Free University of Bozen-Bolzano
Faculty of Computer Science

Discovering Related Terms on a large Solr Document Database

Ettore Ciprian

Supervisor

Prof. Dr. Johann Gamper

Company Tutor
Dr. Christoph Moar
Alpin AG

September, 2015

Abstract

The search for documents and information represents the main activity for researchers or ordinary users of a library, especially if it is easily accessible with any Internet-connected device. Nowadays, the amount of these particular virtual libraries, without shelves and librarians, is increasing due to the digitalization of paper content. The responsibility of finding a specific books, however, is left to the individual user, that knowing *where* to look, it is not necessarily aware of *what* and *how* to obtain the desired information. In our region, thanks to the project *Teßmann Digital*, you can now search among more than two million pages of historical documents of South Tyrol, including books, newspapers and prints.

This thesis will consider a solution, in the framework of the Teßmann Portal, to enhance the experience of the researcher dealing with such a large archive.
Specifically, I analyzed the discovery of related terms, terms in relevance to a given search term in the corpus of documents, which will be used to provide new search possibilities to the user.
The final solution integrates in the architecture already in use on the extensive archive, not only to maintain sufficient response performance during the search, but taking into account the relevance of the results and effectiveness that these can have in improving the user search experience.
At the level of implementation, it will be considered the search engine Apache *Solr*, already integrated in the portal, and two possible compatible solutions are evaluated, that are the use of *term vectors* or *clustering* algorithms. Finally, I have implemented a browsable *word cloud* in order to show the related terms, which belong to specific categories: people, organizations and places.

Abstract (Italiano)

La ricerca di documenti e informazioni rappresenta la principale attività per il ricercatore o ordinario fruitore di una biblioteca, nondimeno se quest'ultima è comodamente accessibile con qualsiasi dispositivo connesso a Internet. Al giorno d'oggi, la presenza di queste particolari biblioteche virtuali, senza scaffali e librai, è in continuo aumento grazie alla crescente digitalizzazione dei contenuti cartacei.
La responsabilità del trovare uno specifico contenuto è però lasciata al singolo utente, che pur sapendo *dove* cercare, non è necessariamente a conoscenza di *cosa* e di *come* ottenere le desiderate informazioni.
A livello locale, grazie al progetto *Teßmann Digital*, è ora possibile cercare tra più di due milioni di pagine tra documenti storici dell'Alto Adige, tra cui libri, stampe e giornali.

In questa relazione verrà valutata una soluzione per migliorare l'esperienza del ricercatore alle prese con un grande archivio, come quello disponibile sul portale. Nello specifico, viene analizzata la scoperta di *termini correlati*, termini in rilevanza ad un dato termine di ricerca all'interno del corpus di documenti, che serviranno per fornire nuovi spunti di ricerca all'utente.
La soluzione finale vede l'utilizzo della tecnologia e architettura già in uso sull'esteso archivio per mantenere non solo sufficienti le prestazioni di risposta durante la ricerca, ma che prenda in considerazione la rilevanza dei risultati e l'efficacia che questi possono avere nel migliorare l'esperienza di ricerca dell'utente.
Viene infatti considerato il search engine *Solr* di Apache, già integrato nel portale, e vengono estratte e valutate due possibili soluzioni compatibili, che sono l'utilizzo di *term vector*s o di algoritmi di *clustering*. A fruizione dell'utente, viene quindi prodotta una browsable *word cloud* di termini correlati racchiusi in specifiche categorie: persone, organizzazioni e posti.

Abstract (Deutsch)

Die Suche nach Informationen und Dokumenten ist eine wichtige Aktivität für Forscher und Bibliotheksbenutzer, vor allem wenn diese über Internet zugänglich ist. Virtuelle Bibliotheken ohne Regale und Bücher nehmen zu, dank der Digitalisierung von Drucksachen.
Das Durchsuchen von spezifischen Dokumenten in diesen Bibliotheken ist allerdings oft schwierig für Nutzer, da diese wissen müssen *wo* und *wie* gewünschten Informationen zu finden sind.
In der Region gibt es dank des Projekts *Teßmann Digital* jetzt die Möglichkeit mehr als 2 Millionen Seiten von historische Dokumenten aus der Region zu durchsuchen, darunter Bücher, Zeitungen und andere Drucksachen.

Hier präsentiere ich eine Lösung welche die Informationen im Teßmann-Portal für Nutzer einfacher zugänglich macht. Ich habe das Finden von ähnlichen Begriffen analysiert, was dazu verwendet werden kann die Suche für Nutzer zu verbessern. Meine Lösung passt in die existierende Architektur und verbessert die Relevanz der Resultate und damit das Benutzererlebnis, ohne nennenswerte Performanceeinbußen. Dafür wurde die Suchmaschine Apache Solr verwendet und in das Portal integriert. Es werden zwei mögliche, kompatible Lösungen vorgeschlagen: *term vectors* und die Verwendung eines *Clustering-Algorithmus*. Es wird damit eine "Word Cloud" mit verschiedenen ähnlichen Begriffen generiert, die in die Kategorien Personen, Organisationen und Orte eingeteilt ist.

Contents

I Introduction

1.1 Motivation

Nowadays, the way people accesses information from printed documents has changed. Thanks to technology and new precise information retrieval techniques, we are able to search for a specific term across a six hundred pages book written a century ago with the comfort of a simple smartphone. This is because the entire written human heritage is turning into an easy-to-use, compact and freely accessible format.
Because of the advantages of digital over physical material, such as term search, faster linguistic analysis, portability, etc., libraries and collections all around the world have started to embrace the digitalization of their holdings. For example, The Library of Congress and the British Library, the two biggest and most comprehensive selections of multi-language documents, have both launched initiatives for publishing their material online, making more than 10 million items available to the public.
This enormous amount of data is now accessible from over 3 billion of users, leading to several dilemmas for both information technology experts and final consumers. One of the most pressing need is well summarized in the *information retrieval paradigm*:

> *Given a set of documents and a query, determine the subset of documents relevant to the query* (Selberg, 1999)

While the paradigm states an obvious and widely discussed problem, further can be done in helping users refining the research for documents. Unfortunately, only a fraction of them is able to find appropriate shelves and books in this huge library. The knowledge of *where* to seek for information turns out to be not enough, thus, I worked on a solution for a local digital library, which should help an ideal researcher to better filter his/her research in a large document archive.

1.2 The Problem

The main goal was to make available for the user of the Teßmann Library portal specific metadata of the OCR-ed documents such as *persons, locations* and *organizations* cited in the text.
The user should be able to know which of the three categories, from now on

referred as *Named Entities*, are contained in a page of a document, and it should be able to filter a research by more than one Entity. More importantly, the user should know which Named Entities have more frequently occurred in relation to the query term and visualize this information in a way that the strength of the connection is intuitively clear.
For instance, I want to search for the term "*Andreas Hofer*" and as a result I am expecting not only to know in which pages it appears most, but also with which locations, organizations and persons "*Andreas Hofer*" has been more frequently mentioned.
Formally: given a set *N* of document which contains at least one occurrence of the query term *q*, return the set *R* containing all the terms contained in *N* ordered by importance.

1.3 The Solution

The solution I will discuss in this paper is the use of clustering algorithms, such as Lingo and STC, compared to a simpler usage of the filter capabilities of the Solr server. The idea is to collect, with clustering or term vectors, all the most inversely frequent Named Entities that appears after the overall document set has been reduced by the search term, that is, a subset of document containing the query term. Correctness of the result, performance and integration in the existent system architecture will be the leading factors in the final choice of implementation.

1.4 Related Work

While the purpose of this paper is, given a pre-ordered list of elements, to find the most frequent word by association to the search query, different researches and algorithms other than the one cited here have been developed for the *associated term retrieval* on a full text. In *Information Retrieval Architecture and Algorithms* by Gerald Kowalski (2011) is reported the general term clustering procedures and algorithms, focused on generating a helpful thesaurus of topics from a document text. In *Web Document Clustering: A Feasibility Demonstration* (Zamir and Etzioni, 1998) which set the foundation for meaningful term-based clusters retrieval, with an algorithm (explained later in Paragraph 4.4.1) that allows the categorization of full sentences surrounding a term and applications that span from simple textual description to Web Search results.
However, the main contribution to this thesis was given by the critics to STC

itself, a new algorithm that provides clusters with a new top down approach and revaluates the importance of sentences surrounding the query term. It is the work of Stanislaw Osinski in *An Algorithm for clustering of web search results* (2003), which reports the implementation of the algorithm *Lingo.*

1.5 Organization

In *Chapter 1,* I will introduce the problem at the origin of my proposed solution and in *Chapter 2* it will be illustrated the context in which my proposition has been developed. In *Chapter 3* the architecture of the project is analyzed, for a better understanding of the integration possibilities. Finally, in *Chapter 4* I will explain the possible solutions in detail, given the problem stated; in *Chapter 5* I will provide the final test results and conclusions about the discovering of related terms in a large Solr document database.

II Background

2.1 The OPATCH Project

The Open Platform for Access and Analysis of Textual Documents of Cultural Heritage (OPATCH, s.d.) is an initiative financed by the “Provincia Autonoma di Bolzano - Alto Adige”, which main aim is to provide a set of standards and tools for accessing and analyzing a large collection of electronic documents, specifically, the corpus of an historical document collection. The main intent is to use open source front-end and back-end technologies to allow a comprehensive and extended content search, with the final goal of creating a set of tools for linguistic analysis.
Currently, the project is relying on the data provided by Teßmann Library, which corpus infrastructure has been OCR-ed by a group of researchers at the Institute for Specialized Communication and Multilingualism inside the Eurac Research Center, Bolzano. In collaboration with Alpin S.r.l, the company behind all of the development process of the portal and for which I worked in order to produce this paper, they opened a virtual library called *Teßmann Digital.*

2.2 The Teßmann Digital Library

The Dr. Friedrich Teßmann Library collects historical documents, mainly newspapers, publications and artworks from 1800 published in South Tyrol, Italy. In conjunction with the researchers at Eurac and Alpin S.r.l, Teßmann opened a portal (http://digital.tessmann.it/), which gives free access to more than 147,400 documents with more than 2,280,000 pages.
What is peculiar about the archive is that the research feature is not limited to metadata, but there is also the possibility to search through actual newspapers and books content. Moreover, the webpage highlights the term queried directly on the scanned image of the founded page, providing the user a normalized version of the text.

Figure 2.1 – The Tessmann Digital Portal

This has been made possible due to the Optical Character Recognition research on *Fraktur* font based text, which was carried out at Eurac. Firstly, the text is scanned and cleaned from noise, such as punctuation and bad recognitions. Lastly, the obtained clean text is marked with appropriate dictionaries: the result is an ALTO standard xml file.

2.3 Analyzed Layout and Text Object

Analyzed Layout and Text Object (ALTO) is a XML Schema that details technical metadata for describing the layout and content of physical text resources, such as pages of a book or a newspaper (Alto - Technical Metadata for Optical Character Recognition, 2014).
An ALTO file consists of three major sections as children of the root `<alto>` element:

`<Description>` `<Styles>` `<Layout>`

The `<Description>` section contains metadata about the ALTO file itself and processing information on how the file was created.
The `<Styles>` section contains the text and paragraph styles with their

individual descriptions:

- `<TextStyle>` has font descriptions
- `<ParagraphStyle>` has paragraph descriptions, e.g. alignment information

The `<Layout>` section contains the content information. It is subdivided into `<Page>` elements.
A page consists of margins and print space; all of those are non-intersection rectangular areas within the page area. Each of these can contain any number of objects like lines, images or text blocks and more. A text block is divided into text lines and those are further divided in strings and spaces.

Figure 2.2 - A page of the "Pustertaler Bote" parsed into Alto

```
    <NamedEntityTag ID="Tag-LOC-ST:622" TYPE="LOC-ST" LABEL="Tagen"/>
    <NamedEntityTag ID="Tag-LOC:655" TYPE="LOC" LABEL="Rumänenvolkes"/>
    <NamedEntityTag ID="Tag-LOC:743" TYPE="LOC" LABEL="Tarnopol"/>
    <NamedEntityTag ID="Tag-LOC-ST:813" TYPE="LOC-ST" LABEL="Front"/>
    <NamedEntityTag ID="Tag-LOC-ST:884" TYPE="LOC-ST" LABEL="Karpathen"/>
    <NamedEntityTag ID="Tag-LOC:761" TYPE="LOC" LABEL="Trembowla"/>
    <NamedEntityTag ID="Tag-LOC-ST:920" TYPE="LOC-ST" LABEL="Früh"/>
    <NamedEntityTag ID="Tag-LOC-ST:936" TYPE="LOC-ST" LABEL="Boden"/>
    <NamedEntityTag ID="Tag-PER-ST:943" TYPE="PER-ST" LABEL="Feuer"/>
    <NamedEntityTag ID="Tag-PER:932" TYPE="PER" LABEL="Mackensen"/>
    <NamedEntityTag ID="Tag-LOC:964" TYPE="LOC" LABEL="Balkan"/>
    <NamedEntityTag ID="Tag-PER-ST:963" TYPE="PER-ST" LABEL="Am"/>
    <NamedEntityTag ID="Tag-LOC-ST:983" TYPE="LOC-ST" LABEL="Flandern"/>
    <NamedEntityTag ID="Tag-PER-ST:1015" TYPE="PER-ST" LABEL="Maas"/>
    <NamedEntityTag ID="Tag-LOC-ST:1044" TYPE="LOC-ST" LABEL="Schwarzen"/>
    <NamedEntityTag ID="Tag-LOC-ST:1051" TYPE="LOC-ST" LABEL="Kämpfe"/>
    <NamedEntityTag ID="Tag-LOC-ST:1114" TYPE="LOC-ST" LABEL="Dorf"/>
    <NamedEntityTag ID="Tag-LOC:974" TYPE="LOC" LABEL="Berlin"/>
    <NamedEntityTag ID="Tag-PER:1123" TYPE="PER" LABEL="Böhm-Ermolli"/>
    <NamedEntityTag ID="Tag-LOC:1144" TYPE="LOC" LABEL="Linien"/>
    <NamedEntityTag ID="Tag-LOC-ST:1121" TYPE="LOC-ST" LABEL="Sand"/>
  </Tags>
- <Layout>
  - <Page HEIGHT="4079" ID="Page1" PHYSICAL_IMG_NR="1" WIDTH="2799">
      <TopMargin HEIGHT="674" HPOS="0" VPOS="0" WIDTH="2799"> </TopMargin>
      <LeftMargin HEIGHT="3140" HPOS="0" VPOS="674" WIDTH="44"> </LeftMargin>
      <RightMargin HEIGHT="3140" HPOS="2581" VPOS="674" WIDTH="218"> </RightMargin>
      <BottomMargin HEIGHT="265" HPOS="0" VPOS="3814" WIDTH="2799"> </BottomMargin>
    - <PrintSpace HEIGHT="3140" HPOS="44" VPOS="674" WIDTH="2537">
      - <TextBlock HEIGHT="103" HPOS="54" ID="Page1_Block1" STYLEREFS="font6" VPOS="674" WIDTH="2458" language="de">
        - <Shape>
            <Polygon POINTS="807,674 1571,674 1571,681 1874,681 1874,687 1875,687 1875,688 2506,688 2506,771 2512,771 2512,777 54,777 54,771 60,771 60,6
          </Shape>
        - <TextLine HEIGHT="74" HPOS="60" VPOS="696" WIDTH="2446">
            <String CONTENT="(Brunecker" TAGREFS="" HEIGHT="66" HPOS="60" STYLE="bold" VPOS="700" WC="0.3880000114" WIDTH="340"/>
            <String CONTENT="Zeitung)." TAGREFS="" HEIGHT="72" HPOS="428" STYLE="bold" VPOS="698" WC="0.4055555463" WIDTH="296"/>
            <String CONTENT="Politisches" TAGREFS="" HEIGHT="74" HPOS="756" STYLE="bold" VPOS="696" WC="0.6090909243" WIDTH="342"/>
            <String CONTENT="Lokal-" TAGREFS="" HEIGHT="62" HPOS="1126" STYLE="bold" VPOS="698" WC="0.5699999928" WIDTH="194"/>
            <String CONTENT="und" TAGREFS="" HEIGHT="58" HPOS="1344" STYLE="bold" VPOS="700" WC="0.2533333302" WIDTH="116"/>
            <String CONTENT="Provinzblau." TAGREFS="" HEIGHT="70" HPOS="1486" STYLE="bold" VPOS="698" WC="0.462500006" WIDTH="426"/>
            <String CONTENT="—" TAGREFS="" HEIGHT="8" HPOS="1948" STYLE="bold" VPOS="728" WC="1." WIDTH="68"/>
            <String CONTENT="«7." TAGREFS="" HEIGHT="58" HPOS="2046" STYLE="bold" VPOS="698" WC="0.7400000095" WIDTH="102"/>
            <String CONTENT="Jahrgang." TAGREFS="" HEIGHT="72" HPOS="2182" STYLE="bold" VPOS="696" WC="0.5511111021" WIDTH="324"/>
          </TextLine>
        </TextBlock>
```

As shown in the scanned page (Figure 2.2), from version 3.0 of the schema it is possible to mark, separately from the `<Page>` block, a series of specific words, the Named Entities. Each Entity has a type tag:

POS/POS-ST: stores a location name
ORG/ORG-ST: stores organizations, as well as shops and trademarks
PER/PER-ST: holds first, last or complete person names.

While in the previous definition of the Schema it was not possible to hold different variables than text lines, the new standard together with linguistic and historical researches has made 1000 newspaper issues with Named Entites available. More than one million pages will be added in the next future.
My work will be focused on collecting these data arrays, extracting the related terms and display them in a meaningful way for the end user, taking into account performances, the existent project architecture and the technology used.

III The Architecture

3.1 Overview

The following diagram illustrates the existent *Teßmann Portal* architecture.

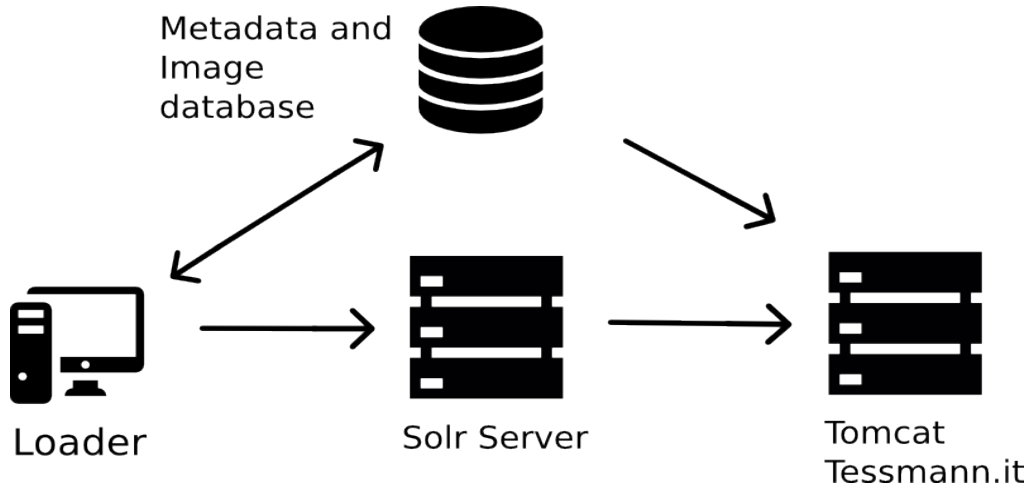

Figure 3.1 – overall architecture

In figure 3.1, it is possible to see the main components of the project architecture. I focused the major changes in the *loader* and the *Solr server* when it came to index and retrieve the Named Entities, while I made modifications on the presentation layer of the Tessmann portal in order to display the results.

3.2 Loader

The Loader is the Java Application client responsible for parsing and committing the xml Alto file into the metadata Database and in the Solr index. It reads each single xml page using the Schema Definition, then it queries the metadata database for extra information or it updates previous entries and, finally, it commits the full text of the documents to the Solr Server, where they are indexed and stored. As one can imagine, this operation possess high performance costs and it is executed only when large batch of documents have been converted into ALTO.

3.3 Solr Server

Solr is a well-established standalone server application, which is responsible for indexing and rapidly search through the full text of the documents. (http://lucene.apache.org/solr/)
When the data have been committed is possible to query the server with standard RESTful methods and obtain either XML or JSON in return.
In this architecture, Solr is queried by the Tomcat server and returns an XML formatted answer.

Solr demonstrated to be a versatile and customizable tool: it is possible to define multivalued field type, search handlers and query facets for filtering, features that came in handy for the main task. Furthermore, it has been possible to integrate different clustering algorithm through the Carrot2 library (http://project.carrot2.org/).
Note that the version used in this architecture is 4.2.1, updates might have been pushed on the current 5.x, but the following chapter will be based entirely on the older configuration.

3.4 Hardware

Being the application almost completely server based, it is worth mentioning the hardware used. Especially with clustering analysis, hardware specifics have a great impact on performances and thus can better explain the test results and limitations in the tuning of the clustering algorithm.
The Tomcat application and the Solr server run on the same virtualized server with the following specifics:

RAM: 6 GB
CPU: 1 virtual CPU
3 Hard drives: 25 GB, 500 GB, 250 GB

The index is stored on a fast solid drive for higher reading performance, when tested it scored 251.00 MB per second in reading.

IV Implementation

When I was facing the overall problem and I was given the use cases (*Paragraph 1.3 – The Problem)*, I divided it in three minor subtasks:

1. Making the Named Entites available to Solr
2. Use the best performing Solr functionality to retrieve the correct data
3. Display the data in a meaningful way

In the following chapter I will explain in detail how to achieve the tasks mentioned, reporting the implementation possibilities. In *Chapter 5* I will instead narrow down to the most suited solution.

4.1 Making the Named Entities Available

The first task was easily achievable due to Solr versatility. I simply enhanced the *Loader* in order to read the XML files of the document, looping over the Named Entities tags and extracting the base String content. After that, it was sufficient to commit the data into the server with the ad-hoc Solr java API (Sorlj, 2013).

```
<field name="location" type="text" indexed="true" stored="true"
multiValued="true"/>

<field name="person" type="text" indexed="true" stored="true"
multiValued="true"/>

<field name="organization" type="text" indexed="true" stored="true"
multiValued="true"/>
```

As shown above, by defining a multivalued field in the Solr configuration file, it is possible to store arrays of data. This allows Solr internal mechanism to select the elements into the array and address specific fields in a search request.
Now the Named Entites are stored by the server application, together with the full text of the document and other metadata such as author, publishing year etc. Additionally, each document has one unique id assigned by Solr.

4.2 Retrieving Data from Solr

In general, it is the possible to query Solr with the */clustering* search parameter, or by any of the *SearchComponent* defined, for example a */select* or */tvrh* for Term Vector retrieval. A simple search request might be:

```
http://localhost:8983/solr/tessmann/select?q=Hofer&start=0&rows=10&fl=fulltext+mediaName
```

Different parameters can produce specific filters in the JSON response. In this case, it asked Solr to return a list of terms in the *fulltext* and *mediaName* field, containing the term *Hofer* and returning the first ten match.
After a study of the Solr architecture and features, I came out with two possible solutions, which I will proceed to evaluate in the next chapter: the use of *clustering* algorithms or *term vectors*.

4.3 Term Vectors

The *TermVectorComponent* is a search component in Solr designed to return additional information about documents matching a query term.
The term vector must be enabled before committing the data, because information are permanently stored at commit time.

```
<field name="location" type="text" indexed="true" stored="true"
multiValued="true"
termVectors="true"
termPositions="true"
termOffsets="true"/>
```

For each document in the response, the *TermVectorComponent* can return the term vector, the term frequency, inverse document frequency, position, and offset information.
When properly interrogated from the Tomcat server by means of ad-hoc classes, the server will output a XML structured response with the selected document to which is appended a list of the desired Named Entities and frequency scores.

```
<lst name="37-128387-1212644">
  <str name="uniqueKey">37-128387-1212644</str>
    <lst name="mediaPerson">
      <lst name="alois">
        <int name="tf">2</int>
        <int name="df">3152</int>
        <double name="tf-idf">6.345177664974619E-4</double>
      </lst>
     <lst name="august">
       <int name="tf">1</int>
       <int name="df">4727</int>
       <double name="tf-idf">2.115506663845991E-4</double>
     </lst>
    <lst name="benjamin">
      <int name="tf">1</int>
      <int name="df">92</int>
      <double name="tf-idf">0.010869565217391304</double>
    </lst>
    <lst name="borgo">
      <int name="tf">1</int>
      <int name="df">164</int>
      <double name="tf-idf">0.006097560975609756</double>
    </lst>

...
```

For example, the *mediaPerson* Entity of this particular document has “alois” appearing 2 times (*tf* – term frequency) with an overall document frequency (*DF*) of 3152 and inverse document frequency of 6.345 (*tf-idf*).
The function term frequency - inverse term frequency reflects how important a word is to a document in a collection or corpus, that is, whether the term is common or rare across all documents (Rajaraman & Ullman, 2011). The *idf* is the logarithmically scaled fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{Idf}(t, D) = \log(N/\mathrm{df})$$

The final weight of the word is calculated as the *tf–idf*, by multiplying the term frequency with the inverse document frequency.

$$\text{Tf-idf}(t, d, D) = \mathrm{tf}(t, d) * \mathrm{idf}(t, D)$$

where *t* is term, *d* is the number of document and *D* the document result set.

The *tf-idf* value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. In our case, it is appropriate when it comes to weight *locations* and *first names* frequency, which are common Named Entites and not necessarily related.

The idea is to use a class defined in the Portal business layer, in order to retrieve the inverse document frequency of each Named Entities for each document in the result set, and compare the resulting lists. Since Solr returns only the *tf-idf* repeated for each term of each document of the resulting set, it necessary to merge all *tf-idf* labels, resulting in all Named Entities statistically more important and closer to the term query.
This solution will be compared with my second proposition, that is, the use of clustering algorithms.

4.4 Clustering in Solr

The second solution I have considered is the integration of clustering algorithm in order to retrieve the related Named Entities. In this section, I will limit the report to the implementation and behavior of the algorithms available for a Solr instance, while in the next Chapter I will provide the evaluation and comparison of the clustering options.
Let's start with a quick introduction on the subject:

The definitions in the literature of what constitutes clustering reflect the different philosophical points on the matter. The top-down view regards clustering as the segmentation of a heterogeneous population into a number of more homogeneous subgroups. A bottom-up view defines clustering as finding groups in a data set "by some natural criterion of similarity" (Estivill-Castro, 2002) .

In this particular case, clusters analysis will serve the purpose of grouping the Named Entities by frequency, which will be the criterion of similarity used to distinguish the clusters.
This is exactly what two of the main algorithms contained in the *Carrot2* open source framework are able to compute, making it the natural candidate for this task. Currently available in the framework are STC (Suffix Tree Clustering) and Lingo algorithms, which main functions are exposed in the

following subchapters.

4.4.1 *Suffix Tree Clustering*

The Suffix Tree Clustering (STC) algorithm groups the input texts according to the identical *phrases* they share. The rationale behind such approach is that phrases, compared to single keywords, have greater descriptive power. This results from their ability to retain the relationships of proximity and order between words. A great advantage of STC is that phrases are used both to discover and to describe the resulting groups. The pseudo-code of the Suffix tree algorithm (Osinski, 2003) is reported below.

```
/** * Suffix Tree Clustering algorithm */
split text into sentences consisting of words;
/* Phase 1a. Creation of a Generalized Suffix Tree of all sentences
*/
for each document {
  for each sentence {
    if (sentence length > 0) {
      insert sentence and all its substrings into generalised suffix
      tree and update internal nodes with the index to current
      document while rearranging the tree;
                }
           }
      }
/* Phase 1b. Build a list of base clusters */
for each node in the tree {
  if (number of documents in node's subtree > 2) {
      if (candidateBaseClusterScore > Minimal_Base_Cluster_Score) {
        add a base cluster to the list of base clusters;
      }
  }
}
/* Phase 2.   Merge base clusters */
      build a graph where nodes are base clusters and there is a
      link between node A and B if and only if the number of common
      documents indexed by A and B is greater than the
      Merge_Threshold;
      clusters are coherent subgraphs of that graph;
```

Figure 4.1 - STC pseudo-code

Taking into account that in this case it is operating on multivalued lists, containing only cleaned text such as the Named Entities, the algorithm will actually proceed as follow:

1. Create a tree for each document;
2. Create a *node* in the document Suffix Tree for each instance of them,

thus, each unique Named entity will count as cluster;
3. Based on a Merge Threshold value, it will try to merge the lower nodes, thus, the Named Entities in the different documents sub-trees.

The algorithm will then output a list of clusters sorted by score, that is, the frequency of each Named Entity.

4.4.2 *Lingo*

```
/**  Lingo algorithm */

/** Phase 1: Preprocessing */
for each document
{
do text filtering;
identify the document's language;
apply stemming;
mark stop words;
}
/** Phase 2: Feature extraction */
discover frequent terms and phrases;
/** Phase 3: Cluster label induction */
use LSI to discover abstract concepts;
for each abstract concept
{
find best-matching phrase;
}
prune similar cluster labels;
/** Phase 4: Cluster content discovery */
for each cluster label
{
use VSM to determine the cluster contents;
}
/** Phase 5: Final cluster formation */
calculate cluster scores;
apply cluster merging;
```

Figure 4.2 – Lingo pseudo-code

In Figure 4.2 the main phases of Lingo, as for STC, will operate on multivalued lists instead of full text sentences. Once it is run on the Teßmann database, it will:

1. Calculate the *term frequency - inverse term frequency* of each Named Entity;
2. Use Latent Semantic Indexing (LSI) to establish candidate labels first, thus establishing the name of all the clusters before determining

the content;
3. For each candidate cluster label, that represent the *abstract concept* of the cluster, applies VSM (Vector Space Modelling) in order to discover the content of the cluster and assign the documents;
4. Merge similar clusters.

As reported from Osinski in *An Algorithm for clustering of web search results*, LSI plays a major role in defining better clusters, compared to Vector Space Model:

LSI tries to overcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated Singular Value Decomposition (SVD) is used to estimate the structure in word usage across documents. Retrieval is then performed using a database of singular values and vectors obtained from the truncated SVD. Performance data shows that these statistically derived vectors are more robust indicators of meaning than individual terms. (Michael W. Berry, 1995)

While LSI and VSM are not, unfortunately, object of this paper, it sufficient for the reader to know that LSI is a better information retrieval system when facing similar words, especially synonyms and overlapping words, and refines the search for most frequent documents more than VSM. Successful though the classic Vector Space Model is, it does not address many of the problems of modern Information Retrieval. First of all, the model is based on literal matching between terms from a document and those of a query. Due to the phenomenon of *synonymy* however, there are usually many ways in which the same idea can be expressed- the terms found in the user's query may not match those of a relevant document. In this way, some of the documents will not appear in the search results even though they should. Second of all, the Vector Space Model makes an assumption that the keywords used to index the document collection are mutually independent (Osinski, 2003).
Nevertheless, considering multivalued lists as data input instead of a full-sentences text, as expected by the algorithm creators, it is interesting to see the merging of synonyms in Named Entities retrieval. As we will see in following Chapter, the added value of a similar word merging technique

should be taken into consideration, for example, in case of unifying words such as "*Stadt Bozen*" and "*Bozen*" under the same cluster, or better, in the composition of new related terms which could enrich the information retrieval experience for the final user.

4.4.3 *Clustering Data Retrieval*

Retrieving data from the server application is then achievable by using the previously mentioned Search Component:

```
<searchComponent name="clustering"
      enable="${solr.clustering.enabled:true}"
      class="solr.clustering.ClusteringComponent" >
      <!-- Declare an engine -->

      <lst name="engine">
      <!-- The name, only one can be named "default" -->
      <str name="name">default</str>

      <!-- Class name of Carrot2 clustering algorithm.-->
      <str name="carrot.algorithm">
      org.carrot2.clustering.lingo.LingoClusteringAlgorithm
      </str>

      <!-- Overriding values for Carrot2 default algorithm attributes.
      http://download.carrot2.org/stable/manual/#chapter.components.-->
      <str name="LingoClusteringAlgorithm.desiredClusterCountBase">20</str>

      <!-- Location of Carrot2 lexical resources.
      http://download.carrot2.org/head/manual/#chapter.lexical-resources-->
      <str name="carrot.lexicalResourcesDir">clustering/carrot2</str>

      <!-- The language to assume for the documents.
      http://download.carrot2.org/stable/manual/#section.attribute.lingo.Multi
      lingualClustering.defaultLanguage
      -->
      <str name="MultilingualClustering.defaultLanguage">GERMAN</str>
      </lst>

      <!--STC engine definition -->
      <lst name="engine">
      <str name="name">stc</str>
      <str name="carrot.algorithm">
      org.carrot2.clustering.stc.STCClusteringAlgorithm
      </str>
      </lst>
</searchComponent>
```

Figure 4.3 – Search Component configuration for clustering

As shown in Figure 4.3 it is necessary to import the necessary algorithms

classes on Solr, as well as enabling it as a *select-first* component. By querying the Solr instance, it will output an xml formatted document (Figure 4.4).
Each *labels*, represent the name given by the algorithm to a certain cluster. Each clusters possess a *score*, which determines the label overall importance, and an array of attached documents IDs where the label has scored a match. Of course, the same document could be contained in different clusters.

```
<lst>
  <arr name="labels">
    <str>Bozen</str>
  </arr>
  <double name="score">23.0</double>
  <arr name="docs">
    <str>37-128491-1214326</str>
    <str>37-127454-1197446</str>
    <str>37-128199-1209707</str>
    <str>37-128798-1217945</str>
</lst>
<lst>
  <arr name="labels">
    <str>Wien</str>
  </arr>
  <double name="score">21.0</double>
  <arr name="docs">
    <str>37-128491-1214326</str>
    <str>37-127454-1197446</str>
    <str>37-129640-1223858</str>
  </arr>
</lst>
<lst>
  <arr name="labels">
    <str>Bayern</str>
  </arr>
  <double name="score">19.794981002807617</double>
  <arr name="docs">
    <str>37-129228-1220820</str>
  </arr>
</lst>
```

Figure 4.4 – XML response form clustering algorithms in Solr

4.4.4 *Clustering Optimization*

Clustering is the most demanding resource solution compared to Term Vectors. In order to make it more suitable to a given database there are different tuning strategies I have applied:

- *Caching* of results array is stored at level of the business layer and

Solr server itself. This brings response time sensible down in case of consequent requests.

- *Maximum cluster limits* have been passed as parameters to the clustering requests. In case of STC and Lingo, both algorithms will stop discovering clusters at the desired limit. While computing 1000 pages, for example, it is sufficient to return no more than 100 from the top scoring clusters. This reduced sensibly the algorithm running time and untimely filters the amount of data to display.
- *Prioritization of search fields*: due to Solr Search Component capabilities, it is possible to give priority to certain search fields by a value of 0.0 to 1.0. This option revealed to be useful when directing the algorithm, and accordingly to the requirements it has been given priority to the Named Entity Solr internal fields (see *Paragraph 4.1 - Making the Named Entities Available*).
- *Disabling word filtering*: in this particular case, techniques such as *stemming*, *word filtering* and *stops words*, applied by default by Solr Clustering components, are not necessary. Those functions can be disabled because we are working on a sufficiently cleaned text.

4.5 Visualization

The role of the search results presentation interface is to display the results in a way that helps the users to identify the specific documents they searched for. Thus, an ideal interface should not make the users stumble upon the documents that they would judge as irrelevant.
The presentation should take into account the two following principles:

- *Relevance*: The algorithm ought to produce clusters that group documents relevant to the user's query separately from irrelevant ones. The user shall be able to determine at a glance which elements are more related to the query term provided, in comparison to the other results.
- *Browsable results*: The user needs to continue the research easily and intuitively by starting another query from the results given.

The choice of implementation felt upon the use of a Word Cloud (*Figure 4.5*), which provides the user an instant and intuitive view of the more relevant results. Furthermore, the user can issue another search query by clicking on an element of the cloud.In practice, data presentation on the Webpage has been achieved by feeding either the results of TermVectors or

Clustering, through a class I have implemented, to a simple JavaScript based tool.
https://github.com/lucaong/jQCloud
This particular plugin accepts a JSON array as input and displays a cloud shaped block of texts highlighted by the score of the clusters/termVectors.

Figure 4.5 - The final related term visualization on the Portal

By using a combination of HTML, CSS and Velocity expressions on the presentation layer of the Portal, I have styled the cloud to display results colors and size accordingly to a scale of 1 to 10. The java script library provided by jQCloud will then group by this scale all the clusters, proportionally to the weight of each label.

V Evaluation

5.1 Data Set

I have proceeded to test the three solutions on the real Teßmann corpus database using an actual block of newspapers with aggregated Named Entities provided by Teßmann.
The test database is composed as follow:

24.786 pages from the "*Bozner Nachrichten*" (1910-1920)
898 pages from the "*Pustertaler Bote*" (1917 -1920)
1 gigabyte index
25.684 total pages

The running time comparison will be executed by querying the Solr instance from a local client application of the Portal. It will be first issued a "select by term" request, that is a simple search query, and then the resulting document set will be feed to the three algorithms.
All three operations will be executed on the first 10, 100 and 1000 documents of the reduced set. As it will be clear from the running time comparison, executing those three techniques on a larger or the exact set of documents would be unthinkable: the size of the *select* returning set is not know in advance and the running time will be far too long for the final user performance requirements. Besides, the first 1000 pages represent usually a good subset of analysis compared to the total size of documents possibly found.
For the purpose of testing, I have chosen terms such as "*Bozen*", "*Hofer*" or "*Meran*" as main query terms for all the tests because they returned a sufficient number of documents, more than one thousand.
The running time of the query is returned by the Solr application itself, expressed in *millisecond*, and represents the effective time needed to execute the algorithms on Solr, while response time of the REST request is left out. The query times are computed from the average of 10 different requests.

5.2 Running Time Comparison Table

Documents	10	100	1000
Term Vectors	71	568	1151*
STC	58	432	1701
Lingo	31	517	3940

**It is important to notice that the application issuing the requests crashed several times due to long response received by the test queries. In fact, data transmitted through the servers revealed to be far more compact in case of clustering, contrarily to Term Vectors, which are not merged in place but on the Tomcat server. This issue is to be considered for the final evaluation.*

In general, Lingo revealed to be the less scalable solution, while Terms Vectors scored the best performing results.

5.3 Correctness and Expressivity of the Results

In order to test the algorithm for correctness I have run further analysis tests resulting in the following statistics divided per field. In order to make sense of the algorithm, the statistics were executed based on a subset of the overall document database, which is reduced by the same query term *q* for which we want to find the related terms. In this test report, five different sufficiently common terms were selected as *q*.
As one can notice, some entities do not always belong to the category in which they are assigned to. The reader must consider that the data provided by Eurac is still object of continuous research and improvements at the time this paper was written.

5.3.1 *Top 10 Terms by Frequency*

Here is reported the most frequent words together with the inverse document frequency score founded manually when reducing the document set by the word *Bozen*. As the reader can imagine, reducing the set by

different words rarely change the output of this table, especially if it is a common word.

Location	Organization	Person
15,473 bozen	3,442 bozen	6,848 frau
12,343 bozner	3,267 nachrichten	6,662 franz
8,303 wien	2,282 hotel	6,297 josef
6,990 franz	1,625 armee	4,882 august
6,732 gries	1,507 gries	4,251 karl
6,182 josef	1,472 bozner	4,071 johann
6,017 lage	1,434 firma	3,389 grund
5,991 innsbruck	1,239 deutschen	3,340 anton
5,333 wältherplatz	1,232 roten	3,226 alois
5,283 kaiser	1,141 kreuz	2,698 maria

5.3.2 *Outputs*

For each of the five initial query term, this is the percentage of correctness and variation of results with the table above. *Correctness* has been established by simply confronting the most frequent word found in each subset with the algorithm output. One cluster label is considered correct if it contains at least one of the most inversely frequent word. On the other side, the *variation* value establishes the differences between the five requests themselves and is represented by a value ranging from 0.0 to 1.0. As explained later, variation of the responses revealed to be a key factor in the choice of implementation.

q	Term Vectors	STC	Lingo
Bozen	100%	90%	65%
Meran	100%	82%	70%
Tirol	100%	86%	65%
Franz	100%	89%	75%
Gries	100%	81%	70%
Variation	**0.1**	**0.2**	**0.6**

Here is reported one actual output of ten main clusters, when searching for *Bozen*:

Term Vectors	STC	Lingo
Bozen	Kaiser	Bozen Wältherplatz
Bozner	Bozen	Meraner
Wien	Gries	Gotthard Ferrari
Franz	Josef	Bozen Gries
Frau	Wien	Laubengasse
Gries	Innsbruck	Oberau
Josef	Freitag	Museumstraße
Innsbruck	Bozner	Eppan
Freitag	Kaltern	St Josef
Kaiser	Franz	Franz Hofer

From the Term Vectors output it can be evinced that the terms found are the one expected, that is, the most inversely frequent words. What also stands out from this solution, even if correct, is that it has no *actual value* for the user. Term Vectors will, in fact, be more or less the same for each query term *q*, because they are bound to the subset of documents by the *tf-idf (see Paragraph 4.3 – Term Vectors)*.

STC revealed to produce a more various result set, due to the merging at the lowest level of the Suffix Tree structure. However, the variation of the output is still pretty low, thus, it will remain more or less homogeneous in any combination of the query term. This is because the algorithm is operating on a multivalued list of disconnected words instead of full text sentences. It would be far more accurate when parsing sentences, instead of single words, close to *q* (*see Paragraph 4.4.1 – STC)*, as suggested also from the algorithm's creators (Osinski, 2003).

More surprising is the output of Lingo: due to the Latent Semantic Index technique it is able to abstract new possible search results, still strictly related to the query term, but not only by frequency, but also by *meaning*. While it agglomerates similar clusters, it also creates new ones from related words such as *Bozen Wältherplatz* or *Bozen Gries*.

Conclusions and Future Work

Tests proved the superior performance of Terms Vectors merging over clustering algorithms, given the particular input and architecture. What is not taken into consideration in the tests though is the *data transmission time* and *computational memory* necessary on the Tessmann server instance, required to merge the *tf-idf* of each document Entity. This necessary operation cannot be avoided, neither computed by the Solr instance. For this reason, together with the repetitiveness of the results seen previously, TV will not be the solution deployed.

As for STC, the same reasoning may be applied. Even though the algorithm proven high scalability, I suggest the use of Suffix Tree Clustering only when parsing small but full-text sentences in a large database.
The only appropriate candidate for this task remains Lingo: even if this solution is not the most performant, it remains sufficient to meet the user non-functional requirements. Additionally, it has revealed to be more useful to the user, when providing new related terms. It does not only find associated terms, but gives the user new search starting point and suggestions, which can be considered an added feature for the user of a virtual library and the main goal of this paper.

At this point, the reader might still question the relevance of the results given by cluster analysis on full pages. Even though terms outputted by Lingo stimulate a further and more advanced research, it remains bounded to the content and analysis of an entire page of a newspaper and still approximates the relation with the query term. What the user might want to find out are terms located at the level of single articles, instead of the entire page.
At the moment, this is not achievable due to the current level of ALTO files layout and content analysis, but in a later stage of development this will possible thanks to a new tagging system, which will provide ALTO files with distinguished tags per article. The same implementation of Lingo could be used then to focus the related terms retrieval to single articles and titles, for a better user experience and more accurate results.

References

Alto - Technical Metadata for Optical Character Recognition. (2014). Retrieved from The Library of Congress: http://www.loc.gov/standards/alto/

Estivill-Castro, V. (2002). *Why so many clustering algorithms - a Position Paper*. Callaghan: University of Newcastle.

Michael W. Berry, S. T. (1995). *Using Linear Algebra for Intelligent Information Retrieval.* SIAM Rev.

OPATCH. (n.d.). Retrieved from Eurac Research: http://www.eurac.edu/en/research/projects/Pages/projectdetails.aspx?pid=11263

Osinski, S. (2003). *An Algorithm for clustering of web search results.* Poland: Poznań University of Technology.

Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets.* Cambridge University Press.

Selberg, E. W. (1999). *Towards Comprehensive Web Search. Doctoral Dissertation*. University of Washington.

Sorlj. (2013). Retrieved from Apache Solr Wiki: https://wiki.apache.org/solr/Solrj