Large-scale Publikationsanalyse und Geodaten-Visualisierung

Teilnehmer:
- Richard Khulusi
- Daniel Alexander

Verantwortlicher:
- Dr. Anika Groß [link]

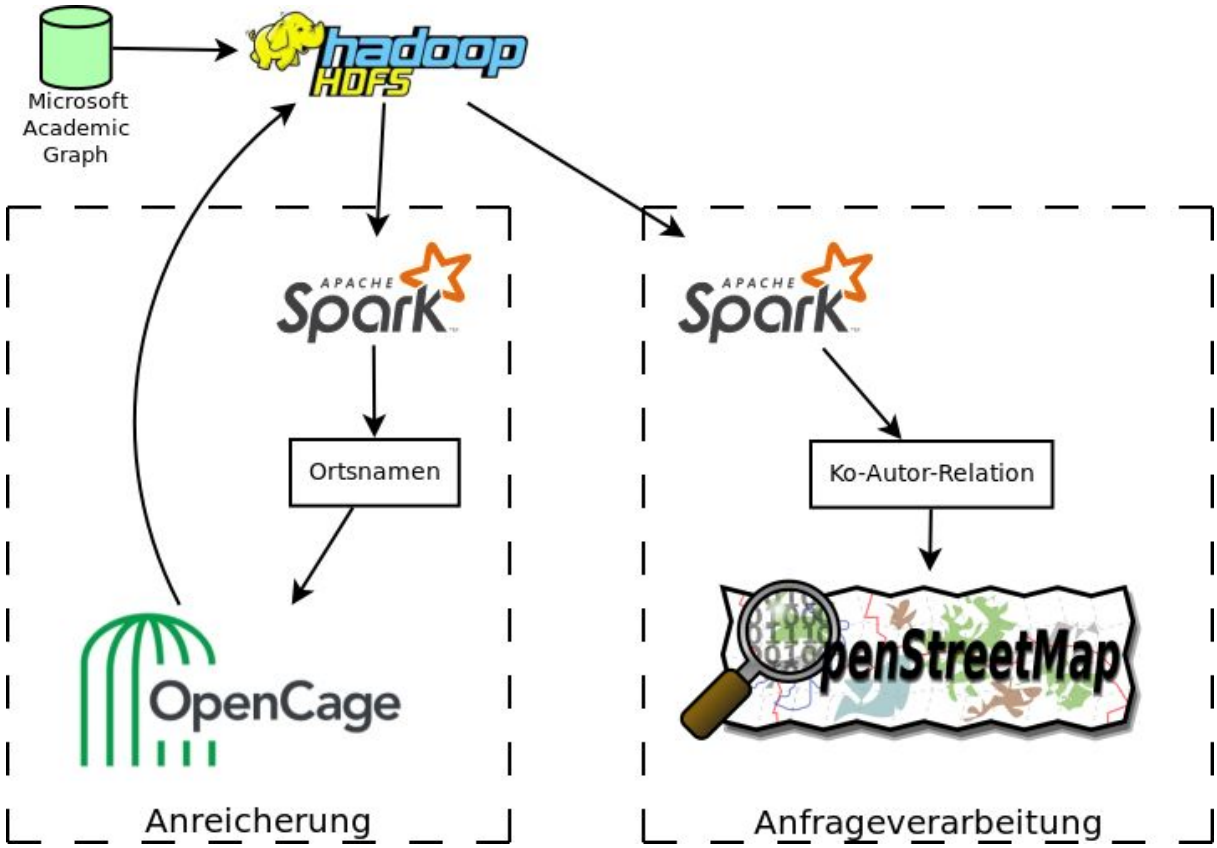
Repository:
- Github [link]

Aufgabenstellung:

Aufgabe ist es, die Beziehungen zwischen Arbeitsgruppen in verschiedenen Orten zu analysieren und zu visualisieren. Der Fokus liegt dabei auf der Analyse von Ko-Autor-Beziehungen. Wir sollen beispielsweise herausfinden, mit welchen anderen Hochschulen die Datenbank-Abteilung der Uni Leipzig bei der Erstellung ihrer Publikationen zusammengearbeitet hat. Als Datenquelle dient der Microsoft Academic Graph (MAG). Zur Visualisierung der Ergebnisse soll eine Landkarte verwendet werden, auf der die Ko-Autor-Beziehungen als Verbindungslinien zwischen verschiedenen Orten dargestellt werden.

Weiterführende Aufgabenstellungen:
Je nach Fortschreiten des Projekts können weitere Fragestellungen behandelt werden. Diese könnten sich mit der Visualisierung von Standorten befassen, deren Publikationen häufig zitiert werden oder von Zusammenhängen zwischen Orten und Forschungsgebieten (FieldOfStudy). Damit könnten wichtige Zentren für die Erforschung bestimmter Themengebiete ausfindig gemacht werden.

Workflow:



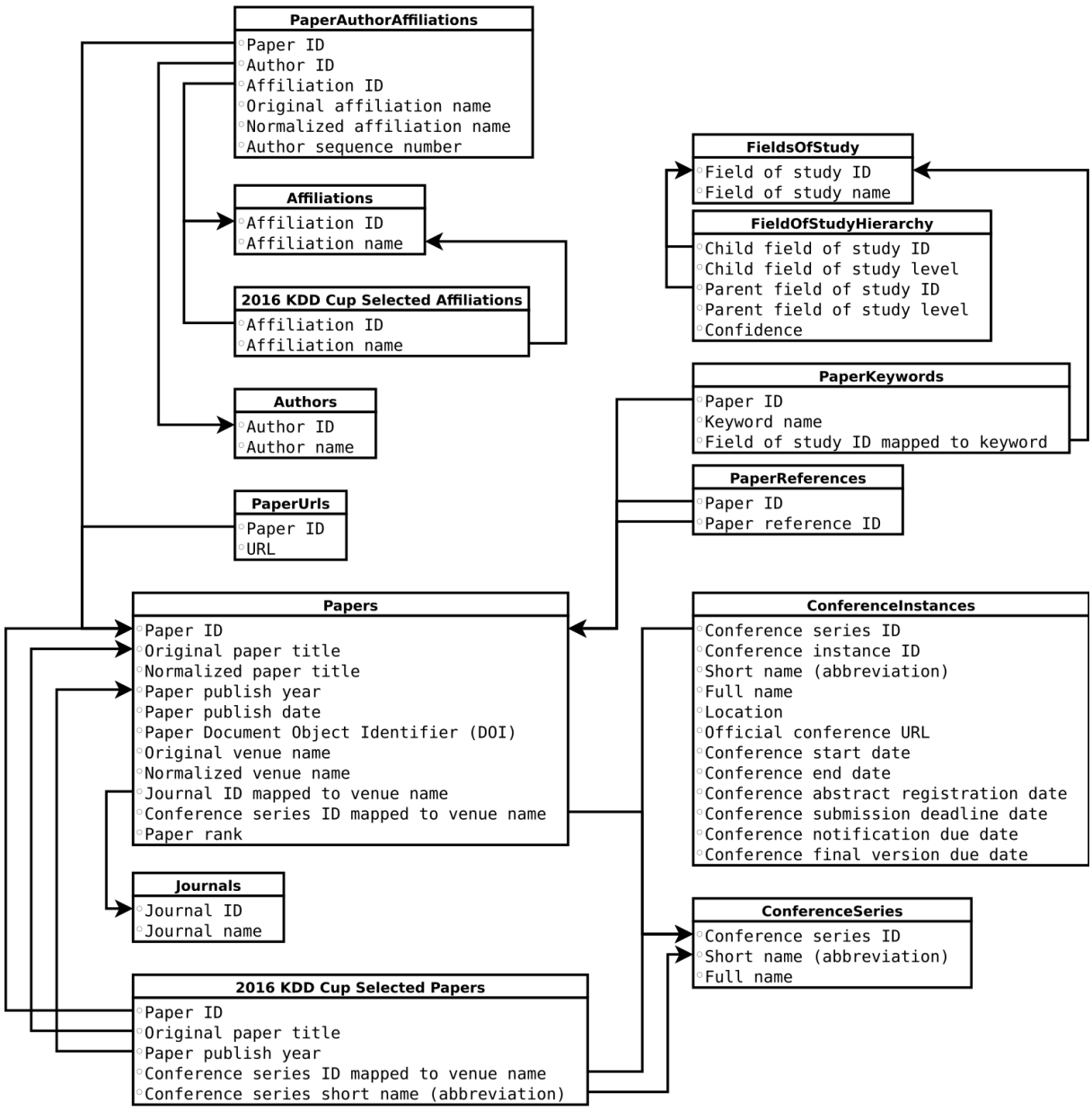
Software:

- Apache Spark [link]
  - Verarbeitung der Daten im Cluster
- Apache Hadoop [link]
  - Datenspeicherung für Cluster mittels HDFS
- OpenStreetMap [link]
  - Visualisierung von Geodaten

Datenimport:

Die Publikationsdaten entstammen dem Microsoft Academic Graph [link]. Es handelt sich dabei nach Angaben der Verantwortlichen um einen heterogenen Graph, der Informationen zu wissenschaftlichen Publikationen beinhaltet. Das betrifft Zitier-Beziehungen zwischenden Publikationen sowie Informationen zu Autoren, Institutionen, Journals und wenigen Konferenzen. Die Daten werden als gezippte Textdateien (.TXT) zur Verfügung gestellt und sind über HTTP verfügbar.

Datenbankschema:



Umfang der Daten: (ein vollständiger Eintrag pro Zeile)

Datei(en)	**Größe**
gezipptes Paket	29 GB
entpackte Textdateien	98 GB

Datei	**Zeilen**
PaperUrls.txt	454.070.767
PaperAuthorAffiliations.txt	337.000.600
FieldOfStudyHierarchy.txt	182.103
Conferences.txt	1.283
2016KDDCupSelectedAffiliations.txt	741
Papers.txt	126.909.021
FieldsOfStudy.txt	53.834
ConferenceInstances.txt	50.202
Journals.txt	23.404
Authors.txt	114.698.044
license.txt	85
readme.txt	90
PaperKeyWords.txt	158.280.968
2016KDDCupSelectedPapers.txt	3.677
Affiliations.txt	19.843

Vorverarbeitung:

Zur Beantwortung der obigen Fragestellung ziehen wir die Tabelle PaperAuthorAffiliations heran. Diese enthält Aussagen über Autoren sowie Affiliations zu Publikationen. Aus dieser Tabelle wollen wir die Affiliations entnehmen, um damit Ortsinformationen zu erhalten (siehe Anreicherung). Weiterhin wollen wir mithilfe dieser Tabelle Ko-Autorschaften identifizieren.

Selektion: Welche Datensätze brauchen wir überhaupt?
Vermutlich können wir alle nötigen Informationen aus der PaperAuthorAffiliations-Relation erhalten.

Projektion: Welche Informationen davon brauchen wir?
Gewisse Informationen der PaperAuthorAffiliations werden nicht benötigt. So existieren beispielsweise für jeden Eintrag zwei Affiliations-Einträge. Einmal der Originaleintrag (in der Form, wie er aus dem Paper extrahiert wurde) und einmal eine überarbeitete und normalisierte Version. Letztere versucht eine Standardschreibweise für die Strings zu erzeugen, welche sich für eine automatische Weiterverarbeitung, wie es bei uns auch geplant ist (siehe Anreicherung), besser eignen sollte. Neben diesen normalisierten Affiliations werden auch die PaperID, AuthorID und AffiliationsID gebraucht. Noch offen ist die Frage nach der Author Sequenz Number. Vermutlich enthält diese Informationen zu den Ko-Authorbeziehungen. Somit wäre diese Information ebenfalls relevant.

Reduktion, gruppieren und verknüpfen:
Um den Umfang des Datensatzes zu minimieren und Bearbeitungszeiten damit zu verringern, werden vor der eigentlichen Verarbeitung nur die relevanten Daten extrahiert. Die Punkte Selektion und Projektion (s.o.) erklären bereits, welche Informationen ausgewählt werden (müssen). Die übrigen Daten entfallen in der Betrachtung, was die zu verarbeitende Datenmenge reduziert. Darüber hinaus können weitere Datenvolumen eingespart werden, in dem mittels Gruppierungen und DISTINCT Anweisungen doppelte Einträge, sofern nicht nützlich, entfernt werden, beziehungsweise zu einem Tupel zusammen gefasst werden. Solch ein Vorgehen wird besonders für den Punkt Anreicherung eine Rolle spielen, wo auf Grund von Dienstlimitierungen möglichst mit wenigen Affiliations gearbeitet werden sollte, ohne qualitative Verluste zu erleiden.

Anreicherung:

Nachdem der Datensatz auf den für uns relevanten Teil reduziert wurde und durch die Vorverarbeitung auch die Informationen in verarbeitbarer Form vorliegen, fehlt uns noch eine Information für die Analyse. Da wir Ko-Authorbeziehungen auf einer Karte darstellen wollen, werden Georeferenzen benötigt, welche im MAG nicht enthalten sind.

Wir werden in diesem Punkt also Informationen in den Datensatz einpflegen. Dafür wird eine neue Relation AffiliationsGeoreferences erzeugt. In dieser wird zu jeder Affiliation ein eindeutiger Ort in Form von Longituden und Latituden gemappt. Zudem werden Informationen über Land und Stadt abgespeichert, um später detaillierte Anfragen bezüglich einzelner Orte formulieren zu können. Mittels Selektionsanfragen anhand der Orte können so bequem Einschränkungen getroffen werden. Neben der Beantwortung gezielter Anfragen können wir so falls nötig auf Performanzprobleme reagieren.
Um an diese Informationen zu gelangen, nutzen wir einen GeoCoding Dienst namens OpenCageData [link]. Dieser basiert auf der OpenStreetMap Datenbank und bekommt den normalisierten Affiliationsstring übergeben. Zu diesem String sucht er selbstständig das passendste Ergebnis (auf die Art und Weise sowie die Korrektheit des Rankings haben wir keinen Einfluss*). Dieses wird uns nun in Form eines JSON-Objektes zurückgegeben, woraus wir uns die nötigen Informationen parsen und in der Relationen einfügen (Materialisierung).
Dieser Vorgang wird separat in einem Java-Programm ablaufen. Er stellt also einen einmaligen Arbeitsschritt im Workflow dar, da sich unsere Daten nicht ändern. Anzumerken sei dabei noch, dass es kritisch ist, dass wir nicht unnötig große Datenmengen in diesem Schritt bearbeiten, da der Dienst gewisse Limitierungen hat, welche die Bearbeitungszeit drastisch erhöhen könnte. So lässt sich nicht mehr als eine Anfrage pro Sekunde und nicht mehr als 2.500 Anfragen am Tag abfragen. Bei einem Umfang von 5.000 Affiliations wären dies schon zwei Tage für den Prozess der Anreicherung.

*) Sollte es zu Problemen mit der Korrektheit kommen, so wäre ein alternativer Weg, dass mittels OpenCageData nur Informationen über Land/Stadt extrahiert werden und der letzte Schritt zur Bestimmung der exakten GeoKoordinaten direkt in der Visualisierung mittels OpenLayers' interner Funktion ermittelt werden.

Analyse der Daten:

Die Verarbeitung der Daten soll mittels Spark auf einem HDFS (Hadoop Distributed Files System) erfolgen. Dazu wurde uns von der Universität Leipzig ein PC (Remote-PC) zur Verfügung gestellt. Wir haben ein HDFS im Pseudo Distributed Mode aufgesetzt und nach erfolgtem Download die Daten ins HDFS geladen. Der Zugriff auf Spark / HDFS erfolgt in Java, wir verwenden also die Spark Java API. Weiterhin bietet Spark mit SparkSQL [link] auch ein Framework, welches es ermöglicht auf geladenen Daten SQL-artig zu arbeiten, welches sich für die Analyse hier anbietet.
Nach einem ersten Plan sollte der Spark-Client des entfernten Uni-PCs vom privaten Arbeitsgerät (Lokal-PC) aus angesteuert werden. Dieser Zugriff erwies sich bereits zu Beginn als problematisch, sodass wir vorerst unsere Programme auf Lokal-PCs mit einem ähnlichen Setup aber reduzierten Daten testen und anschließend das

Programm als JAR-verpackt auf den Remote-PC transferieren und dort auf dem gesamten Datensatz ausführen.
Zur Verarbeitung der Daten mittels Spark werden die TXT-Dateien zeilenweise als n-Tupel entsprechend der jeweiligen Datei verpackt (Referenz: link). Die so aufbereiteten Daten können dann auf ihren relevanten Anteil reduziert und zur Beantwortung von Queries analog zu SQL verwendet werden.
Das Ergebnis soll Aussagen darüber liefern, welche Institutionen miteinander kooperieren, dafür werden entsprechende Kooperationen gezählt und für den Visualisierungsschritt weitergereicht. Je nachdem, was die Daten tatsächlich an Details preisgeben, wird eine unterschiedliche Auflösung in den Ergebnissen angestrebt. So steht bislang nicht fest, ob verschiedene Abteilungen einer einzigen Hochschule einzeln betrachtet werden, oder ob diese zu einem Knoten zusammengefasst werden.

Visualisierung der Ergebnisse:

Die Resultate sollen auf einer Landkarte dargestellt werden. Dabei werden Orte Knoten sein und Ko-Autor-Beziehungen werden zu Kanten zwischen diesen Knoten.
Als Software zur Visualisierung soll die Open-Source API OpenLayers verwendet werden. Diese bietet das Erstellen einer Karte via JavaScript an, wie sie auch bei beispielsweise OpenStreetMap zu sehen ist (Referenz: [link]).
Wie im Punkt Anreicherung beschrieben, befinden sich zu diesem Zeitpunkt alle nötigen Daten in Relationen und so sollte es genügen diese mittels der Query in Spark zu laden und an eine Implementation von OpenLayers zu reichen.
In dieser ist zu klären, wie die Knoten angezeigt werden können, was primär vom genauen Umfang der Daten abhängt. So kann es bei zu vielen Daten zu Überdeckungsproblemen kommen, welche es nötig machen, entweder die dargestellten Daten in ihrer Anzahl weiter zu reduzieren oder durch Zusammenfassen die Menge an Daten auf der Karte zu reduzieren (beispielsweise durch einen Glyphansatz).
Weiterhin soll durch die Implementierung der Kanten die Darstellung von möglichst vielen Ko-Autor-Beziehungen gleichzeitig möglich sein, ohne die Darstellung zu überladen. Erste Ideen dazu würden Ansätze beinhalten, die Kanten nur nach Selektion von Knoten anzeigt, was mit einem Glyphenansatz zu bewältigen wäre.
Durch die Nutzung der OpenLayers API werden wir die resultierende Karte in Javascript im WebBrowser anzeigen lassen.

Erkenntnisse über die Daten:

PaperAuthorAffiliations:

- Paper tauchen mehrfach auf.
- Pro Eintrag gibt es einen Autor
- Pro Eintrag mit gleicher PaperID gibt es eine andere AuthorSequenceNumber
- Alle Autorschaftsbeziehungen sind somit per Gruppierung auf den PaperIDs zu erhalten.
- Ob der Autor mit der AuthorSequenceNumber = 1 auch der Erstautor ist, bleibt zu klären.

Unreinheiten der Daten

- PaperAuthorAffiliations:
    - Nicht jeder Eintrag hat Informationen zu Affiliations
    - Alle Einträge ohne normalizedAffiliationsString haben keine AffiliationsID
    - Manche Einträge tragen keine normalizedAffiliationStrings, obwohl originalAffiliationStrings verfügbar sind. Diese Einträge haben auch keine AffiliationsID
- Normalisierte Strings können Tag-Bausteine enthalten (welche nicht wohlgeformt sind und somit das XML-Format beschädigen). Als Beispiel hier zwei normalisierte Strings:
    - Advanced Concepts Team, ESA, ESTEC Keplerlaan 1, Postbus 299, <ce:hsp>2200 AG, Noordwijk, The Netherlands
    - BRICS, <ce:cross-ref><ce:sup>3</ce:sup></ce:cross-ref><ce:footnote><ce:label>3</ce:label><ce:note-para>Basic Research in Computer Science, a center of the Danish National Research Foundation.</ce:note-para></ce:footnote> Department of Computer Science, University of Aarhus, Ny Munkegade bldg. 540, 8000 Aarhus C, Denmark

Umsetzung:

Zum Anzeigen der Daten auf der Karte wird OpenLayers und ein Glyphenansatz gewählt. Es folgt eine Beschreibung des Workflows zum generieren/extrahieren aller dafür benötigten Daten.

Die Daten werden durch eine Pipeline aus SparkSQL-Anfragen, Java Datentransformationen und JavaScript Visualisierungen bearbeitet.

Für die Verwendung von SparkSQL haben wir Java-Klassen erstellt, die die einzelnen Entitäten der verschiedenen Tabellen abbilden können. Mit deren Hilfe ist ein Mapping der Tabellen auf Java-Klassen und damit wiederum das Anfragen der Tabellen mit SparkSQL möglich.

Die folgenden SQL-Anfragen wurden mit SparkSQL umgesetzt:

Query: “duplikatfreie PaperAuthorAffiliation”
Welche Affiliations haben an welchem Paper mitgearbeitet? (jede Affiliation soll nur 1malig gezählt werden)

```
SELECT
        paperID,
        FIRST(authorID),
        affiliationID,
        FIRST(originalAffiliationName),
        FIRST(normalizedAffiliationName),
        FIRST(authorSequenceNumber)
FROM PaperAuthorAffiliation
GROUP BY paperID, affiliationID
```

-> erzeuge eine duplikatfreie PaperAuthorAffiliation
-> Liste aller PaperAuthorAffiliation-Einträge passt nicht in den RAM, daher ist diese Query für uns nicht ausführbar

Query: “wichtigste Affiliations”
Welches sind die 1000 “wichtigsten” Affiliations und wie wichtig ist jede von ihnen? Die Gewichtung einer Affiliation ermittelt sich dabei aus der Anzahl an Papern, bei der die Affiliation mitgewirkt hat

```
SELECT
        COUNT(affiliationID) as Anzahl,
        affiliationID as affiliationID,
```

```
        FIRST(normalizedAffiliationName) as Name,
        FIRST(originalAffiliationName) as Fullname
FROM
        PaperAuthorAffiliation
WHERE
        NOT (normalizedAffiliationName = '')
GROUP BY
        affiliationID
ORDER BY
        Anzahl DESC
LIMIT 1000
```

-> Duplikatfreiheit ist hier wichtig, da die COUNT-Aggregation einige Affiliations für einige Paper mehrfach wertet.
-> Beispiel: Ein Paper entsteht als Zusammenarbeit der Uni Leipzig und der HTWK Leipzig. Autoren: Autor1 (Uni Leipzig), Autor2 (Uni Leipzig), Autor3 (HTWK). Würde die Tabelle PaperAuthorAffiliations nur Informationen zu diesem einen Paper enthalten, so sähe das Ergebnis der Query wie folgt aus.

Anzahl	affiliationID	Name	Fullname
2	1	leipzig university	Universität Leipzig
1	2	htwk	HTWK Leipzig

Gewünscht ist allerdings das folgende Resultat:

Anzahl	affiliationID	Name	Fullname
1	1	leipzig university	Universität Leipzig
1	2	htwk	HTWK Leipzig

Query: “Ermittle Ko-Autorschaften”
Welche Affiliation hat mit welcher anderen Affiliation zusammengearbeitet?

```
SELECT
        A.affiliationID AS affID_A,
        B.affiliationID AS affID_B,
        COUNT(A.affiliationID, B.affiliationID) AS anzahl
FROM
        View_pID_affID_affName A JOIN
        View_pID_affID_affName B
        ON A.paperID = B.paperID
WHERE NOT(A.affiliationID = B.affiliationID)
```

GROUP BY A.affiliationID, B.affiliationID

Geocoding

Nachdem die Daten nun als materialisierte Anfrageergebnisse vorliegen beginnt die Transformation in Java.
Hier beginnt der Prozess mit dem Geocoding (Geocoding.java).

Geocoding beschreibt im Allgemeinen den Prozess, ein Mapping zwischen von Adressen oder Beschreibungen von Adressen auf Koordinaten zu erstellen.
Vor dem eigentlichen Geocoding erfolgt noch ein Preprocessing-Schritt. Dabei lesen wir die Datei affiliations_top_1000.txt ein, die die materialisierten Anfrageergebnisse der Query “wichtigste Affiliations” enthält, und erzeugen daraus dann ein XML-File nach KML-Standard (Erklärung weiter unten).
Dabei werden allerdings die “<coordinates>”-Tags zunächst leer gelassen. Diese Information wird dann im eigentlichen Geocoding erst ermittelt.
Der Dienst, der für das Geocoding verwendet wird ist OpenCageData (siehe oben). Wir stellen zu jedem Ort eine Anfrage an den Dienst und extrahieren Longitude und Latitude aus dessen Antworten. Sofern entsprechende Daten extrahiert werden konnten, werden sie in das XML-File geschrieben, andernfalls werden die Koordinaten [0.0, 0.0] (Koordinaten von “Null Island”, einem Debug-Ort) als Antwort des Dienstes angesehen.
Aufgrund der Zugriffslimitierungen von OpenCageData haben wir die Schritte des “Erstellen des XML-Files” und “Ausführen des Geocoding” logisch von einander getrennt, so dass der Prozess über verschiedene Tage neugestartet werden kann. Dafür wird für das Geocoding über das XML iteriert und die normalizedAffiliationStrings (Im “Id”-Attribut) extrahiert (sofern dies kein Ergebnis gibt wird der originalAffiliationString aus dem “name”-Tag verwendet). Mithilfe eines Offsets (in der “tmp_offset_txt”) der separat abgespeichert liegt, kann der Prozess nachträglich einfach weiter fortgesetzt werden.

Das Resultat des Geocoding ist damit eine XML-Datei, welche der Karte übergeben werden kann, um dort die Glyphen der Affiliations anzuzeigen.

Kanten

Zum Darstellen der Ko-Authorschaft-Beziehungen werden die Affiliations auf der Karte mittels Kanten verbunden. Eine Kante gibt dabei an, dass die verbundenen Affiliations an mindestens einem Paper zusammengearbeitet haben.

Für die Kanten werden ol.geom.MultiLineStrings verwendet. Diese benötigen kein XML, sondern ein Array mit Kanten. Dieses wird bei uns in der Klasse “mapCoauthorships.java” erstellt.

Die Java-Klasse benötigt als Eingabe das Ergebnis der Query “Ermittle Ko-Autorschaften” (Datei: “coauthorships_complete_top_100.txt”).
In dieser sind alle Ko-Autorschaft-Beziehungen in der Form
“AffiliationID_A AffiliationID_B Anzahl”
gelistet.
Um aus dieser Tabelle Kanten auf der Karte generieren zu können, müssen wir das Ergebnis des Geocoding hinzunehmen. Mithilfe der “wichtigsten Affiliations” können wir die AffiliationIDs auf Affiliation-Namen mappen, die wir wiederum durch Mit XPath-Anfragen auf das XML in Koordinaten übersetzen können.
XPath funktioniert nicht mit dem KML-Format, allerdings erreichen wir schnell ein funktionierendes Format, indem die Zeilen “<kml…>” und “</kml>” entfernt werden. Die so veränderte Datei für die XPath-Anfragen speichern wir unter dem Namen “mapCoauthroship_input.xml”.
Die so entstandenen Kanten werden nach ihrer Häufigkeit klassifiziert und jeweils in einer separaten Datei für ihre Klasse abgelegt. Die dient dazu, die Kanten in der späteren Ausgabe unterschiedlich behandeln zu können (verschiedene Farben, Linienstärke), damit der Betrachter der Karte, die Unterschiede leichter erkennt.

Das Format in dem die Dateien aufgebaut ist hier beschrieben:

```
var kantenArray# =
[
        [ [ lng_start, lat_start ], [ lng_start, lat_start ] ],
        [ [ lng_start, lat_start ], [ lng_start, lat_start ] ],
        ...
        [ [ lng_start, lat_start ], [ lng_start, lat_start ] ]
];
```

Die entstandenen Dateien können nun in der HTML-Datei eingelesen werden. Diese behandelt sie wie JavaScript-Dateien und greift auf die Arrayelemente zu, um sie an OpenLayers MultiLineString zu übergeben.

Openlayers - Glyphen & Kanten

Die Karte nutzt verschiedene Layers um die Daten zu visualisieren. Auf der tiefsten Ebene werden die Glyphen abgetragen.

Zum Einlesen der Daten in die Glyphe wird ol.layer.Vector, ol.source.Cluster und ol.source.Vector genutzt. Dabei kann einfach eine Datei im KML-Standard

eingelesen werden und OpenLayers lässt die einzelnen Glyphen über Features darstellen.
Da die Tags im KML-Standard fest definiert sind, nutzen wir bestehende Tags um unsere Daten zu übertragen.
So speichern wir zum Beispiel die Anzahl an Vorkommnissen im “description”-Tag.
Der Aufbau des KML-Formates:

```
<?xml version=’1.0’ encoding=’UTF-8’?>
<kml xmlns=’http://www.opengis.net/kml/2.2’>
        <Document>
                <Placemark id=’normalizedAffiliationStringA’>
                        <name>originalAffiliationStringA</name>
                        <description>anzahlA</description>
                        <Point>
                                <coordinates> lngA, latA, 0 </coordinates>
                        </Point>
                </Placemark>
                <Placemark id=’normalizedAffiliationStringB’>
                .
                .
                .
        </Document>
</kml>
```

Wie oben beschrieben wird jeder Affiliation (sofern sie unterschiedliche normalizedAffiliationStrings haben), die angezeigt werden soll eine Glyphe an ihren ermittelten Koordinaten zugewiesen.
Die Glyphen für jede Affiliationen haben die Form einer Raute und der repräsentierte Koordinatenpunkt befindet sich im Zentrum dieser. Neben der Position und der Farbe, welche keine besondere Bedeutung trägt und sie nur von dem Kartenhintergrund und den Kanten abheben soll, besitzt jede Glyphe noch eine Größe.
Über die Glyphengröße wird die Häufigkeit abgetragen, mit der eine Affiliation vor kommt.
Dies bedeutet, dass Gylphen größer sind, wenn eine Affiliation an mehr Publikationen beteiligt sind.
Hierbei sei anzumerken, dass nicht nur die Anzahl der Publikationen einfließt, sondern auch wieviele Personen daran beteiligt waren (siehe Tabelle unten).
Die Berechnung einer Größe aus der Anzahl der Vorkommnisse erfolgt logarithmisch.
Eine lineare Skalierung ist nicht möglich, da der Zahlenbereich zwischen 1 und 1.510.405 liegt und demnach entweder die häufigsten Affiliations zu groß wären oder die seltenen garnicht zu sehen.

Um trotz logarithmischer Skalierung eine gewisse Größe zu garantieren wird das Ergebnis noch um 5 Punkte verschoben.
Der Radius ergibt sich für eine Glyphe dann aus folgender Vorschrift:
radius = 5 * log2(anzahl)

Damit für die Nutzung Rückschlüsse auf die Anzahlen zu schließen sind, enthalten die Glyphen noch eine Zahl in ihrem Zentrum. Diese gibt die genaue Anzahl an Vorkommnissen an.

Wir haben uns für den Glyphenansatz entschieden, um damit Überdeckungsprobleme aus dem Weg zu gehen. Dies erfolgt dadurch, dass immer dann, wenn sich zwei Glyphen zu nahe kommen, diese verschmolzen werden.
Die resultierende Glyphe bekommt eine andere Form um sie visuell unterscheidbar zu machen. Statt einer Raute ist ihre Form nun ein Kreis.
Dabei ist zu beachten, dass die Größe hier das Gebiet angibt, welches von der aggregierten Glyphe überdeckt wird.
Die Anzahl im Zentrum ist nun die Summe aller vereinigten Glyphen und die Zahl dahinter in Klammern gibt an wie viele einzelne Glyphen vereinigt wurden.

Die Glyphen werden bei jeder Änderung der Zoomstufe neu bestimmt, so dass ein Zoomen es ermöglicht gewisse Bereiche im Fokus zu betrachten oder auch einfach eine globale Sicht auf einen Bereich zu machen.

Neben den Glyphen sind auf der Karte natürlich auch die Kanten zu sehen. Diese haben hier als Merkmal unterschiedliche Farben und Linienstärken.
Dabei werden alle Ko-Autorschaften klassifiziert und in 9 Klassen eingeteilt:

Anzahl >	Farbe HEX	Linienstärke
1.500.000	#E31A1C	2.0
1.000.000	#CB181D	1.5
500.000	#CA2931	1.0
100.000	#A53F4F	0.75
50.000	#745D78	0.5
10.000	#98A093	0.25
1000	#8CB7A7	0.2

100	#85C2B1	0.15
1	#7FCDBB	0.1

Die Klasseneinteilung erfolgt über die Anzahl der Zusammenarbeiten dieser Affiliations. Seltene Beziehungen werden unaufällig und dünn dargestellt, häufige mit auffälliger Farbe und breit.
Somit ist zwar kein Ablesen von genauen Werten möglich, allerdings lassen sich wichtige (weil häufige) Beziehungen ablesen.
Es kommt noch hinzu, dass die breiten Kanten auf einem Layer über den dünnen Kanten gemalt werden. Somit ist gewährleistet, dass diese Kanten auch zu erkennen sind.

Je nach gewählter Reduzierung des Datensatz wird die Anzahl der Kanten schnell groß. So haben zum Beispiel die Top 100 der Affiliations (welche am häufigsten an Publikationen beteiligt waren) bereits 8.520 einzelne Kanten. Diese Anzahl sorgt schon zum einen für große Überdeckungen und zum anderen für Probleme mit der Performanz. Für solche Fälle ist es zu empfehlen, die Anzahl der Kanten auf die relevantesten (erneut: die Häufigsten) zu reduzieren (siehe auch Punkte: Datenreduzierung und Analyse)

Datenreduzierung
Neben dem Entfernen von “Datenmüll”, also Einträgen mit denen wir nicht umgehen können (fehlende Spalteneinträge zum Beispiel) wurden die Daten weiter eingeschränkt. So haben wir uns im Geocoding zum Beispiel auf die 1000 häufigsten Affiliations beschränkt.
Somit sind im Endergebnis auch nur diese Affiliations vertreten (und somit auch nur maximal 1000 Glyphen zu sehen).
Bei den Kanten mussten wir das sogar noch ausweiten und haben nur Kanten für die Top 100 erstellt.
Selbst dies gibt beim Anzeigen noch massive Überdeckungsprobleme, weshalb wir für eine Betrachtung der Karte empfehlen auch nur die Kanten von Beziehungen ein zu blenden, welche 50.000 oder gar 100.000 mal und häufiger vorkommen.

Analyse:
Bei den uns gegebenen Daten handelt es sich um einen sehr großen Datensatz (siehe Datenimport).

Sowohl das Analysieren als auch das Darstellen von solchen Datenmengen kann problematisch werden. Zwar ist es mit Big Data Techniken möglich, mit solchen Daten überhaupt umzugehen, allerdings gibt es dennoch Probleme in Bereichen wie der Darstellung dieser Daten.
Im Verlauf des Praktikums hat sich ergeben, dass eine Analyse über den vollen Datenumfang aufgrund technischer Einschränkungen für uns unmöglich ist und es daher nötig sein wird, die Daten einzuschränken (siehe Datenreduzierung).
So fallen zwar Daten aus der Analyse, dennoch handelt es sich um große Datenmengen, welche analysiert werden können.
Bei dieser Datenmenge und der Anforderung, die Daten auf einer Karte zu visualisieren, bietet sich ein Distant Reading Ansatz (https://www.digitalhumanities.tu-darmstadt.de/index.php?id=37) zur Analyse an. Wir versuchen also die Daten nicht Entität-für-Entität zu betrachten, sondern wir aggregieren Datenmengen, welche sich in Attributen ähneln.
Schlussfolgerungen können dann aus den aggregierten Darstellungen gezogen werden.
So fassen wir zum Beispiel Glyphen zusammen, welche geografisch gesehen nah bei einander liegen. Dies verzerrt zwar die Korrelation, da nicht nur verschiedene Institute, sondern eventuell sogar verschiedene Hochschulen aus benachbarten Städten zusammengefasst werden. Dadurch wird allerdings eine globale Sicht auf die Daten ermöglicht
So kann beispielsweise aus dem Datensatz der Top-100-Affiliations gesehen werden, dass ein umfangreiches globales Netz an Zusammenarbeiten für wissenschaftliche Publikationen besteht.
Insbesondere interkontinentale Zusammenarbeiten sind zahlenmäßig stark vertreten.
Aber auch Schlüsse über statistische Besonderheiten lassen sich mit dem Distant Reading einfach herausfinden.
Ein möglicher Schluss aus unseren Darstellungen ist beispielsweise, dass Europa einen Knotenpunkt bildet und europäische Hochschulen die häufigsten Publikationspartner sind.
Eine andere Erkenntnis ist, dass Vertreter Südamerikas oder Indiens zwar zahlenmäßig oft an Publikationen beteiligt sind, sie allerdings entweder weniger Kooperationen mit anderen Affiliations haben oder ihre Partner häufig wechseln und sie somit nicht unten den häufigsten Kanten zu finden sind.

Wenn man nun versucht einen Ort zu finden, an dem die meisten Ko-Autorschaften zusammen kommen, dann liegt dieser in Frankreich. Genauer gesagt bildet Genf einen Knotenpunkt, bei dem nicht nur Kanten von verschiedenen Kontinenten eingehen, sondern auch die meisten Zusammenarbeiten mit festen Partnern (dicke Kanten) auftreten.