht so an erster Stelle aller Revision-Divs wiederum ein *all\_equal*-Span. Also wird eine neue

Zeile erstellt und fortgefahren, bis alle Revision-Divs leer sind.

Eine sinnvolle Erweiterung für das Erstellen von neuen Zeilen ist es, den bisherigen Inhalt der aktuellen Zeile zu überprüfen. Oft entstehen sehr kurze Zeilen, wenn zum Beispiel nur ein Stoppwort in allen Revisionen gleich ist. Deswegen wird überprüft, ob die aktuelle Zeile mindestens zwei Wörter besitzt, von denen mindestens eins ein Nicht-Stoppwort ist. Ist dies nicht der Fall, wird die Zeile gelöscht und der Inhalt der vorherigen Zeile hinzugefügt.

Zu jeder Zelle jeder *Revision-Line* werden Click-Callbacks hinzugefügt, damit diese manuell verglichen werden können. Zwei ausgewählte Zellen werden zusammengefügt. Dazu wird die zweite Zelle gelöscht und der *colspan* der ersten Zelle auf zwei erhöht. Der Inhalt dieser Zelle wird ebenfalls gelöscht und ein Diff mit dem Inhalt der beiden Zellen erstellt. Dazu gibt es eine Funktion, die alle Spans beider Texte einliest und sie als Tokens in den Differenzalgorithmus einfügt. Mit dem Ergebnis wird eine Classic Diff Visualisierung erstellt und an die erste Zelle, die jetzt eine Breite von zwei Zellen hat, angehängt. Für die Zelle wird wiederum ein neuer Click-Callback erstellt, der sie wieder aufspaltet und beide Zellen mit dem ursprünglichen Inhalt füllt.

Werden mehr als zwei Zellen ausgewählt, wird für diese Zellen eine weiteres *Revision-Grid* erstellt. Dazu werden alle ausgewählten Zellen zu einer zusammengefasst und das neu erstellte *Revision-Grid* an diese Zelle gehängt. Dieses geht mit den gewählten Revisionen vor wie das ursprüngliche *Revision-Grid* mit allen Revisionen. Diese Rekursion kann auch automatisch für alle Zeilen durchgeführt werden. Zeilen, die in allen Revisionen übereinstimmen, werden mit einem Attribut markiert, damit sie hier übergangen werden können. Um *Revision-Grids* nur für ähnliche Zellen zu erstellen, wird zuerst der Jaccard-Koeffizient berechnet. Dieser ergibt sich durch die Anzahl gleicher Wörter zweier Texte geteilt durch die Anzahl aller Wörter beider Texte ( $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ ). Der Koeffizient liegt zwischen 0 und 1, wobei er näher an 1 ist, desto ähnlicher die Texte sich sind. Liegt der Jaccard-Koeffizient von mindestens zwei Zellen über einem, per Widget einstellbaren, Grenzwert, wird für diese Zellen ein *Revision-Grid* erstellt.

Um die Übersichtlichkeit der gesamten Visualisierung zu erhöhen, werden Zeilen, in denen alle Revisionen übereinstimmen, farblich markiert. In dem Widget können eine Start- und eine Endfarbe gewählt werden. Je nach Anzahl der durchgeföhrten Rekursionen wird zwischen diesen Farben interpoliert, so dass jede Rekursion eine eindeutige Farbe zugewiesen wird. Zeilen die in allen Revisionen gleich sind, werden so mit einer anderen Farbe markiert, als Zeilen die in zwei, drei,... Revisionen gleich sind.

Animatoren und ihre Funktionen

Animator	Funktion
Array Fade Out	Blendet alle Elemente des Arrays gleichzeitig aus
Array Fade In	Blendet alle Elemente des Arrays gleichzeitig ein
Array Translate	Verschiebt alle Elemente des Arrays gleichzeitig um X und Y
Array Highlight	Markiert der Reihe nach alle Elemente des Arrays
Natural Highlight	Markiert alle Elemente diagonal
Double Click Highlight	Markiert erst ein Elemente und nach einer kurzen Pause alle Elemente
Ctrl+A Highlight	Blendet die Tastenkombination <i>Ctrl+A</i> ein und markiert alles
Copy-Paste Fade	Markiert gewählte Elemente und blendet sie nach kurzem aufblenden an anderer Position wieder ein
Typing	Blendet ein Wort Zeichen für Zeichen ein und imitiert so Tippen
Caret over Span	Schiebt ein Caret Zeichen für Zeichen über ein Wort
Editing	Verändert Quellwort zu paraphrasiertem Wort

**Tabelle 6.1:** Animatoren

#### 6.4.4 Animatoren

Das Parameter-Widget für die Animation Diff Visualisierung stellt eine Auswahl für die Art der Visualisierung (sequentielles, gleiches oder alles kopieren), die Art der Kopieranimation (Translation und verschiedene Markierarten) und die Art der Einfügeanimationen (Tippen oder Fade-In) bereit. In Kapitel 4.5 wurden alle verfügbaren Animatoren, die in Tabelle 6.1 dargestellt sind, bereits beschrieben.

Um die Animation neu zu starten, besitzt das Widget einen „Restart Animation“ Button. Dieser speichert die aktuelle Auswahl im Local Storage und aktualisiert die Seite. So wird sichergestellt, dass alle HTML-Container geleert und alle laufenden Animationen beendet werden. Da die Animation beginnt, sobald das Polymer-Element angehängt wird, startet die Animation direkt nach der Aktualisierung. Das Template der Visualisierung besteht aus zwei Divs, die jeweils die Hälfte der Breite des Fensters einnehmen. Nach Initialisierung des Widgets, werden die Spans des originalen Textes an das linke Div angehängt. Außer den Divs für den Textinhalt gibt es noch zwei absolut positionierte Divs, die die ganze Höhe und Breite des Fensters einnehmen und Container für Markierungen sind. Wörter werden Zeichen für Zeichen markiert, indem eine Transformation auf ein farbiges, transparentes Div angewandt wird. Diese *Highlight-Divs* werden an eines der absolut positionierten Divs angehängt und stehen somit immer über dem Text. Das zweite absolut

positionierte Div dient als Container für den Caret, welcher auch ein farbiges, transparentes Div ist, das durch Animatoren transformiert wird. Caret und Highlight-Divs könnten auch an das selbe Parent-Div angehängt werden, wurden aber der Übersichtlichkeit halber auf zwei Parents aufgeteilt.

Je nach ausgewählten Animationen in dem Widget, werden Animatoren initialisiert und gestartet. Animatoren besitzen immer einen Konstruktor, in dem das zu animierende Objekt und die Dauer der Animation übergeben werden muss. Um die Animation zu starten, muss die *animate* Funktion aufgerufen werden, welche als Übergabeparameter eine Callback Funktion bekommt, die ausgeführt wird, sobald die Animation vollendet wurde. In Abbildung 6.6 sieht man ein Beispiel für einen Konstruktor und einer *animate* Funktion. Außerdem sieht man gut, wie das Aneinanderreihen von Animatoren funktioniert.

**Listing 6.6:** Beispiel für Callback-Funktion der Animatoren

```
1 var FlashHighlightingAnimator = function(highlightDivs, duration) {
2
3     // Member variables
4     this.highlightDivs = highlightDivs;
5     this.duration = duration;
6
7     // Init durations
8     this.fadeInDuration = duration / 2;
9     this.fadeOutDuration = duration / 2;
10 };
11
12 // Fade in and out to create a flash effect
13 FlashHighlightingAnimator.prototype = {
14
15     // Start animation
16     animate: function(callback) {
17         var self = this;
18
19         // Fade in Highlighting
20         var fadeIn = new ArrayFadeInAnimator(self.highlightDivs,
21             self.fadeInDuration);
21         flashIn.animate(function() {
22
23             // Fade out Highlighting
24             var fadeOut = new ArrayFadeOutAnimator(self.highlightDivs,
25                 self.fadeOutDuration);
25             flashOut.animate(function() {
26
27                 // Animation finished: Call back
28                 callback();
29             });
30         });
31     }
32 };
```

Beim Start der Animation des *FlashHighlightAnimators*, wird zuerst ein Fade-In Animator erstellt und ausgeführt. Als Übergabe an die *animate* Funktion wird eine weitere Funktion erstellt, die einen Fade-out Animator erstellt und startet. Ist dieser ebenfalls vollendet wird die Callback Funktion des *Flash-*

*HighlightAnimators* ausgeführt, da die gesamte Animation beendet ist.

Animatoren, die sich nicht aus anderen Animatoren zusammensetzen, benutzen entweder die Javascript Bibliothek „Web Animations“<sup>9</sup> oder eine manuelle Veränderung der gewählten Elemente innerhalb einer rekursiven *setTimeout* Funktion.

Web Animations werden ähnlich wie Animatoren erstellt. Auf dem zu animierenden Element kann eine *animate* Funktion aufgerufen werden, welche als Übergabe das zu animierende Attribut, die Dauer und die Anzahl der Wiederholung bekommt. Davon zurückgegeben wird ein Objekt, welches mit einem Event Listener erweitert werden kann. In diesem kann wieder die Callback Funktion aufgerufen werden (siehe Listing 6.7).

**Listing 6.7:** Beispiel für eine Web Animation Funktion

```
1 ArrayFadeInAnimator.prototype = {
2
3     animate: function(callback) {
4
5         var fade = this.element.animate([
6             {opacity: 0},
7             {opacity: 1}
8         ], {
9             direction: "alternate", duration: this.duration, iterations:
10                1
11        });
12
13        fade.addEventListener('finish', function() {
14            callback();
15        });
16    }
17};
```

Am Beispiel des Array Highlight Animators kann gut das manuelle Setzen von TimeOuts und Transformationen erklärt werden. Der Animator benutzt ein farbiges, transparentes Div um ein Wort Zeichen für Zeichen zu markieren. Zu Beginn hat das Div eine Größe von 0 und ist am Anfang des Wortes positioniert. Die gegebene Dauer für die Animation wird durch die Anzahl der Zeichen des Wortes geteilt um eine konstante Zeit für jedes Zeichen zu bekommen. Diese *CharDur* wird mit dem Index des aktuellen Zeichens und der Callback Funktion an eine rekursive Funktion übergeben (siehe Listing 6.8). Ist der Index größer als die Länge des Wortes, ist das ganze Wort markiert und die Animation vollendet, also kann die Callback Funktion ausgeführt werden. Falls das nicht der Fall ist, werden alle Zeichen des Wortes bis zu dem aktuellen Index einem lokalen String hinzugefügt, um die Länge der aktuellen Markierung zu messen. Das *Highlight-Div* wird dann auf die gemessene Breite gesetzt. Die *setTimeout* Funktion bekommt als Übergabeparameter eine Funk-

---

<sup>9</sup><https://github.com/web-animations/web-animations-js>

tion und eine Zeit, die gewartet wird, bis diese Funktion ausgeführt wird. Die Zeit wurde vorberechnet und die Funktion ist die Rekursion mit einem um 1 erhöhten Index.

**Listing 6.8:** Beispiel für eine TimeOut Animation Funktion

```
1 | highlightCharacter: function(charDur, index, callback){  
2 |     var text = this.element.innerText;  
3 |     if (index >= text.length) {  
4 |         callback(); // finished word  
5 |         return;  
6 |     }  
7 |  
8 |     // Add every character until index to the highlighting  
9 |     var highlightText = '';  
10 |    for (var i = 0; i != index+1; i++)  
11 |        highlightText += text[i];  
12 |  
13 |    // Set highlight div to current text width  
14 |    this.highlightDiv.style.width = textWidth(highlightText) + 'px';  
15 |  
16 |    // Wait and jump to next character  
17 |    var self = this;  
18 |    setTimeout(function() {  
19 |        self.highlightCharacter(charDur, index+1, callback);  
20 |    }, charDur);  
21 |}
```

# Kapitel 7

## Zusammenfassung und Ausblick

In dieser Arbeit wurden, basierend auf bereits bekannten Arbeiten und eigenen Ideen, verschiedene Techniken zur Visualisierung von Textvergleichen entworfen und implementiert. Um die Visualisierungen ausführlich testen zu können, wurde ein Framework erstellt, welches vier Korpora bereitstellt, von denen zwei revisionierte Texte enthalten und zwei Texte, die eine Quelle-Plagiat Beziehung haben. Mit Polymer wurden komponentenbasierte Visualisierungen erstellt, die sehr leicht in das Framework und jede andere Webumgebung eingebunden werden können. Das Framework ermöglicht außerdem mit einem Parameter-Widget die Anpassung der Parameter aller Visualisierungen, um die optimale Darstellung für einen bestimmten Textvergleich zu finden.

Zuerst wurden die bereits bekannten Visualisierungen „Classic Diff“, mit Fokus auf den Unterschieden beider Texte, und „Vertical Aligned“, welches den Fokus auf die Gemeinsamkeiten legt, erstellt. Die „Two Lines Diff“ Visualisierung bietet eine gute statische Darstellung für den Vergleich zweier Texte, bei der sowohl Gemeinsamkeiten als auch Unterschiede deutlich zu erkennen sind. Da die Unterschiede direkt übereinander stehen, ist der Vergleich hier einfach.

Um ein dynamischeres Konzept zu testen, wurde mit Animationen versucht, das Kopieren und Bearbeiten eines Autors zu imitieren. Diese Visualisierung bietet zum Beispiel die Möglichkeit, eine „Classic Diff“ Visualisierung animiert zu erstellen, indem gelöschte und hinzugefügte Wörter markiert werden. Für den Vergleich mehrerer Revisionen stellte sich jedoch heraus, dass eine statische Übersicht sinnvoller ist. Dazu wurde der „Multi Revision Browser“ erstellt, bei dem der Fokus auf den Gemeinsamkeiten liegt. Alle Revisionen werden anhand ihrer Gemeinsamkeiten ausgerichtet. So können alle Revisionen übersichtlich miteinander verglichen werden, gleichzeitig bleibt jede Revision als zusammenhängender Text bestehen. Außerdem wurden hier auch Interaktionen implementiert, um Textabschnitte im Detail zu vergleichen. Hier wird der Vorteil generischer Polymer-Elemente deutlich, weil beim Vergleich von zwei

## KAPITEL 7. ZUSAMMENFASSUNG UND AUSBLICK

---

Textabschnitten verschiedener Revisionen eine Quelle-Plagiat Visualisierung eingefügt werden kann.

Weiterhin wurde eine Nutzerstudie durchgeführt, bei der 34 Schüler eigene Visualisierungen zu Textvergleichen erstellen sollten. Die Ergebnisse zeigten, dass es keine natürliche Lösung für das gegebene Problem gibt. Die Entwürfe reichten von einfachen Markierungen im Text bis zu abstrakten Darstellungen.

Um die Tauglichkeit der entwickelten Visualisierung zu testen, wäre für die Zukunft eine Nutzerstudie sinnvoll. Allerdings ist es schwierig, ein objektives Maß zu finden, anhand dessen die Visualisierungen bewertet werden können. Eine sinnvolle Erweiterung der Textvergleiche wäre die Einbeziehung von linguistischen Informationen. Durch die Tokenisierung mit UIMA stehen bereits Wortart und Wortstamm des Tokens zur Verfügung. Eine sinnvolle Einbringung in den Vergleich und die anschließende Darstellung könnte vorteilhaft sein. Außerdem kann die Verwendung der Levenshtein-Distanz zur Bestimmung von abgeänderten Wörtern noch verbessert werden. Zurzeit ist diese besonders bei kurzen Wörtern fehleranfällig. Zu Beginn sollten verschiedene Differenzalgorithmen getestet und miteinander verglichen werden. Es stellte sich jedoch schnell heraus, dass die wenigen brauchbaren Differenzalgorithmen in Javascript sich fast nur durch ihre Tokenisierung unterschieden. Diese wurde dann mit UIMA durchgeführt. Für die Zukunft wäre es sinnvoll, eigene Differenzalgorithmen zu entwickeln und für Textvergleiche auf Tokenbasis zu optimieren.

Die Visualisierungen bieten auch noch Raum für Verbesserungen. Der „Multi Revision Browser“ beispielsweise ist recht ineffizient, umso mehr Revisionen verglichen werden. Animationsvisualisierungen müssen meist stark an den gegebenen Text angepasst werden. Eine feinere Berechnung der Animationsgeschwindigkeiten je nach Textlänge und Unterschieden wäre sinnvoll.

Weiterhin gibt es noch Visualisierungen und Konzepte, die noch nicht implementiert wurden. Das Framework ist leicht um neue Visualisierungen erweiterbar und kann so letztendlich zur Sammlung und Erforschung aller Visualisierungen für Textvergleiche beitragen.

# Literaturverzeichnis

Fintan Culwin and Thomas Lancaster. Visualising intra-corporeal plagiarism. In *International Conference on Information Visualisation, IV 2001, London, England, UK, July 25-27, 2001*, pages 289–296, 2001. doi: 10.1109/IV.2001.942072. URL <http://dx.doi.org/10.1109/IV.2001.942072>. 2.3.3, 2.10, 7

Manuel Freire. Visualizing program similarity in the ac plagiarism detection system. In *Proceedings of the working conference on Advanced Visual Interfaces, AVI 2008, Napoli, Italy, May 28-30, 2008*, pages 404–407, 2008. doi: 10.1145/1385569.1385644. URL <http://doi.acm.org/10.1145/1385569.1385644>. 2.5, 2.18, 7

Bela Gipp, Norman Meuschke, and Corinna Breitinger. Citation-based plagiarism detection: Practicability on a large-scale scientific corpus. *JASIST*, 65(8):1527–1540, 2014. doi: 10.1002/asi.23228. URL <http://dx.doi.org/10.1002/asi.23228>. 2.3.1, 2.6, 7

James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Commun. ACM*, 20(5):350–353, 1977. doi: 10.1145/359581.359603. URL <http://doi.acm.org/10.1145/359581.359603>. 2.1

Stefan Jänicke, Annette Gefner, Marco Büchler, and Gerik Scheuermann. Visualizations for text re-use. In *Proceedings of the 5th International Conference on Information Visualization Theory and Applications, IVAPP 2014, Lisbon, Portugal, 5-8 January, 2014.*, pages 59–70, 2014. doi: 10.5220/0004692500590070. URL <http://dx.doi.org/10.5220/0004692500590070>. 2.3.3, 2.11, 2.12, 7

Carlos Monroy, Rajiv Kochumman, Richard Furuta, and Eduardo Urbina. Interactive timeline viewer (itlv): A tool to visualize variants among documents. In *Visual Interfaces to Digital Libraries [JCDL 2002 Workshop]*, pages 39–49, 2002. doi: 10.1007/3-540-36222-3\_4. URL [http://dx.doi.org/10.1007/3-540-36222-3\\_4](http://dx.doi.org/10.1007/3-540-36222-3_4). 2.3.4, 2.14, 7

## LITERATURVERZEICHNIS

---

Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi: 10.1007/BF01840446. URL <http://dx.doi.org/10.1007/BF01840446>. 2.1, 2.1, 7

Randy L. Ribler and Marc Abrams. Using visualization to detect plagiarism in computer science classes. In *IEEE Symposium on Information Visualization 2000 (INFOVIS'00), Salt Lake City, Utah, USA, October 9-10, 2000.*, pages 173–178, 2000. 2.3.4, 2.13, 7

Patrick Riehmann, Henning Gruendl, Martin Potthast, Martin Trenkmann, Benno Stein, and Bernd Froehlich. WORDGRAPH: keyword-in-context visualization for netspeak’s wildcard search. *IEEE Trans. Vis. Comput. Graph.*, 18(9):1411–1423, 2012. doi: 10.1109/TVCG.2012.96. URL <http://doi.ieee.org/10.1109/TVCG.2012.96>. 2.3.2, 2.8, 7

Patrick Riehmann, Martin Potthast, Benno Stein, and Bernd Fröhlich. Visual assessment of alleged plagiarism cases. *Comput. Graph. Forum*, 34(3):61–70, 2015. doi: 10.1111/cgf.12618. URL <http://dx.doi.org/10.1111/cgf.12618>. 2.3.2, 2.7, 2.9, 4.2, 7

Ross Shannon, Aaron J. Quigley, and Paddy Nixon. Deep diffs: visually exploring the history of a document. In *Proceedings of the International Conference on Advanced Visual Interfaces, AVI 2010, Roma, Italy, May 26-28, 2010*, pages 361–364, 2010. doi: 10.1145/1842993.1843063. URL <http://doi.acm.org/10.1145/1842993.1843063>. 2.4, 2.17, 7

Fernanda B. Viégas, Martin Wattenberg, and Kushal Dave. Studying cooperation and conflict between authors with *history flow* visualizations. In *Proceedings of the 2004 Conference on Human Factors in Computing Systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004*, pages 575–582, 2004. doi: 10.1145/985692.985765. URL <http://doi.acm.org/10.1145/985692.985765>. 2.16, 2.4, 7

# Abbildungsverzeichnis

2.1	„Edit graph“ aus der Arbeit von Myers [1986] . . . . .	5
2.2	Diffuse Screenshot: <a href="http://diffuse.sourceforge.net/screenshot.png">http://diffuse.sourceforge.net/screenshot.png</a>	7
2.3	Github Screenshot . . . . .	7
2.4	similarity texter von Weber-Wulff . . . . .	8
2.5	Diffchecker . . . . .	8
2.6	CitePlag von Gipp et al. [2014] . . . . .	9
2.7	PlagVis von Riehmann et al. [2015] . . . . .	10
2.8	„Wordgraph“ von Riehmann et al. [2012] . . . . .	11
2.9	„Diff Blending“ von Riehmann et al. [2015] . . . . .	11
2.10	Matrixvisualisierung von Culwin and Lancaster [2001] . . . . .	12
2.11	„Text-Reuse-Grid“ von Jänicke et al. [2014] . . . . .	13
2.12	„Dot-Plot-View“ von Jänicke et al. [2014] . . . . .	13
2.13	„Categorical Patterngram“ von Ribler and Abrams [2000] . . . . .	14
2.14	Timeline-Visualisierung von Monroy et al. [2002] . . . . .	15
2.15	Barcodevisualisierung von GutenPlag . . . . .	15
2.16	„History Flow“ von Viégas et al. [2004] . . . . .	16
2.17	„Deep Diff“ von Shannon et al. [2010] . . . . .	17
2.18	Graphvisualisierungen von Freire [2008] . . . . .	18
2.19	Visualisierung aus dem Projekt „Visual Text Analysis“ . . . . .	19
3.1	Erstes Beispiel für eine Visualisierung mit Referenzdokument . . . . .	21
3.2	Zweites Beispiel für eine Visualisierung mit Referenzdokument . . . . .	22
3.3	Beispiel für eine gleichberechtigte Darstellung aller Revisionen . . . . .	22
3.4	Eine abstrakte Graphenvisualisierung . . . . .	23
3.5	Eine abstrakte Koordinatensystemvisualisierung . . . . .	23
3.6	Beispiel für eine Fragmentierung in Textabschnitte und Darstellung in einer neuen Ansicht . . . . .	24
4.1	Side By Side Beispiel . . . . .	27
4.2	Vertical-Aligned Beispiel . . . . .	27
4.3	Classic Diff Beispiel . . . . .	28
4.4	Two Lines Diff Beispiel . . . . .	29

## *ABBILDUNGSVERZEICHNIS*

---

4.5	Copy & Paste Animation . . . . .	32
4.6	Multi Revision Browser Beispiel . . . . .	36
4.7	Beispiel für das manuelle Vergleichen innerhalb des Multi Revision Browsers . . . . .	37
6.1	Framework Startbildschirm . . . . .	43
6.2	Beispiel für ein Token-Objekt in JSON . . . . .	45
6.3	Beispiel URL . . . . .	46
6.4	Beispiel für ein dat.gui Parameter-Widget . . . . .	47
6.5	Horizontales Scrollen mit Canvas . . . . .	49

# Listings

6.1	Weiterleitung des <i>Resize-Events</i> zum Framework . . . . .	40
6.2	Template für Polymer-Elemente . . . . .	41
6.3	Bootstrap-Dropdown-Menü für die Auswahl des Korpus . . . . .	43
6.4	Daten einlesen mit jQuery . . . . .	44
6.5	Beispiel für einen Span der in den ersten beiden Vergleich vor- kommt . . . . .	50
6.6	Beispiel für Callback-Funktion der Animatoren . . . . .	54
6.7	Beispiel für eine Web Animation Funktion . . . . .	55
6.8	Beispiel für eine TimeOut Animation Funktion . . . . .	56