

Mitigating the Bias of Retrieval Systems by Corpus Splitting

An Evaluation in the Patent Retrieval Domain

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Information & Knowledge Management

eingereicht von

Elisabeth Weigl

Matrikelnummer 0305104

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber

Wien, 22.09.2011

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Elisabeth Weigl
Leystraße 115-117/4/21, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasserin)

Acknowledgements

A lot of people helped and supported me in making this thesis. First of all I would like to thank my advisor ao.Univ.Prof. Dr.techn. Andreas Rauber for his professional support and guidance during this work, even though I often had questions when he was boarding a plane. I also thank the people at the IRF because without them I would not have been able to do most of this work. Special thanks goes to Florina Piroi and Mihai Lupu for helping me in engineering and evaluation difficulties. And to Linda Andersson: You helped me to improve my English through talking and presenting, assisted me during the making and gave critical suggestions and you always could make me laugh about something even though the thesis did not get along. Thank you! I also like to thank Shariq Bashir for his data on the low and high split because without it this work would be lacking important content.

Während der langen Zeit, die diese Arbeit für mich in Anspruch genommen hat, haben mich viele Freunde moralisch unterstützt und mir geholfen, bei klarem Kopf zu bleiben. Euch allen Danke dafür! Ein spezieller Dank geht an Claudia, mir der ich mich in unzähligen Gesprächen über die Diplomarbeit auslassen konnte, an Karin&Matthias, die mich mit englischen Büchern versorgt haben, an Babsi, die mir näher gebracht hat, dass kurze Erklärungen wichtig für das Verständnis sein können, an Carina, die mir dabei geholfen hat, meine physische Gesundheit beizubehalten, an Jiradet, der mir den nützlichen Tipp gegeben hat, schriftliche Aufzeichnungen meiner Experimente zu machen, und natürlich Luki, ohne den ich niemals auf die Idee gekommen wäre ein technisches Studium anzufangen, das mich dazu befähigt die für mich interessantesten Themen der Welt zu behandeln.

Meiner Familie danke ich für die vielfältige Unterstützung und Hilfe während all der Ausbildungsjahre. Ohne euch wäre nichts von alledem möglich gewesen!

Und ohne Nav, der mich während der Arbeit in guten Zeiten ausgehalten genauso wie er mir in schlechten Zeiten beigestanden und mich aufgemuntert hat, der dabei bestimmt genausoviel Arbeit mit mir hatte wie ich mit der Diplomarbeit, hätte ich diese Arbeit nicht fertig gebracht. Danke dir für deine Liebe, deine moralische Unterstützung und deine Ermutigung während all dieser Monate!

Abstract

Typical information retrieval systems retrieve a low number of documents that are preferably close to the query. In contrast to that stands the patent domain as a recall oriented field where missing one single document in the patentability process can lead to costly law suits afterwards if a granted patent is invalidated. However, research showed that retrieval engines cannot find certain documents because they show a bias towards other document characteristics. Thus the goal of this work is to look further into one approach that deals with retrievability of documents and splits a single corpus in two corpora, one containing high, the other low findable documents. For this, the experimental setup has to be provided and the split done again. Afterwards merging strategies that combine the low and high result sets in different ways are tested with the presumption that low retrievable documents are now higher ranked and thus improve recall. This is tested with several models of three different retrieval engines, namely Terrier, Lemur and Solr. Evaluation shows that in most cases the models do not seem to be suitable for this merging, regarding recall and MAP values. Only precision at high rank seems to improve in general. The few models that perform better and which attributes make them more suitable are explained.

Zusammenfassung

Üblicherweise suchen Information Retrieval Systeme eine kleine Anzahl an Dokumenten, die möglichst genau der Suche entsprechen. Die Patentdomäne hingegen ist auf hohen Recall angewiesen und darf kein einziges relevantes Dokument im Prozess der Patentanmeldung übersehen, da die Folge teure Gerichtsverfahren sein können wenn die Bewilligung für ein Patent später ungültig wird. Es hat sich jedoch herausgestellt, dass einige Dokumente von Retrievalsystemen, die eine Präferenz zu bestimmten Dokumenten aufweisen, gar nicht gefunden werden können. Das Ziel dieser Arbeit ist daher, die Herangehensweise einer anderen Publikation, die sich mit der Auffindbarkeit (retrievability) von Dokumenten beschäftigt, weiterzuentwickeln. Die Basis dafür ist das Teilen eines einzigen Dokumentenkorporus in zwei Korpora, wobei einer davon ausschließlich gut auffindbare, der andere hingegen ausschließlich schlecht auffindbare Dokumente enthält. Dazu muss zunächst sowohl der Versuchsaufbau als auch die Korpusteilung neu gemacht werden. Danach werden verschiedene Kombinationsstrategien, die die Resultate der schlecht und gut auffindbaren Korpora verbinden, unter der Annahme getestet, dass die vormals schlecht auffindbaren Dokumente nun einen besseren Rang erhalten und dadurch den Recall verbessern. Dieses wird mit mehreren Retrievalmodellen dreier Retrievalsysteme (Terrier, Lemur, Solr) getestet. Das Ergebnis zeigt, dass sich die meisten Retrievalmodelle nicht gut für diese Kombination eignen, vor allem bezogen auf MAP und Recall Werte. Nur die Precision auf hohen Rängen verbessert sich bei fast allen Retrievalmodellen. Einige der Modelle allerdings liefern bessere Ergebnisse als andere, weshalb die Eigenschaften, die sie geeigneter dafür machen, aufgezeigt und diskutiert werden.

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Goal of work	1
1.3. Outline	2
2. Fundamentals	4
2.1. Intellectual property and patents	4
2.1.1. Intellectual Property	4
2.1.2. Patents	5
2.1.3. European patents	10
2.1.4. US patents	12
2.2. Information retrieval	15
2.2.1. Establishing an index	16
2.2.2. Scoring and term weighting	18
2.2.3. Retrieval models	19
2.2.4. Query expansion	22
2.2.5. Evaluation	22
2.3. Patent search	26
2.3.1. Types of patent searches	26
2.3.2. Techniques	28
2.4. Toolkits	29
2.4.1. Lemur	30
2.4.2. Terrier	31
2.4.3. Solr	34
2.5. Corpora split and result set merging	37
2.5.1. Improving retrievability by corpus partitioning	37
2.5.2. Result set merging including different retrieval engines	40
3. Related work	43
3.1. Categorization of methods	43
3.1.1. Accessibility, findability, retrievability	43
3.1.2. Prior art search	46
3.1.3. Retrievability and precision	47
3.2. Assessment of methods	47

4. Methods	49
4.1. System set-up	49
4.1.1. Hardware and software versions	49
4.1.2. Creation of the test collection	50
4.1.3. Terrier	55
4.1.4. Lemur	57
4.1.5. Solr	59
4.1.6. Baseline	61
4.2. Corpora split	63
4.3. Analysis of split corpora	64
4.4. Result set merging	69
4.4.1. CombineTrecResults	69
4.4.2. Methods of CombineTrecResults	71
5. Experimental results	74
5.1. Overview	74
5.2. Normalization	75
5.3. equalSize method	76
5.4. highestSimilarity method	76
5.5. partitionSize method and optimalMap	77
5.5.1. Comparison to baseline and different splits	77
5.5.2. Comparison to eqS and highSim	81
6. Conclusions	83
6.1. Discussion and interpretation of results	83
6.2. Future prospects	88
A. Listings	89
B. Tables	95

1. Introduction

1.1. Motivation

Patents are a way to keep possession of an idea. There are almost no limitations to what kind of idea the protection is expanded, however the industries with major interests can be found in life science technologies and engineering. The vast amount of patents gives them enormous economic value: Over 50 million patents exist worldwide [40] and every year the number increases. From 2007 to 2008 1.91 million (2.8%) more patents were filed [55]. From a technological point of view this huge amount of data is a challenge of its own. For every newly filed patent the patentability process starts over, including different kinds of patent search. These searches have to be done efficiently and effectively which includes not missing one single relevant document that could invalidate a granted patent afterwards because this usually leads to costly law suits. Thus, information retrieval that commonly tries to reduce the amount of results and therefore sometimes intentionally sorts out relevant data, is required to look at this problem from another point of view and provide different approaches.

With its very particular demands concerning relevant documents, the patent domain was found to be recall oriented, which means that in contrast to a common internet search engine that sometimes sorts out relevant data in order to provide the best results within the first few hits, the number of relevant documents within the result set is more important than having a compressed result set. Of course it is also necessary that the relevant results are not the last documents in the ranking but a patent searcher is thought to be more willing to proceed further down the result list than a usual internet surfer. To not lose sight of the position in the result set, other measures like the average precision are also common for information retrieval in the patent domain.

1.2. Goal of work

The goal of this work is to look further into a very specific field within the patent retrieval. It was found out that retrieval engines showed a bias towards particular documents because of certain document features. The affected systems tend to return documents with these attributes more often than other documents.

Based on that, an approach was made by Bashir and Rauber [11] that divided a document collection in low and high retrievable documents, which were classified low and high based on a retrievability measure proposed by Azzopardi [5], and afterwards merge it again with the presumption that poor findable documents are now ranked higher and thus lead to a lower bias. In [11] this merge was made with a single index and two

methods, one combining the low and high results in an equal ratio, the other combining them in the ratio of their distribution in the original collection. This approach leaves open a lot of other combining strategies and deals only with the bias of IR systems, not with the recall or MAP values.

Another approach dealing with different retrieval engines in the patent domain is the work of Zenz et. al [59]. There, topics and relevance assessments were made from scratch and the result sets of particular retrieval models of different retrieval engines, that worked with independently created indexes, were combined to improve recall and MAP values.

The exact goal of this work lies on one hand in combining the two approaches described, which includes working with independently indexed low and high corpora in contrast to Bashir and Rauber. On the other hand different merging strategies will be developed and compared to the methods proposed in [11].

1.3. Outline

The work at hand provides background information on the topic in chapter 2. The first part of this is separated in three logical sections. First, an overview of intellectual property and patents, including European and US patents and information about the patentability process, can be found in section 2.1. The next section 2.2 contains information retrieval theory necessary for the experiments made, including information about the retrieval models used and evaluation measures such as recall and MAP. Then follows the consequence from the patent and IR section, patent search in section 2.3, which gives an overview of different search types. Afterwards, in section 2.4, the retrieval engines used are introduced. Section 2.5 presents the contents of the main papers this work builds on.

Chapter 3 contains the basic content of other scientific work that is related to this topic. This includes section 3.1, giving an overview of methods in other work that were used here also and concludes in section 3.2 that briefly explains which part of the methods will be used afterwards.

In chapter 4 the methods used are explained in detail. This includes information on work that had to be done previous to the experiments. How the experimental setup was built can be found in section 4.1. Next, section 4.2 provides information on the split in low and high corpora. Afterwards, the split corpora are analyzed regarding their relevant document distribution in section 4.3. Finally the result set merging section 4.4 combines them again and describes the different merging strategies.

Chapter 5 contains detailed descriptions on the results of different merging strategies. This includes an overview of the methods and models used in section 5.1. Section 5.2 briefly shows the best performing normalization method, which is then used in the merging methods. The first merging method described in 5.3 is equalSize, which combines an equal number of low and high results. Next, in section 5.4 the results of the highest-Similarity method, that takes the best performing 1000 results, is explained. Finally the values of the partitionSize method, that combines a selectable number of results from

each split, are discussed and compared to the other methods in section 5.5.

Eventually the results of the previous chapter are discussed in chapter 6. Details on certain methods and models are examined in section 6.1 and section 6.2 argues what can be concluded of this result and which methods could be further looked into.

2. Fundamentals

The following chapter is dedicated to provide an overview of the fields that are necessary to improve retrievability of patent documents. Therefore it is divided in several sections that together build the background of this work’s topic. First, section 2.1 and 2.2 provide the most fundamental insight into the patent domain and respectively IR. The former describes the broad field of intellectual property and narrows down on one sub-area, patents. European and US patents are emphasized especially because these are used in the practical chapter. The latter section explains basics of IR like Boolean Retrieval, term weighting and evaluation in this field as well as certain IR methods like query expansion and language models or the BM25 model for they will be used later on. The patent search section 2.3 summarizes the two preceding parts from the patent domain’s view by describing the kind of patent searches, when they are used and how they are issued. In section 2.4 the retrieval systems that will be used in the work are presented and described. The last section 2.5 summarizes the works on which this thesis is built.

2.1. Intellectual property and patents

If nature has made any one thing less susceptible than all others of exclusive property, it is the action of the thinking power called an idea, which an individual may exclusively possess as long as he keeps it to himself; but the moment it is divulged, it forces itself into the possession of every one, and the receiver cannot dispossess himself of it.

Thomas Jefferson

2.1.1. Intellectual Property

A way of keeping possession of an idea taxed the brain of a lot of people a long time ago. Thomas Jefferson was instrumental in crafting patent laws that made it possible for creative people to maintain their intellectual ownership. Nowadays, uniqueness and idiosyncratic prospect to own an idea are still the foundations of intellectual property (sometimes abbreviated as IP). The WIPO (World Intellectual Property Organization) defines intellectual property as “the legal rights which result from intellectual activity in the industrial, scientific, literary and artistic fields” [53]. These legal rights make patents marketable good for they can be sold, mortgaged or willed to another. This potential high value can lead to contention and discussions, like it happened with the controversy about software patents in the 2000s. The law around intellectual property helps creators and producers of intellectual goods and services. It gives them the opportunity to make

up for their effort, like research, development or marketing, by giving them exclusive control for a certain time period.

Intellectual property can be divided into different branches. The WIPO [53] distinguishes two parts, *industrial property* and *copyright*. They include certain areas:

- *Industrial property*:
 - inventions in all fields of human endeavor
 - industrial designs
 - trademarks, service marks and commercial names and designations
 - protection against unfair competition
- *Copyright*:
 - literary, artistic and scientific works
 - performances of performing artists, phonograms, broadcasts (related to copyright)

The last area of industrial property, protection against unfair competition, can be considered of belonging to that branch because the *Paris Convention* (see section 2.1.2) included a similar concept. When summarizing the areas, industrial property can be seen as new solutions to technical problems (*inventions*), aesthetic creations determining the appearance of industrial products (*industrial designs*) and transmitting information to consumers (*trademarks, commercial names*), where the main focus is on protecting signs that are likely to mislead a consumer. [53]

The last area, *scientific discoveries*, are not represented in these branches for they are by definition no inventions, they are “the recognition of phenomena, properties or laws of the material universe not hitherto recognized and capable of verification” [53]. Because inventions build on said properties or laws that are - maybe - already recognized, the two concepts are not the same.

2.1.2. Patents

A patent has to describe the solution to a problem within a technological field and can therefore provide the owner of the patent the authorization to exploitation, like manufacturing, usage or selling. It does not give the proprietor a right to make, use or sell anything, it only grants that no other person in the country that granted the patent exploits the invention. This is done to compensate for the owner’s development costs and to enable him to achieve profits without the risk of competitors. In order to obtain a granted patent, its patentability has to be given which means that certain requirements have to be fulfilled.

Patentability

According to [54], patentability of an invention is given if it conforms to four consecutive parts:

- **Patentable Subject Matter:** In general every invention in the technological scope is a patentable subject matter, unless it falls into certain fields, like discovering materials or substances that already exist in nature, scientific theories and mathematical methods or methods of treatment for humans or animals (except for products used in these methods, this could be patentable) amongst others. TRIPS (see section 2.1.2) also excluded patents if their commercial exploitation would conflict public order or morality.
- **Industrial Applicability (Utility):** A patentable invention has to have a practical purpose, it must not be purely theoretical. This intention is also emphasized by *industrial applicability*, which reflects that it has to be possible to make and manufacture as well as carry out and use the invention in practice. There are countries where utility and not industrial applicability is required.
- **Novelty:** A solution to a problem is new as long as nobody else had the idea first. Though, this fundamental element of patentability can never be proved, only its absence can be shown. An important term here is *prior art*, which refers to all knowledge that exists until a certain date. This knowledge can be a published writing or other publication, it can be an oral disclosure like publicly spoken words, or it can be a *disclosure by use*, which means the publicly usage of the invention. It is important that only disclosure that is explicitly contained in the publication destroys the novelty of an invention.
- **Inventive Step (Non-Obviousness):** This part confirms that the invention “would [not] have been obvious to a person having ordinary skill in the art” [54], the obviousness referring to a part of the prior art which a non-expert on the field could have concluded. This part, as distinguished from *novelty* which searches for any difference between invention and prior art, asks for an inventive step that depends on novelty in the first place. The inventive step is denied if the combination and the choice of combined elements is obvious.
- **Disclosure of the Invention:** In the patent application the invention must be disclosed in a way that it can be carried out by a person skilled in the art. In order to give others the chance of filing oppositions, the content of the application has to be made publicly available (in a journal or gazette).

Filing and examination of a patent application

The process of filing a patent begins on the applicant’s side, by drafting a patent application. This starts by identifying the intended invention, which involves deciding whether the patentability requirements are met. It is also essential to determine critical features for the future claims and to see if these features can be altered or substituted and still lead to the same result.

After this the content of the application has to be determined. This content has to be filled in mainly three sections (see also sections 2.1.3 and 2.1.4 for details on European and US patents):

- an abstract,
- a description and
- claims.

The abstract should not be too long and is used for a short summary of the other two components. In general an application typically relates to one invention, which has to be sufficiently disclosed in the description. This element usually also contains explanations to included drawings. The claims part is the most important section of every application because claims are relevant for the scope of the protection the patent eventually has to provide.

Usually three kinds of examination are performed before a patent can be granted. The exact execution of them is left to the patent office, but to provide that a patent fulfills certain requirements three steps are done:

- *examination as to form*: ascertain that the above mentioned sections (and/or other sections) are included in the application
- *search*: the search for prior art in the invention's specific field is conducted
- *examination as to substance*: ascertain that the conditions of patentability are fulfilled

If the examination processes find that the application fulfills all necessary requirements, the corresponding Patent Office will grant a patent on the application. This includes that details, which do not include technical information, of the granted patent are entered into a Patent Register. In countries that require fee payments, details about paid fees are also recorded in the Register. In an Official Gazette a reference to the grant of the patent is published and the applicant receives a *Certificate of Grant* that establishes his ownership of the patent. [54]

International Patent Classification - IPC

With the Strasbourg Agreement 1971, the first edition of the *International Patent Classification* or IPC was established. Until today it provides a hierarchical system of language independent symbols to classify patents as well as utility models into different technological areas. The major objective is to obtain an internationally uniform patent document classification in order to establish an effective search tool for patent document retrieval. This directly relates to the search for patentability (cf. section 2.1.2), where novelty, the inventive step and non-obviousness can be discovered more easily. At the topmost level of hierarchy the IPC divides technology into eight *sections*, each of it designated by a capital letter (A-H), with e.g. *human necessities* being A or *electricity* being H. There can also be subsections to these sections. The next level, *class*, is denoted with a two-digit number and a title, e.g. *H01 Basic electronic elements*. Every class has on its subsequent level one or more *subclasses*. This adds another capital letter to the code, e.g.

H01S Devices using stimulated emission. The subclasses are broken into subdivisions, groups, that can be main groups or subgroups. Main groups add a sequence like *3/00* and subgroups determine the digits behind the slash. Both also add titles, which ends in a construct as in listing 2.1, in which the title of *3/14* is read as *Lasers characterised by the material used as the active medium*. An overview over the IPC hierarchy is provided in figure 2.1. [57]

Listing 2.1: IPC example

H01S	3/00	Lasers
H01S	3/14	* characterised by the material used as the active medium

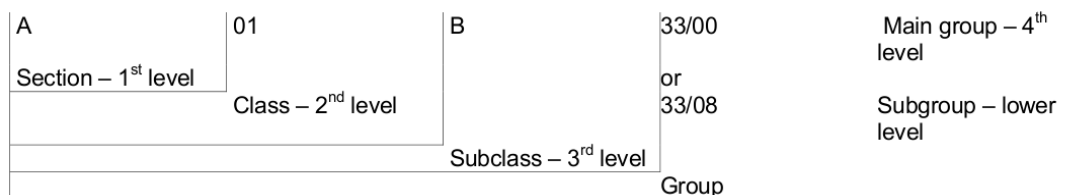


Figure 2.1.: IPC hierarchy [57]

An extension to the IPC is provided with the ECLA (European Classification System), which is used by the EPO (see section 2.1.3). It uses more than twice as much subclasses as the IPC (140,000 compared to 70,000). ECLA follows the rule of the IPC and in most cases they are identical to the IPC subgroups. From there it extends the classification by another capital letter that can also have another digit as successor, e.g. A61C1/05B1, which can again have another capital letter as successor. [17]

The USPTO also developed an intern classification system, the USPC (United States Patent Classification), which is only used for classifying US patents. It is no extension of the IPC and follows another format, which makes it important to lead one format into another. When converting between IPC and USPC there exists concordance tables, which however sometimes lack accuracy and completeness. The USPC uses about 163,000 subclasses. [22]

The problem of locally different patent classification systems has lead to a trilateral harmonization project enforced by the three biggest patent organizations worldwide: EPO, USPTO and *Japan Patent Office* (JPO). It was set up in 1983 and continues to find a way to standardize the patent classification.¹

Patent harmonization

In general a patent is only enforceable within the jurisdiction of the state it was filed in. In order to provide more internationality to this localization different arrangements were made between certain countries. The most important are as follows [24]:

¹www.trilateral.net

- **The Paris Convention**, which was held and signed in 1883 [51], has since been adopted by every industrialized nation (173 countries) except for Taiwan [50]. It provides an inventor from any signatory country the possibility to file an application first in his home country and subsequently within one year to file a corresponding application in any other signatory country. The filing date of the foreign country is the same as in the home country, which is a benefit regarding prior art issues.
- **The Patent Cooperation Treaty PCT** was signed in 1970 and is adopted by 142 countries as of 1 December 2010 [56]. It enables an inventor to file an *international* patent application with which he seeks patent protection for his invention in each signatory country simultaneously. The application can be filed by any national or resident of the contracting country. The patent prosecution is subsequently carried out as an *international search* which checks all available published documents of the contracting countries. Eventually, if the international application is not withdrawn the application gets published by the International Bureau. [52]
- **The Trade-Related Intellectual Property Rights (TRIPS)** was signed in Morocco on 15 April 1994 and is administered by the *World Trade Organization* (WTO). Until this agreement the US patent term was 17 years from the date of its issuance, which got extended to 20 years from the priority date. TRIPS focuses on not building boundaries for legal trading as well as to ensure reasonable protection of intellectual property at the same time [58].
- **The American Inventors Protection Act of 1999 (AIPA)** most important content states that, in general, any application that is filed since December 2000 has to be published and publicly available 18 months after the filing.

Patent family

According to [29] there are four different definitions of what a patent family refers to. Three of them take into account the priority dates of their databases when building families. They build families out of equivalent patents, links between two patents (extended families) or base the relationship on the first filing. The fourth definition defines families that are validated by experts. The problem with narrowing down a precise terminology is that patent database providers define what a patent family is in their own database.

The EPO [18] speaks of a patent family if there is more than one application or publication of one invention with the same priority date in another country. This group of patents is related to each other by common priority numbers. In view of kind codes (cf. section 2.1.3), patent relation is formed by documents with the same numerical identifier but a different kind code (A or B) [21]. The USPTO defines a patent family similar to the EPO: the same invention, patented in more than one country.

2.1.3. European patents

In 1973, 20 states met at a diplomatic conference in Munich in order to develop a patent grant procedure for European patents. 16 of the participants signed the resulting *European Patent Convention* (EPC) which came into force four years later. The first patent applications were filed a year later, on 1 June 1978. With this the *European Patent Organization* (EPOrg) fulfilled its intention as a public international organization and later succeeded an EPO (*European Patent Office*) branch in The Hague as well as sub-offices in Berlin, Vienna and Brussels. Though it has no legal boundaries to the *European Union* as of today, additionally to the 27 EU member states eleven other states (in total 38) belong to the EPOrg, amongst others Liechtenstein, Switzerland or Turkey. [16]

The EPC establishes a system of law for the grant of patents for invention, which are called *European patents*. These shall be handled at the same conditions as a patent granted in the national scope of a signatory country, unless the EP provides otherwise. It is therefore possible to request the grant for one or more States ([19], Art. 1,2,3).

According to the EPC, the EPOrg shall have administrative and financial autonomy. The Organization's organs are divided into:


- the *European Patent Office* (EPO) and
- the *Administrative Council*.

The organization's task is to grant European patents (see figure 2.2 for an example), which shall be carried out by the EPO with the supervision of the Administrative Council ([19], Art. 4).

The EPO accepts applications in one of the three official languages (English, French, German) or in another language if it is translated into one of the three. The EPC provides for several sections and divisions for processing an application or patent ([19], Art. 15 ff.).

The EPC also defines patentable inventions, which follow the rules of patentability as in section 2.1.2. Regarding the first point, *patentable subject matter*, it additionally clearly excludes - to the extent that the application relates to such subject-matter or activities as such - aesthetic creations, programs for computers and presentation of information ([19], Art 52). Also inventions that risk the *ordre public* or morality are excluded. As far as the other patentability requirements are concerned, the EPC goes along with the WIPO publication.

A European patent application can be filed at the EPO or, if possible by law, in the central industrial property office of a State (Art. 75), by a natural or legal person as well as anybody equivalent to that. It is also possible to apply with multiple applicants ([19], Art. 58, 59). The right to a patent usually belongs to the inventor, but if the inventor is an employee this right is determined according to the law of the State of the employee's employment. If more than one person made the same invention independently of each other, the right to the patent belongs to the person with the earliest filing date (cf. section 2.1.4 for *first-to-invent* principle). ([19], Art. 60)

(19) 

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
05.01.2011 Bulletin 2011/01

(21) Application number: 10186273.8

(22) Date of filing: 22.04.2005


(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR
HU IE IS IT LI LT LU MC NL PL PT RO SE SI SK TR

(30) Priority: 22.04.2004 NL 1026007

(62) Document number(s) of the earlier application(s) in
accordance with Art. 76 EPC:
05736373.1 / 1 740 691

(71) Applicant: **Campina Nederland Holding B.V.**
5301 LB Zaltbommel (NL)

(72) Inventors:
• van Dijk, Johannes Henricus
5409 TN, Odiaapeel (NL)

(11)  **EP 2 270 125 A2**

(51) Int. Cl.:
C12G 3/04 (2006.07)

(74) Representative: **Smulders, Pauline Elisabeth Antoinette**
4011 JW, Zoelen (NL)

Nederlandsch Octrooibureau
J. W. Frislaan 13
2517 JS Den Haag (NL)

Remarks:
This application was filed on 01-10-2010 as a
divisional application to the application mentioned
under INID code 62.

(54) **Stable mildly acidic alcoholic milk- and/or soy protein-based drink**

(57) The present invention relates to a stable alcoholic milk and/or soy protein-based drink with a pH in the range from 5.2-6.5, comprising at least one or more milk and/or soy proteins, alcohol and one or more stabilizers. Furthermore, the invention relates to a method for preparing a stable alcoholic milk and/or soy protein-based drink with a pH in the range from 5.2-6.5, comprising as constituents at least one or more milk and/or soy proteins, alcohol and one or more stabilizers, which method comprises the following steps: a) mixing at least a portion of the milk and/or soy proteins and at least a portion of the stabilizers in an aqueous medium to obtain a mixture; b) setting the pH of the mixture to 3.5-4.4; c) homogenizing the mixture obtained in step b) to obtain a homogenate; d) setting the pH of the homogenate to 5.2-6.5; and e) mixing in the alcohol and optionally a residual portion of the milk and/or soy proteins and of the stabilizers during or after one or more of steps a)-d).

EP 2 270 125 A2

Printed by Jouve, 75001 PARIS (FR)

Figure 2.2.: Example of a European patent

According to Art. 63 of the EPC the term of a European patent is 20 years beginning at the filing date of the application. This period can be extended by a Contracting State under certain circumstances, like there has been a state of war in that State for a while or if there has to be some administrative authorization procedure before marketing the invention. Art. 64 states that a European patent shall provide the same rights as a national patent in the States that it was granted for. An infringement of such a patent has to be dealt with by national law. Art. 69 declares that a European patent's scope of protection is determined by the claims. Description and drawings in the application are used for their interpretation.

Regarding the requirements of a proper European patent application, it shall contain (Art. 78):

- a request for the grant of a European patent
- a description of the invention

- at least one claim (which has to be supported by the description (Art. 84); claims will be used in the experiments, see section 4.1.2)
- drawings to which the description or claims refer to
- an abstract (only for technical information, not for scope of protection)

This article also states that, if the filing or search fees are not paid in time, the application will be deemed to be withdrawn. This also happens if the annual renewal fees are not paid.

Kind codes

When a patent application is in the patenting process it changes its status several times. Each time it will be published again under another patent id in order to make its status clear. The patent applications get a numerical identifier followed by an extension, the kind code, that refers to the status. **A** documents are applications, published 18 months after filing with the EPO or 18 months after priority date; **B** documents are patent specifications. There are codes that are used more often than others, which are (listed by frequency [21]):

A1 application published with search report

B1 patent specification (granted patent)

A3 publication of search report

A2 application without search report

B2 patent specification with amendments

There are also *corrected A documents* **A8** and **A9**, which can be the correction of the title page or a complete reprint and according to that *corrected B documents* **B8** and **B9**, as well as the supplementary correction codes **W1** and **W2**, which indicate the first and respectively the second corrected version of a patent document. [15]

2.1.4. US patents

The US patent system goes back to the government of George Washington, when in 1790 the first patent laws were enacted. Even then the problem of exclusive property over an idea was known and so the United States patent system was established. In 1952, due to various changes, the Congress amended a new patent act. Based upon this amendment, which also included the requirement of an invention not being suggested itself by state of the art [30], until today the *Copyright and Patent Clause* empowers Congress “To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries”². In order

² *The Constitution of the United States*, Article 1, Section 8, Clause 8.

to use this right, the inventor has to provide a significant description, which describes the way to make and use his invention, to the USPTO (United States Patent and Trademark Office). In return a granted patent gives the inventor or patent owner exclusive rights for a limited time.

In case of colliding filings of patents, if the same invention gets filed twice in a short time, the US patent system has an inference procedure in which it has to be determined who the first inventor was (cf. section 2.1.3). This is directly associated with the *first-to-invent* system used by the US patent system [7]. On the contrary to most other countries, who use a *first-to-file* system where the first person who files the patent application gets the right to the grant, the right to the grant goes to the person who invented it in the first place³. Usually the patent applicant, who is often referred to as the assignee in the process of filing a patent, is the same as the inventor of the patent, who is the owner by right [24]. Though it happens that the invention was done by an employee in his working time, in which case the rights of the patent go over to the employer.

According to Title 35 of the United States Code (35 U.S.C.) §154(a)(2), the term of a US patent is usually 20 years from the priority date (the earliest filing date), provided that the maintenance fees are paid on time. This can be extended, e.g. if there are delays in the issuing of the patent (*Adjustment of Patent Term, Extension of Patent Term*), by at most five years [24]. The priority date also plays a vital role regarding patentability. When doing a prior art search during the patenting process, every publication before this date counts as prior art.

Sections of a patent

According to section 2.1.2, a granted US patent consists of several sections, of which the most important are:

- *Front page*: The front page contains the bibliographic data of the patent. It includes the patent title, filing and grant date, name of inventor, patent owner, priority data and the numbers and filing dates of related patent applications. It also contains classes and subclasses that the patent office assigned to the document as well as the (sub)classes the examiner searched in. Also other patents and non-patent literature that was cited as prior art is printed. An example of the front page can be found in figure 2.3.
- *Abstract*: Summarizes the invention briefly.
- *Specification*: A long description of the invention. Here it is described how a person of ordinary skill in the art is able to make and use the invention without undue burden.
- *Claims*: The most important part of the document (used in the experiments later, see section 4.1.2) because it defines the scope of protection provided by the patent.

³In January 2011 a *Patent Reform Act* was presented in the US Senate that proposes to use the first-to-file principle in the US patent system, too [47].

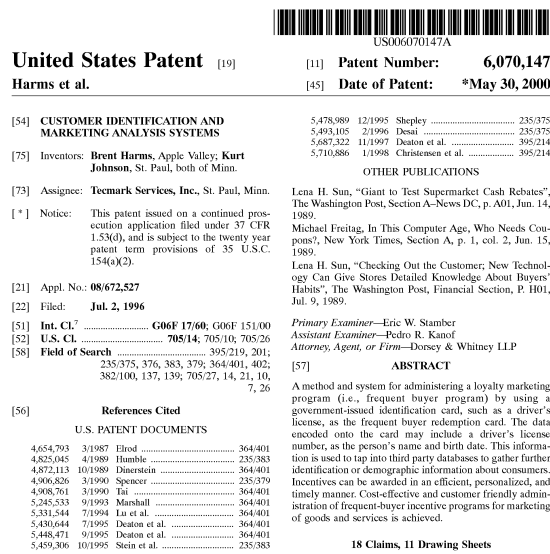


Figure 2.3.: Example of a US patent's front page

In a granted patent only claims allowed by the examiner are provided, other (filed) claims can be found in the official patent file history. An examiner does not always allow all claims because “although the claims are interpreted in the light of the specification, limitations from the specification are not read into the claims” [24]. An example of the claims section of a patent can be found in figure 2.4.

- *Drawings*: The drawings section provides illustrations for more details of the claimed invention.
- *List of Cited References*: The applicant as well as the examiner cite other patents or nonpatent documents. An examiner's citations are marked with an asterisk * on the granted patent. The most important references can not be found on the patent. These are the citations considered by the examiner and they only show up in the official patent file history.

<p style="text-align: center;">13</p> <p>information by periodically electronically retrieving updated additional, non-transactional government information about the consumer from the government information database; and</p> <p>(g) analyzing the marketing data record in the consumer database to determine whether the consumer meets predefined criterion for a promotional action.</p> <p>2. A method for collecting and analyzing information relating to consumer purchases and assembling marketing data relating to the consumer, where the method comprises the acts of:</p> <p>(a) capturing purchase information about a consumer's purchase of goods or services, including information about the goods or services purchased;</p> <p>(b) reading the consumer's government-issued identification card to acquire basic identification information about the consumer at the time of purchase, wherein the basic identification information includes an identification number that is stored on the government-issued identification card;</p> <p>(c) acquiring the consumer's address by electronically retrieving the consumer's address from an address database, wherein the address database is maintained by the government and is indexed by the identification number read from the consumer's government-issued identification card, and wherein the government requires the consumer to keep an up-to-date address in the address database;</p> <p>(d) electronically assembling a marketing data record by combining the purchase information and the basic identification information with the address acquired in act (c);</p> <p>(e) storing the marketing data record in a consumer transaction database;</p> <p>(f) periodically updating the marketing data record in the consumer transaction database by again electronically retrieving the consumer's address from the address database and updating the consumer's address in the marketing data record if it has changed; and</p> <p>(g) analyzing the marketing data record to determine whether the consumer meets predefined criterion for a promotional action based on the consumer's purchase of goods or services and the consumer's address, wherein the predefined criterion relates to amount or number of purchases, dates of purchases, and products or services purchased.</p> <p>3. The method of claim 2, wherein act (g) includes the acts of:</p> <p>identifying consumers who meet the predefined criterion for a promotional action based on the consumer's purchase of goods or services; and</p> <p>issuing an award to the identified consumer(s).</p> <p>4. The method of claim 2, wherein act (g) includes the acts of:</p> <p>calculating a consumer balance, wherein the consumer balance is a total of previous purchases by the consumer; and</p> <p>notifying the consumer, at the time of purchase, of the consumer's balance.</p> <p>5. The method of claim 2, further comprising the act of:</p> <p>(b) acquiring demographic information for the consumer by retrieving demographic data from a database, wherein the demographic information is retrieved based on information in the consumer's marketing data record.</p> <p>6. The method of claim 5, wherein the demographic data is geodemographic data that is retrieved from the database based on the consumer's address in the marketing data record.</p>	<p style="text-align: center;">14</p> <p>7. A system for collecting and analyzing information relating to consumer purchases and assembling marketing data relating to the consumer, comprising:</p> <p>(a) transaction equipment that acquires and captures purchase information about a consumer's purchase of goods or services and basic identification information about the consumer, wherein the purchase information includes information about the goods or services purchased, and the basic identification information is taken from the consumer's government-issued identification card;</p> <p>(b) a network connected to the transaction equipment; and</p> <p>(c) a processing system that is connected or connectable to the network, wherein the processing system comprises a computer, a consumer database, a government information database, and a loyalty marketing database, wherein the processing system retrieves additional, non-transactional government information about the consumer from the government information database, wherein the government information database is indexed by the basic identification information taken from the consumer's government-issued identification card, and wherein the consumer is required by the government to maintain up-to-date information in the government information database;</p> <p>stores a marketing data record in a consumer transaction database, wherein the marketing data record is a combination of the purchase information, the basic identification information, and the additional, non-transactional government information;</p> <p>periodically updates the marketing data record with updated additional, non-transactional government information by periodically retrieving updated additional, non-transactional government information about the consumer from the government information database and updating any additional, non-transactional government information that has changed; and</p> <p>analyzes the marketing data record in the consumer transaction database to determine whether the consumer meets predefined criterion for a promotional action, wherein results thereof may be used for the promotional action.</p> <p>8. The system of claim 7, wherein the processing system further comprises:</p> <p>a demographics database, and wherein the processing system acquires demographic information for the consumer by retrieving demographic data from the demographics database, wherein the demographic information is retrieved based on information in the consumer's marketing data record.</p> <p>9. The system of claim 8, wherein the demographic database includes geodemographic data that is retrieved from the database based on the consumer's address in the marketing data record.</p> <p>10. A method comprising the acts of:</p> <p>(a) acquiring purchase information about a consumer's purchase of goods or services, wherein the purchase information is collected at the point of sale and includes a description of the goods or services purchased and the date of sale;</p> <p>(b) acquiring basic identification information about the consumer by reading the consumer's government-issued identification card at the point of sale, wherein the basic identification information includes an identification number and a date of birth for the consumer, both of which are stored on the government-issued identification card;</p>
--	--

Figure 2.4.: Example of US patent's claims

2.2. Information retrieval

But do you know that, although I have kept the diary [on a phonograph] for months past, it never once struck me how I was going to find any particular part of it in case I wanted to look it up?

Dr Seward, Bram Stoker's Dracula, 1897

The context in which *information retrieval* is used over time has changed with the rapid development of computer networks. A definition given in an introductory book seems to only refer to professional searchers: “Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)” [28]. Today, many people who use an electronic device often engage in the task of searching for information

and getting it by a web search engine or simply their email program's inherent search routine when looking for a specific mail. Professionals like patent searcher use search tools for instance to obtain information on prior art. Therefore IR can be found and is developed in many areas.

2.2.1. Establishing an index

For a machine with enough computing power it is easy to execute a query for a keyword by grepping through text. This linear scan though is only appropriate for a certain amount of information. Once this threshold is exceeded other methods have to be considered. One way to do so is to index the text prior to searching. The result is a set of indexed terms. In order to store the terms efficiently they are saved in an *inverted index*, which is composed of a *dictionary* that holds all terms of the text and the *postings*. Single postings cite the documents the term appears in and are stored in a postings lists, which taken together are referred to as the *postings*. Figure 2.5 illustrates this circumstance.

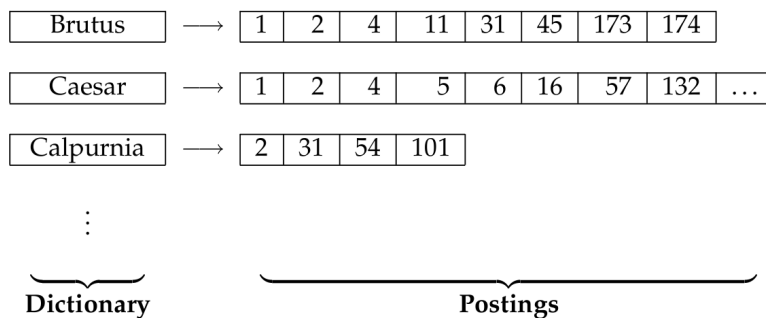


Figure 2.5.: Inverted index represented by its dictionary and postings [28]

To get an inverted index, four steps have to be done:

1. All documents that have to be indexed are collected.
2. Each document is broken down into tokens.
3. All tokens from the last step are normalized, the results are the indexing terms.
4. The documents each term occurs in are indexed. The result is an inverted index.

Tokens and token normalization

The second step involves chopped up pieces of a sentence, *tokens*. A token is defined as “an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing”[28]. Every token that contains the same character sequence is of the same *type*. *Terms* are types that are included in the IR system's dictionary. When doing the tokenization step there can be various problems

an IR system has to face. A language identification is one of the basic sub-steps in order to provide the right tokenization. Certain domains also require specific attention: The text in an IT magazine might contain tokens like *C++* or *person@domain.com* which have to be recognized correctly. Hyphens are a special challenge as well. They are handled as a classification problem or by heuristic rules. In languages like German compound nouns are written without a space between the words, so a compound splitter has to be implemented as well. Major East Asian Languages require a pre-processing that performs a word segmentation before indexing.

A sub-step of the tokenization can be seen when selecting which words are worth adding to the vocabulary and which are not. Extremely common words like *to* or *and* are denoted as *stop words*. In general these words are determined by sorting all terms by their *collection frequency*, adding the most frequent terms to a *stop list* and when indexing, excluding all words of the list from indexing. This significantly helps to decrease the number of postings to store for a system, but it also increases the lack of results when doing a phrasal search. The mentioned stopword *and* for instance can be important when issuing the query *divide and conquer*. By stopping out this word, the meaning of the algorithm design paradigm is lost. Because modern IR systems often do not have storage or processing time problems, frequently stop lists are not used.

In the third step, the *token normalization*, differences in a token's character sequence are analyzed in order to assign the token to other tokens with the same meaning. Usually, equivalent classes are created for this, which hold all tokens that are similar to each other. This can help the user of the IR system when issuing a query for a term that e.g. has different dictions, like *nonpatent* or *non-patent*. Also words with accents can be assigned to the same words without the accents, or case-folding can turn all capitalized words into their equivalent lower case variants.

In this term finding step there also exists the process of *stemming* and *lemmatization*. Both of them are used to deal with different forms of the same words, for instance if a verb is conjugated or different words with similar meanings are used, like *democratic* or *democracy*. In cases like these stemming simply chops off the ends of words to produce a term for the indexer. This can increase the recall during retrieval, but it will also harm precision. Lemmatization on the other hand removes the inflectional endings and returns the base or dictionary form of the word, which is called the *lemma*. Nevertheless stemming can impair terms, it is commonly used because the Natural Language Processing lemmatizer tools often do not increase retrieval performance notably.

Problems with phrasal search were mentioned above. In order to be able to deal with phrasal queries, an IR system has to look for phrases as soon as it indexes the text. One approach is to see every pair of consecutive terms as a phrase, which is called a *biword index*. This method can also be extended to more words than only two and called *phrase index* if the number of words is variable. A more commonly used technique is a *positional index*, which stores for every token a document id together with the position numbers of the specific token in the document. Some IR systems also combine the strategies of biword and positional indexes. [28]

Indexing

The indexing itself, step four, is usually done by the indexer of the IR system at hand. It builds the index, respectively the inverted index, by storing all terms in an applicable way so that it can later be accessed by the retrieval system more or less efficiently.

When the indexing is done, the system can provide relevant documents for a certain information need of the user. This need can be expressed by a query that gets issued to the retrieval system. If dealing with a boolean retrieval system, a query can be made out of terms combined with the boolean operators *AND*, *OR* and *NOT*. To measure the effectiveness or quality of a retrieval system, two key statistics are used in IR: Recall and Precision (see section 2.2.5). [28]

2.2.2. Scoring and term weighting

An indexable document can contain more information than only its text. Digital writings include metadata, which has fields like document format or date of creation. These individual fields can also be indexed separately in parametric indexes. Doing this can extend a query by e.g. searching for certain words in a document with specific properties. A more wider approach would be to index zones, which are user-defined text areas, in independent inverted indexes. With this and (*weighted*) *zone scoring*, which is sometimes called *ranked Boolean retrieval*, retrieval can be done by giving certain zones scores from zero to one (if they do not include the query terms or do so).

Continuative from zone scoring, a text can include zones that mention a term more often than others. These zones are more important when querying for the term, so the terms get a weight for every zone or document they appear in. When the number of occurrences is equal to the weight of a term, this weighting scheme is called *term frequency*. It is denoted as $tf_{t,d}$ and stands for the score between a query term t and a document d . With this, the set of weights of all terms of a document gets more into focus and the ordering of the words is less important. This view of a document is called *bag of words model* and only a term's weight, not the ordering of terms, is important. A much broader approach to term frequency would be the *collection frequency* which is the total number of occurrences of a term in the collection. This can be useful because a collection about a particular subject-matter is likely to contain many topic-specific words in each document. If that happens, the term weight tf could be reduced according to the frequency of that term in the collection.

A more common approach is to use the *document frequency* df_t that denotes the number of documents in the collection the term t appears in. But the more often a term is included, the less its weighting should be. For this the *inverse document frequency* was defined as:

$$idf_t = \log \frac{N}{df_t} \quad (2.1)$$

with N being the total number of documents in a collection. By combining the methods of term and document frequency, a weighting scheme is produced that assigns

- high values to terms that occur often in a small number of documents,

- lower values to terms that occur fewer times in one document or occur in many documents
- low values to terms that occur in almost all documents.

This weighting scheme, *tf-idf*, is denoted as

$$tf-idf_{t,d} = tf_{t,d} \times idf_t \quad (2.2)$$

2.2.3. Retrieval models

Boolean Retrieval

The Boolean retrieval model is the oldest and most widely used approach in IR. Queries are formed out of tokens that are combined with boolean expressions (*AND*, *OR*, *NOT*). Each document on which this query is issued is seen as a set of words, so the result mainly depends on occurrences of the query terms in the document. The importance of each word is not measured, so a word is either in the text (1) or not (0).

Vector Space Model

Opposed to the Boolean retrieval paradigm the *Vector Space Model* (VSM) represents documents as vectors in a common vector space. This document vector can have several dimensions, each of which corresponds to one term t of the dictionary. Each coefficient is a value representing the term importance or presence by e.g. a simple binary code (0,1) or a term weight w_{ij} (the weight for term i in document j) like *tf-idf*, which is the most common representation. The set of documents of a collection can then be viewed as a set of vectors in a vector space. Comparing them with each other may be used when implementing a *more like this* function that finds other documents like the found one (see section 2.5.2 for an example usage). Along the lines of this, a query can also be seen as a vector in the same vector space. A document d or query q is expressed as t -dimensional vectors, an example is given in equation 2.3.

$$d_j = (w_{1j}, w_{2j}, \dots, w_{tj}) \quad (2.3)$$

The VSM makes it possible to represent a collection of N documents and M terms in a $N \times M$ *term-document matrix*. An entry in the matrix corresponds to the weight of a term in the specific document, e.g. 0 stands for no significance or non-existence.

With the documents and query lying in the same vector space, their similarity can be calculated by the angle between them, as in figure 2.6 illustrated. The most common similarity measure for this is the *cosine similarity*:

$$score(q, d) = \frac{q \cdot d}{|q| \cdot |d|} \quad (2.4)$$

The denominator has the effect of normalizing the length of different documents so that a longer document cannot have a better result than a shorter just because it contains more terms.

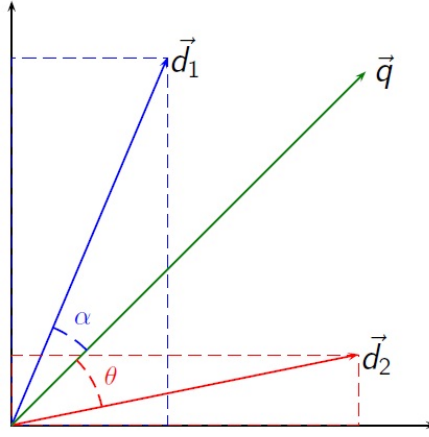


Figure 2.6.: Query q is more similar to \vec{d}_1 than to \vec{d}_2 [38]

The result of this calculation can further be used as a score for the relevance of this document for the specific query. Usually a setting for a retrieval includes the document collection represented by vectors, a free text query represented by a vector and additionally a number K that is equal to the number of results with the highest scores that are sought.

The VSM contains a drawback when it comes to combining Boolean queries with vector space queries. A VSM can be used for issued Boolean queries because the system just has to make sure the weight of the query terms is not zero in the document. On the other side, a Boolean index can not be used to answer queries for a VSM because it does not store a term weight information. This could be solved mathematically but there exists no system that uses this approach. Regarding wildcard queries, the two methods would basically need different indexes. Only on the lowest level it works to issue a query to a VSM, where multiple terms are generated and issued as a vector space query. The scoring would depend on the weights of the generated terms. Phrase queries are even more difficult. In a VSM the ordering of the terms is lost and cannot be regained to help finding consecutive terms. [28]

Okapi BM25

The *Okapi* system first implemented a non-binary IR model, the *BM25 weighting scheme* or often called after the system *Okapi weighting*. This model will be used by two retrieval engines in the experiments, see section 4.1.6. It was developed to provide a probabilistic model considering term frequency and document length, as addition to the *Binary Independence Model*. It builds upon the sum of the inverse document frequency for query terms:

$$RSV_d = \sum_{t \in q} \log \frac{N}{df_t} \quad (2.5)$$

RSV is the *Retrieval Status Value*, the quantity used for ranking, q are the query terms. There are several equations used for the BM25, one of the most common is:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{df_t} \right] \cdot \frac{(k_1 + 1) \cdot tf_{td}}{k_1 \cdot (1 - b + b \times (\frac{L_d}{L_{ave}})) + tf_{td}} \quad (2.6)$$

where L_d is the document length and L_{ave} the average document length of the collection, k_1 is a tuning parameter for the document term frequency scaling ($k_1 = 0$ denotes a binary model, $k_1 > 0$ denotes raw term frequency usage), b is a tuning parameter determining the scaling by document length ($0 \leq b \leq 1$; $b = 1$ denotes fully scaling the term weight by document length, $b = 0$ denotes no length normalization). [28]

Language models

Usually a user thinks of fitting words that describe his information need as a query. When issuing this query, the terms of it are compared to document terms and the documents with the highest count are marked as more important. Language models, which are also common in other language technologies like speech recognition or machine translation, have the opposite approach to this task: Each document and its terms are described by a document model which has a probability of generating a query. A document then matches an issued query if this document model's query is similar to it. For this, the language modeling approach ranks documents based on the probability of the model M_d generating the query: $P(q|M_d)$. In order to do this, probabilities of the words of the language have to be determined. The language model is therefore a function that puts a probability measure over strings from a specific vocabulary. For a language model M over an alphabet Σ the accumulated probabilities of the vocabulary are 1:

$$\sum_{s \in \Sigma^*} P(s) = 1 \quad (2.7)$$

The probabilities of the terms are calculated by different methods. The simplest method for this is the *probability chain rule*, that takes into account the probabilities of earlier events. More common forms for language models in IR are the *unigram language model*:

$$P_{uni}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2)P(t_3)P(t_4) \quad (2.8)$$

which does not consider other conditions, or the more complex *bigram language model*, that takes into account only the previous term:

$$P_{bi}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3) \quad (2.9)$$

These models are often sufficient for IR tasks, opposed to other fields like speech recognition for instance, because the structure of sentences are not that important to analyze. For this reason these methods are also called *bag of words models*.

Language models can be realized in various ways. The most prominent method is the *query likelihood model*. This model constructs for each document d a model M_d with

the goal to rank the document by $P(d|q)$ (cf. equation 2.10) and the probability of each document is interpreted as the likelihood of it to be relevant for the query.

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \quad (2.10)$$

Because $P(q)$ is the same for all documents and $P(d)$ is treated as uniform for all documents, they can be ignored, which sets $P(d|q) = P(q|d)$. A query would be observed as a random sample from the respective document model, and documents are ranked according to this probability. For doing this, the *multinomial unigram language model* is used:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{tf_{t,d}} \quad (2.11)$$

with $K_q = \frac{L_d!}{tf_{t_1,d}!tf_{t_2,d}!\dots tf_{t_M,d}!}$ as the multinomial coefficient for query q , which can be ignored because it is a constant depending on the query. Certain retrieval models run in the experiments of this work also implement the language model function, see section 4.1.6. [28]

2.2.4. Query expansion

Synonyms of words pose a problem for most IR systems when it comes to recall of the returned documents. Often user vary the queries themselves to cope with this issue, but there are also a number of methods that can be done by the system to help refine the query. A local method would be *relevance feedback* that actively involves the user in the retrieval process by telling the system what is relevant or not, which subsequently lets this information alter the final result set. A global method like *query expansion* (QE) acts independently of the query or the results when expanding or reformulating the query terms. It focuses on new queries that produce semantically similar terms in the result set. So users can give input on specific query words or phrases, not on a result. A common approach for this is using some form of thesaurus. The thesaurus can suggest synonyms for the user to select or automatically be used to expand the query with synonyms and related words for each term t in the query. This method can also be used with term weighting, where an additional term has less weight than the original one. The thesaurus could be manually composed by human editors or automatically derived by word co-occurrences or grammatical relations. The advantages of using a thesaurus are that it does not require user input and that it increases recall. The downside is that in a scientific area a good thesaurus is costly to produce and update, because it has to be domain specific. It also decreases the precision especially when it comes to ambiguous terms. [28]

2.2.5. Evaluation

To evaluate a retrieval system is a very important part of IR because it helps research to continue its work. Nevertheless it is also a very tough subject because it involves having standards to compare to and these standards have to be created first. What builds the

basis for the evaluation is how satisfied a user is with the results the system provides. Important factors for this user happiness are:

- speed of response of the system,
- size of the index and
- relevance of results.

The latter one definitely is the most important factor for most users of a system. Therefore IR systems are tested with standard test collections which include three items used to evaluate it:

- A document collection,
- predefined queries for information needs and
- a set of relevance judgments, that determine which document is (non)relevant for which information need.

This last point, relevance judgments, is a binary classification referred to as the *gold standard* or *ground truth judgment of relevance*. These judgments do not refer to a query, but to the information need that stands behind a query. This circumstance reflects that a relevant document is not one that happens to contain all words of a query, but one that conforms to an information need. In order to fulfill the first two requirements there exist standard test collections. The most common collections are:

- **The Cranfield collection:** As the first collection of this kind, it was collected in the UK in the late 1950s. It contains 1398 abstracts about aerodynamics, 225 queries and exhaustive relevance judgments.
- **Text Retrieval Conference (TREC):** TREC was started in 1992 by the National Institute of Standards and Technology (NIST) and U.S. Department of Defense as a workshop. It is divided in different *tracks*, which are dedicated to different research areas like the TREC-CHEM organized by the IRF⁴. It was developed to provide evaluation for large-scale text retrieval issues. To receive an evaluation, NIST provides a test set consisting of documents and questions with which the TREC participants run their IR systems. The retrieved top-ranked documents are returned as a list to NIST, who then evaluates the result. For the retrieval research community it is also possible to obtain the evaluation software to do their own checks of relevance judgment to information needs (which are called *topics* in TREC) mappings. Because the collection is much too large to provide exhaustive relevance judgments, the judgments are only available for documents among the top k results that were returned by a system the evaluation was developed for. [36]

⁴<http://www.ir-facility.org/trec-chem>

- **GOV2**: Evaluations of the GOV2 web page collection were done by NIST as well, although the size of this collection is much larger. Nevertheless it is still smaller than the amount of documents indexed by large web search companies.
- **NTCIR and CLEF**: The NII Test Collections for IR Systems and the Cross Language Evaluation Forum were designed for cross-language information retrieval, whereas the first one focuses on East Asian languages and the latter one on European languages.

Unranked retrieval sets

When an IR system returns a set of unranked documents for a query, precision and recall of this result set is usually interesting to the researcher. These can be derived building on the basic relations from the following table:

	<i>relevant</i>	<i>nonrelevant</i>
<i>retrieved</i>	true positives (tp)	false positives (fp)
<i>not retrieved</i>	false negatives (fn)	true negatives (tn)

Recall is the fraction of relevant documents in the collection that the system returned, whereas Precision is the fraction of returned results that are relevant to the information need (see equations 2.12 and 2.13).

$$\text{Recall} = \frac{tp}{tp + fn} \quad (2.12)$$

$$\text{Precision} = \frac{tp}{tp + fp} \quad (2.13)$$

Another measure that stands for the correct classification of a system is *accuracy* that only evaluates the true positives and negatives: $A = \frac{tp+tn}{tp+fp+fn+tn}$. A reason why this measure is usually not used within the IR domain is given by the fact that the maximum accuracy can be obtained by labeling all documents as nonrelevant and because usually there is only a very small fraction of documents that are relevant to a query, the accuracy of the system would shoot up. However, a user is generally more interested in getting back documents, which maybe includes receiving some false positives, than getting back none at all. The values for recall and precision can also help to improve the results. High precision may be needed in the web domain, where a user wants to have relevant results on the first result page. In the patent domain, however, a high recall is much more important because missing one relevant document can lead to costly law suits. The two measures trade off against one another, because a recall of 1 can be achieved by returning all documents of the collection with the downside of having a low precision. High precision on the other side includes leaving out some relevant documents as well. To cope with this problem, the *F measure* is used to combine the two quantities by calculating the weighted harmonic mean of precision and recall:

$$F_\beta = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \quad (2.14)$$

Generally, the *balanced F measure* (F_1 or $F_{\beta=1}$) is used, in which precision and recall are equally weighted. If $\beta < 1$, precision is emphasized, if it is greater than 1 recall is enlarged.

Ranked retrieval sets

Nowadays, the standard of search engines is using ranked retrieval results. A naturally given example of this result set can be found in the top k retrieved documents. A very common measure among the TREC community is the *Mean Average Precision* (MAP), which calculates a measure of quality over all recall levels. For this, the average precision of each query is computed by calculating the precision for each retrieved relevant document, then averaged with the previous calculations and repeated until all relevant documents are retrieved. The mean of these averages is then computed for each system, which results in the MAP:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} P(R_{jk}) \quad (2.15)$$

$q_j \in Q$ are information needs, $\{d_1, \dots, d_{m_j}\}$ is the set of relevant documents for the information need and R_{jk} is the set of ranked retrieval results from the top result before getting to document d_k . Using the MAP measure with the same information needs in different systems makes more sense than using it with different information needs in only one system. Then the system can be evaluated in contrast to others, while the MAP score tends to vary widely when used in one system with various information needs. Another measure commonly used calculates precision at certain levels of retrieved results. This *precision at k* (sometimes also *precision@k*) called measure is used when it is important to know how well the system works at the first k results, like the first few pages of a web search result. Its downside is that it is the least stable commonly used evaluation measure and that it cannot take into account the total number of relevant documents when calculating the precision. The *R-precision* soothes this problem by calculating the precision at position R , which is the number of relevant documents for the information need. It is identical to the *break-even point* measure and empirical experiments showed its high correlation with MAP.

Similar to *precision at k* is the *recall at k* measure, which will be used together with *precision at k* in the comparison of the results in chapter 4. It calculates the recall values at a specific threshold and is used to give an overview of the progression of the recall values.

Relevance assessment

As said above, the proper evaluation of an IR system is done by using relevance judgments of specific information needs and see how the system at hand performs compared to others. For small test collections these judgments are provided exhaustively, but for large modern collections usually relevance is only assessed for a subset. One approach

that uses this method is *pooling*, where the top k results of some IR systems are evaluated for relevance. Sometimes the result documents of Boolean keyword searches or expert assessments are also added to the subset. A relevance judgment is done by humans who are deciding if a document is relevant for an information need or not. Because humans tend to be not as reliable when it comes to consistent decisions as a computer would do, there are different approaches to this problem. One could be using many judgments for each information need, which can involve very high expenses. The *kappa statistic* offers a mathematical procedure for calculating the agreement between two (or more) judges. It is copied from the social sciences and is a measure considering agreement by chance:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \quad (2.16)$$

$P(A)$ stands for the probability of agreement, $P(E)$ denotes the probability the judges agree by chance (in a relevant/nonrelevant classification it would be 0.5, but can also be calculated using marginal statistics). The κ value for a total agreement will be 1, for random agreement 0 and a $\kappa < 0$ stands for worse than random. So a realistic $\kappa > 0.8$ stands for good agreement, whereas $\kappa < 0.67$ indicates that the data is problematic for evaluation. [28]

2.3. Patent search

Patent searching is a very broad term, it includes various domains. Two distinct domains can be discerned here. One domain are the reasons why a patent search is conducted in the first place. There are different types of patent searches that are distinguished by reasons for the search or the searcher that executes the search. The second domain are the techniques with which a searcher looks for patent documents.

2.3.1. Types of patent searches

The patent domain has various use cases for conducting a search. Not only searches a patent examiner for other patents or literature during the patentability process, there are also other reasons for conducting a patent search. The reason for a search can come up during the development of a product or after a patent has been granted.

Patentability search

Patentability means that a patent application has to fulfill certain requirements in order to conclude as a granted patent (cf. section 2.1.2). Thus a patentability search shall help a patent attorney or agent prior to filing a patent to determine if the invention can be patented and which other documents, be it other patents or non-patent literature, may be relevant for the evaluation. The search also helps the editor of the application to construct claims. Any document, written or otherwise published, patent or not, prior to the filing date of an application can be relevant to the search. Therefore this search is one of the most time consuming searches because the searcher himself has to determine when

to end it. If only locale patents are searched, the patent owner may have legal issues to deal with later, but because of the limitlessness of possible important documents the searcher will not be able to evaluate all documents. [24]

Validity search

(In-)Validity searches are used to determine the novelty at the time of an invention. It is comparable to a patentability search except that it is conducted after a patent application's publication. Common reasons for an invalidity search can be found when a company is sued for patent infringement. If the company conducts this search and finds evidence that important references were not considered in the patenting process, the patent in question might be invalid and therefore unenforceable. This is also important regarding licensing issues. If a patent is invalid, a licensee might have already paid for a license that another company can freely use because of the invalidity. Also the patent owner might be interested in a validity search to assure the enforceability of his own patent, if he registers a patent infringement. Such a search includes the search of the claims or drawings as well as any patent and non-patent literature prior to the earliest claimed priority date of the patent. [24]

Infringement search

An infringement or *freedom to operate* search is used to determine if there exists a patent that includes patent claims that would be violated by producing something of commercial interest [24, 20]. An example of usage would be a company that wants to assure their right of making, using and selling a new product with respect to a potential patent infringement. The searcher needs to search only unexpired and therefore enforceable patents. Compared with other searches (cf. figure 2.7) it is crucial to not oversee only one patent claim, therefore the expected recall in this search is 1 [20]. A common use case of infringement searches in the US, where the *first-to-invent* concept is used, is applied by companies that monitor the patent filing of their competitors closely and file a similar patent with exactly the same claims, confident that they were the first to come up with the invention in question.

A *clearance search* can be seen as a subpart of the infringement search. It is used to determine if a party has clearance to make, use and sell the product of a patented invention. The objective is to find out what inventions exist in the public domain and are free to use. This search will be conducted when a patent has not been infringed or expired. It can be seen as the *freedom to operate* part of infringement search. [24]

State-of-the-Art search

A state-of-the-art search is a comprehensive search of patent and non-patent publications, nevertheless if it is expired. It is usually conducted by people who want to get an overview of the current development in a certain field. Also patent attorneys use them to help with patentability questions. A more comprehensive search in this area is

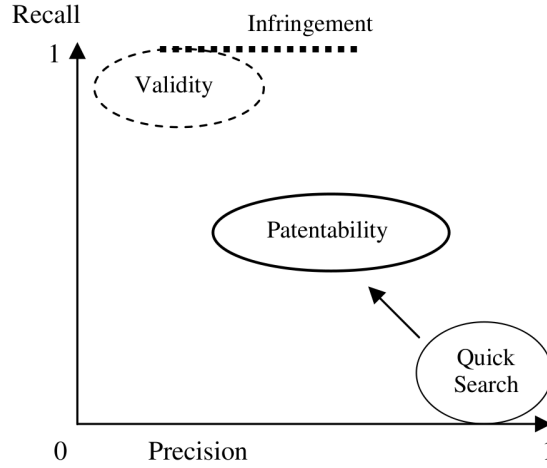


Figure 2.7.: Different patent searches and their required (P,R)[20]

the *patent landscape search*. It extends the state-of-the-art search by categorizing fundamental discoveries and incremental improvements or gathering information about the history of a technology. This search is usually conducted to analyze if making business in a research area is beneficial. [24]

2.3.2. Techniques

When patent documents were not digitized, a patent search was the same as a patent classification search where only patents from one class were examined. However, a common problem was the misclassification of various patents which resulted in not findable patents. With the digital era, searching for certain documents got a lot easier for patent examiners. It was possible to conduct text searches with certain keywords and even citation searching was feasible. Today the recommendation for conducting a proper search is to combine text and classification searching.

Classification search

The classification search was developed out of what was already there: a patent classification system (see 2.1.2 for the IPC classification). This search can be divided into two subsections. The first one, the *core classification search*, leads to the core features of the invention. The search area for this examination (*core class area*) is represented by the classes and subclasses that are most closely related to the invention's subject matter. The subclasses of this area contain the most relevant patent documents and have to be searched exhaustively. The second section, the *peripheral classification search*, includes areas that deal with sub-features or sub-functions of the invention at hand. This section is not as important as the first one, it can be searched depending on time and budget of the patent examiner. [24]

Full-text search

In general, text search is done by entering keywords into a search engine to conduct a search. Though, there should be more to do this to result in a satisfactory output. As keyword searching is always a way of broadening and narrowing different strategies have been developed. A patent examiner could start with the generic structure/function and then narrow down the problem by combining text queries gradually. This can also be done the other way round, by first starting with the problem and narrowing down the structure/function. The third common option is to start tightly with problem, structure and function combined and then subtract keywords to broaden the output. [24]

Citation search

Citation searching is a search that includes previous searches by the patent applicant or the examiner. It therefore can help to understand the interpretation of the examiner and which other documents were relevant at that time. In turn this can assist with finding new perspectives on an invention. There exist two different types of citations. The first one, *backward citations*, were issued during the patent examination process and are citations to already granted patent documents and publications in the scope of the invention. *Forward citations*, the second type, are their counterparts: They are patent documents and publications that are subsequently cited by other granted patents (see figure 2.8 for a more descriptive explanation). [24]

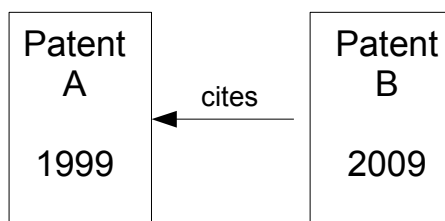


Figure 2.8.: Patent A is a backward citation of patent B, B is a forward citation of A.

2.4. Toolkits

For this project three different indexing tools were used. The first one, Lemur, emphasizes on the construction of an efficient index which comes with the demand for a large disk storage. The second one on the contrary, Terrier, does only need a fraction of disk space and focuses on cross-linguistic indexing. Solr is the third of them and concentrates on high scalability, providing a search server with distributed search and index replication.

2.4.1. Lemur

The Lemur project is well known for the development of the *Lemur Toolkit* and the *Indri Search Engine* among other components. The former is an open source software framework made for support of research in language modeling and information retrieval. The latter one is a subset of the *Lemur Toolkit*, although it is often addressed as a stand-alone tool. This can be explained due to the fact that *Indri* was developed autonomic and later got included into the toolkit⁵.

The main advantages of the *Lemur Toolkit* are therefore its sophisticated structured query languages and the support of XML/HTML documents. As far as indexing is concerned it can handle text in English, Chinese and Arabic, it supports word stemming (Krovetz and Porter), stop words and recognition of acronyms among other features. When doing retrieval a researcher can work with Indri, KL-divergence, vector space, tf.idf, Okapi and InQuery as well as relevance- and pseudo-relevance feedback, passage and XML element retrieval and other components⁶.

Indri, if seen apart from the toolkit in a detailed view, is a search engine that provides state-of-the-art text search. With its index file structure it can handle much larger text collections - up to 50 million documents on a single machine or 500 million with a distributed search - and it is responsible for the support of XML/HTML documents. It also integrates the *Indri query language* which is used to build complex queries about document structure. [44]

When Indri was developed, it was designed to address four goals [39]:

1. complex queries should be supported by the query language
2. superior effectiveness from the retrieval model
3. support retrieval at different levels of granularity
4. support of very large/multiple databases, fast indexing, concurrent indexing, ...

The retrieval model of Indri was designed as an improved version of a model described in [33], which was a combination of language modeling and inference network approaches to information retrieval [39]. In the inference network it is assumed that “a query is composed of a series of concepts, where these concepts may be terms, phrases, or more complex entities” [39] and that a document is relevant for a user if it contains concepts that occur in the query.

The query language of Indri was inherited by its predecessor, the Inquery structured query language⁷. Therefore operators like *#combine* and *#weight* have similar usage as *#sum* and *#wsum* in Inquery, whereas the *#filreq* (filter-require) and *#filrej* (filter-reject) operators are the same. Indri also supports a field operator with which fields, that are tagged information of a document, can be included or excluded in the search for a term. These fields can also be used as regions for scoring, meaning that scoring and

⁵For more information on this topic see www.lemurproject.org/docs/index.php/Lemur_and_Indri

⁶Some of these features were not present in the version used in this work.

⁷For more information see for instance [13]

ranking within a document (for instance in its paragraphs or sentences, depending on the field) is possible. Indri is also capable of recognizing numeric quantities and comparing them.

The Indri system architecture provides different parsers for documents formatted with e.g. XML, HTML or in TREC-style. With these parsers every document gets translated into a *ParsedDocument*, that can be stored by the indexer directly. With this, a list of terms of the document, their position as well as field information get saved. This *ParsedDocument* can get transformed into another one by *Transformation objects*, which are e.g. stemmers or a stopwords removal. These Transformation objects can also be concatenated. As far as indexing is concerned, Indri builds compressed inverted lists for each term and field in memory and writes it to disk. This data is self-contained, which means that all information that is necessary to perform a query is contained. Indri's retrieval feature includes the evaluation of queries against many indexes simultaneously while these indexes can be on different machines. The query evaluation consists of two steps: first, the number of times a term and phrase occur in the collection are calculated, and second, the results of step one are used to evaluate the query against the collection. Indri also supports multi threaded operations, which allows for query runs during document indexing and index writing.

2.4.2. Terrier

Terrier is an open source information retrieval platform, written in Java and developed at the School of Computing Science at the University of Glasgow. The *UK Engineering and Physical Sciences Research Council* supported and funded a project whose result is the Terrier framework [59]. The platform's main goal is to provide a flexible, efficient and effective search engine for large-scale collections of documents. It supports research using standard TREC and CLEF test collections.

Besides this assistance, Terrier also provides many document weighting models such as DFR (Divergence from Randomness) models, Okapi BM25, language modeling and TF-IDF. The platform can also index common document formats like HTML, PDF, Microsoft Word and others and can be easily extended by creating new *Document plugins*. It is also capable of indexing field and position information and supports various stemming techniques including the Snowball stemmer for European languages. As far as retrieval is concerned it comes with an easy to use command-line querying interface to allow for quick testing and it also provides Query Expansion among other features.[46]

The indexing algorithms of Terrier are based on a new DFR framework for deriving parameter-free probabilistic models for information retrieval [1]. To offer flexibility in usage, the system is built in a modular way and provides convenient configuration options, e.g. the parameter setting of Terrier is done in one *terrier.properties* file. Indexing in general is a four stage process which can be altered at any step by adding plugins. An overview of this indexing architecture is given in figure 2.9.

The indexer plugin manages the whole indexing process. In order to do so the corpus of documents gets processed by a *Collection* plugin which generates a stream of *Document* objects. Then the indexer iterates through all these documents of the collection and

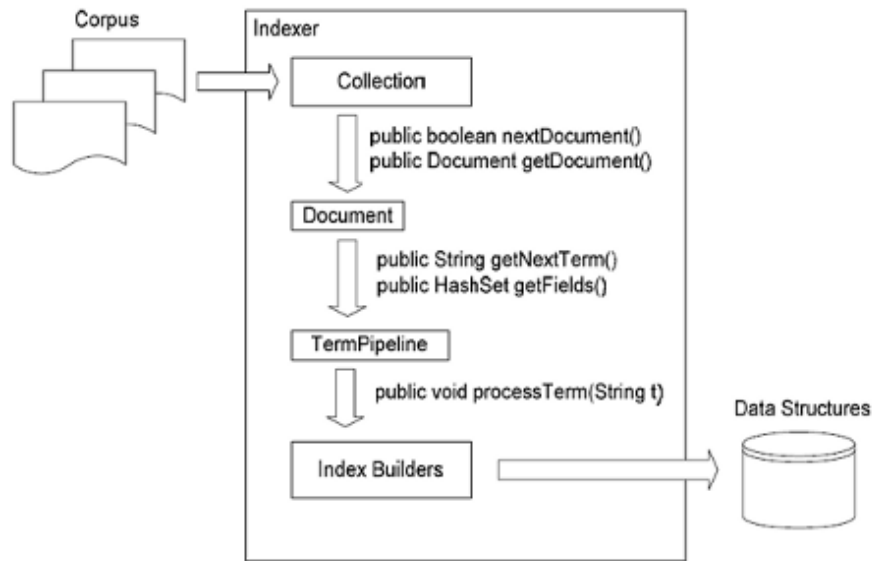


Figure 2.9.: Terrier indexing architecture [46]

sends the terms it finds through one or multiple *TermPipeline* components where the terms can be transformed or removed if they should not be indexed. The implementation of stemmers and stopword removal can be found here.

The terms extracted from a document by Terrier include the textual representation of it, the position in the document and the fields in which it occurs in the document, which is usually a HTML tag but can also be defined by the *Document* plugin. It also provides a *Block Indexer* which allows to save positional information with each term. A block's size is 1 by default, so the exact position gets stored when indexing. If the size is greater than that ($blocksize = N$) the block id is incremented after every N terms. This can be useful for working with semantic entities to allow structured retrieval.

After the *TermPipeline* processing, the terms get aggregated, the indexer writes them to disk and creates a data structure including

- Lexicon,
- Inverted Index,
- Document Index and
- Direct Index.

The *Lexicon* saves the term and the corresponding term id, global term frequency and document frequency as well as the offsets of the postings list in the Inverted Index. This *Inverted Index* is responsible for the postings lists of a term. For each term the document id of the matching document and the term frequency in that document get

stored. The *Document Index* stores document number and id, as well as the length of each document and the offset of it in the *Direct Index*. This *Direct Index* stores terms and term frequencies and is used for query expansion. [46][37]

Figure 2.10 shows an outline of the retrieval architecture used in the Terrier framework. A search application (e.g. Desktop Terrier) sends a query to the Terrier framework. In

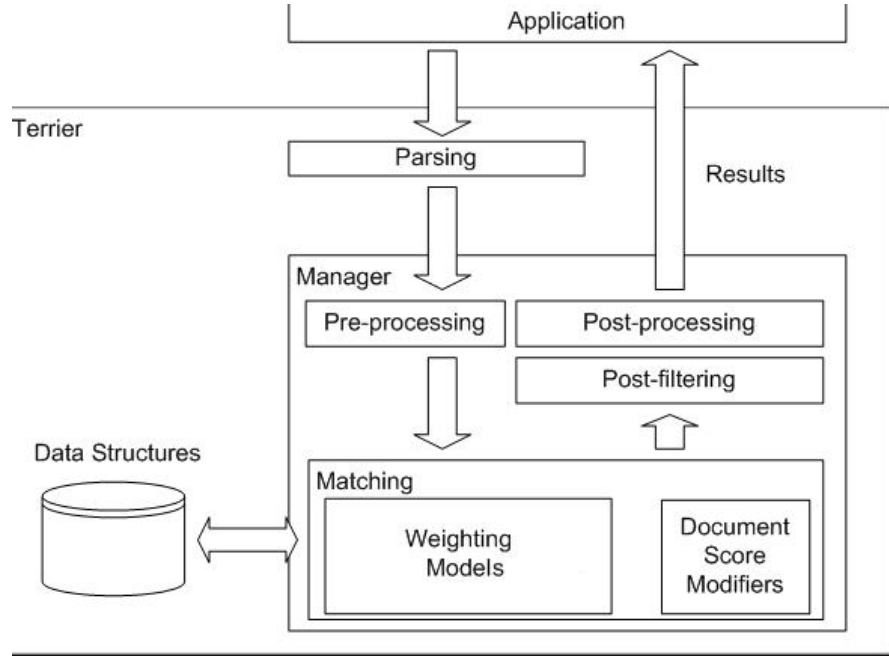


Figure 2.10.: Terrier retrieval architecture [46]

order to deliver a result this query first has to be parsed and the outcome gets handed to the *Manager* component, where it gets applied to the configured *TermPipeline* as a pre-processing step. Then the query gets handed to the *Matching* component at which point document scores are calculated using the combination of weighting model and score modifiers. As said before Terrier provides a wide range of document weighting models of which the Divergence From Randomness models are the noticeable. Terrier’s version 3.0 includes 126 DFR models. Afterwards, in the *Post-processing* the result set can be altered, for instance by invoking query expansion, whereas in the *Post-filtering* step documents can be included or excluded regarding specific restrictions taken by the user. Eventually, in the last step the derived result set gets returned to the executing application.

As said before Terrier includes automatic pseudo-relevance feedback in form of query expansion. For this the top most informative terms from the top-ranked documents in the query are added into the a new query which gets re-weighted and re-run. To tune the inherent parameters of the query expansion the Terrier developers include parameter-free query expansion mechanisms, the Bo1 (Bose-Einstein 1), Bo2 (Bose-Einstein 2) and KL (Kullback-Leibler) term weighting models.

Regarding evaluation, Terrier supports the main evaluation functionalities found in the `trec.eval` tool, for instance calculation of Mean Average Precision (MAP), precision at rank N , interpolated precision or R-precision amongst others. [46][37]

2.4.3. Solr

Solr is an open source enterprise search platform supported by the *Apache Lucene* project and written in Java. Its APIs cover REST-like HTTP/XML and JSON with which the platform can be used from other programming languages. It uses the Lucene search library for full-text indexing and searching and features hit highlighting, faceted search, dynamic clustering, database integration, handling of various document types for indexing and geospatial search. [43]

As Solr builds on Lucene, technical details of the search engine apply to it as well. Figure 2.11 shows typical components of a search application. The darker green shapes highlight the components Lucene handles. As Lucene only provides the core search library, acquisition of content has to be done by Solr, which involves the three lowest shapes. For this, raw content has to be gathered and subsequently translated into documents which eventually can be processed by Lucene.

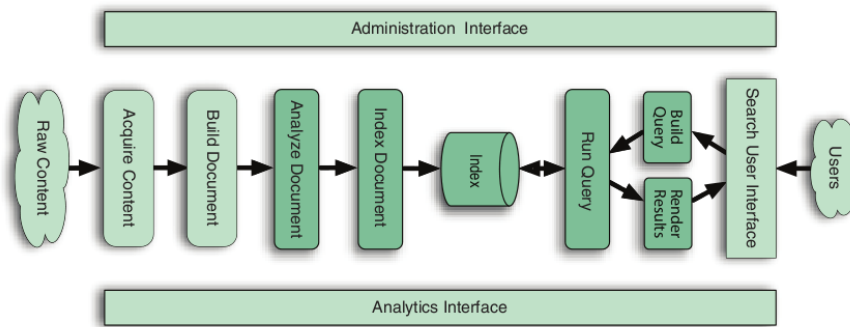


Figure 2.11.: Components of a search application with highlight on Lucene's component[31]

Basically, Lucene generates an index by extracting text, analyze it and add it to the index. In order to extract text from a document, this unit has to fulfill certain requirements. For Lucene a document is a container holding fields which in turn consist of the real content. This content is divided in fields which consist of a name, a text or binary value and detailed options that describe what has to be done with this field when adding a document to the index. Figure 2.12 illustrates this circumstance by giving a Solr example of fields from the Solr *schema.xml* and a sample document. The fields are used in the retrieval process because Lucene searches the values of fields that the user wants to be searched in. This can be useful for instance if only titles are wanted in the result set.

The next step after receiving the content is to analyze it. For this, Lucene provides

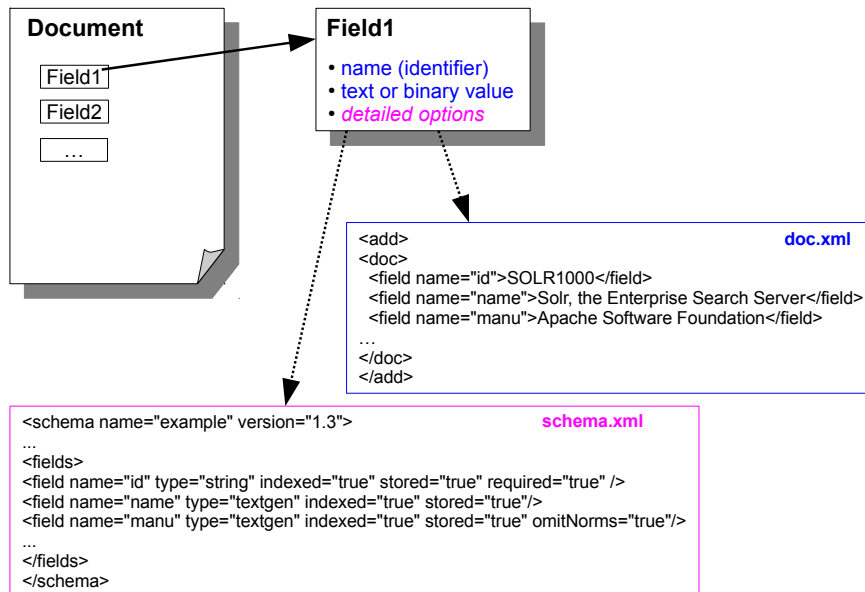


Figure 2.12.: Lucene's content modeling with a Solr example

various analyzers that convert field text into terms by tokenizing and performing different operations on it. This includes normalization, stemming, stopword removal or lemmatization amongst many others. It is also possible to create own analyzers in case the right one is not provided. The stream of tokens the analysis process provides are subsequently written into the files contained in the index. Traditionally Lucene stores input in an inverted index. The index directory consists of segments which are created when a writer flushes buffered added documents and pending deletions into the directory. When a query is issued the results of separately traversed segments are combined. Regarding index functionality, Lucene implements indexing, deleting and, compared to Terrier and Lemur, direct update of documents. [31]

For the retrieval part, the first step is to create a query object. For this Lucene's *QueryParser* uses analyzers to translate a user's textual expression into a user-defined complex query with which the searching process can be issued. Lucene combines the commonly used *pure boolean model* with the *vector space model* (see section 2.2.3) to process an information need. A user can also choose which method is used for each single search.

Regarding scoring, Lucene uses a *similarity scoring formula* to measure the similarity between a query matching document and the query itself. The formula is shown in equation 2.17 and calculates the score for each document d matching term t in query q .

The result of this equation is the raw score, a floating-point number ≥ 0.0 .

$$\sum_{t \text{ in } q} (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{boost}(t.\text{field in } d) \cdot \text{lengthNorm}(t.\text{field in } d)) \cdot \text{coord}(q, d) \cdot \text{queryNorm}(q) \quad (2.17)$$

The factors of the formula are as follows:

- *tf*(*t in d*): term frequency for term *t* in document *d*
- *idf*(*t*): inverse document frequency of term *t*
- *boost*(*t.field in d*): boost factor affect a query or field's influence on the score; set during indexing
- *lengthNorm*(*t.field in d*): field's normalization value, given the number of terms within the field; computed during indexing; this factor boosts fields with fewer tokens
- *coord*(*q,d*): coordination factor; based on number of a document's query terms; gives an AND-like boost to documents containing more search terms than others
- *queryNorm*(*q*): query normalization value; based on sum of squared weights of all query terms

Based on this far-reaching formula, Lucene is able to provide an explanation for the ranking of its results. This explanation can be generated for each document (identified by a document-id) and contains the final score for the document as well as the values for *tf*, *idf* and *fieldNorm*, which is equal to the aforementioned *lengthNorm*.

Regarding the implementation of Lucene in Solr, figure 2.13 illustrates the internal architecture of Solr. Its core is shown as multiple layers because the system supports multiple configurations and indexes with one single instance. The most important parts of Solr that were not mentioned above are:

- *schema.xml* [41]: It contains the fields of the documents and how the indexing or searching should deal with the field, e.g. if it should be retrievable or if it is required (see also figure 2.12). Generally, the file is structured in three parts:
 - *Data types*: This part contains all kinds of types a field can be. Each **fieldType** has a name and a class, for instance **DateField** or **TextField**, that reference a Solr class that is used for that type. It may also define the default options for fields with that type.
 - *Fields*: This section contains all fields that are used in a document. Each field has a *name* and a (*data*) *type* and may have field options, like **indexed=true** or **stored=true**.

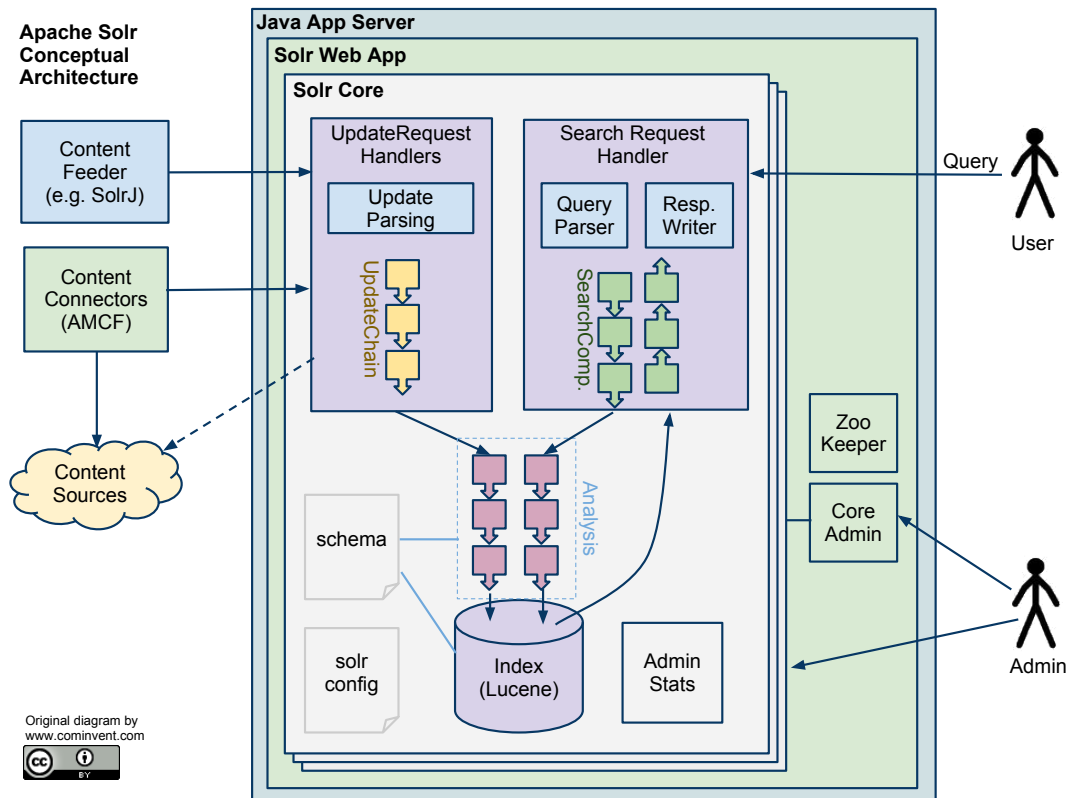


Figure 2.13.: Solr's internal architecture [14]

- *Miscellaneous settings*: These contain settings like the *unique key field*, which defines which field is an identifier of the document, the *default search field*, which sets the search field if none is provided, the *default query parser operator*, which is OR by default and can be changed to AND.
- *solrconfig.xml* [42]: This file contains parameters for configuring Solr itself. This includes the path for plugins, the path to the index data, settings for merging index segments, settings about how an update is handled, query settings like caching parameters or event listeners that fire certain code when querying, settings for HTTP requests, registering request handlers, settings for processing documents before indexing, settings for the administration web page and others.

2.5. Corpora split and result set merging

2.5.1. Improving retrievability by corpus partitioning

Bashir and Rauber often focused on the completeness of the returned result set provided by the retrieval system (cf. section 3.1.1). They found out that retrieval systems tend to

be biased towards certain document features, be it document length or rare, respectively common, words so that the systems return these documents more often than other documents. The problem arises if it is crucial to not miss any document in the result set, like it is in the patent domain where recall is of higher importance than precision. To approach this issue the *retrievability* of a retrieval system provides a measure to calculate “how much each and every document $d \in D$ is retrievable in top-n rank results of all queries” [8].

Their approach in [11], which is also subject to the experiments made later (see section 4.2 and following sections) to increase the retrievability of documents, is to basically divide the entire document collection into documents that are high or low retrievable. After this split query processing is done on each part separately and after that the two result sets are combined again. Eventually this increases the possibility to have documents with low retrievability in the result set, additionally to the already found, high retrievable ones.

To evaluate retrievability Bashir and Rauber calculate the retrievability of documents with a cumulative score which counts the number of times a document can be retrieved within the first c results (cut-off factor c) when invoked over a certain query set. In a formal description, retrievability is:

$$r(d) = \sum_{q \in Q} f(k_{dq}, c) \quad (2.18)$$

$f(k_{dq}, c)$ denotes a utility/cost function, k_{dq} is the rank of the document d in the result set of query q , whereas c is the aforementioned cut-off factor, which denotes up to which rank a user will proceed the result list. This function returns 1 if $k_{dq} \leq c$ and 0 otherwise, so $r(d)$ gets increased every time the document d is listed amongst the top c results. However, longer documents with more vocabulary will potentially have a higher $r(d)$ than shorter documents. To cope with this circumstance the authors suggest to normalize the score by the number of queries that can retrieve d , regardless of any cut-off factor. This set of queries is \hat{Q} in equation 2.19.

$$r(d) = \frac{\sum_{q \in Q} f(k_{dq}, c)}{|\hat{Q}|} \quad (2.19)$$

In order to further investigate retrievability and its inequality in collections the *Lorenz Curve* can be used. The Lorenz Curve plots statistical distributions and illustrates the disparity within the distribution [49]. It is often used to visualize the income distribution of a country. When using it for displaying the system bias, the documents are sorted in ascending order according to their retrievability score. If a system is not biased at all the curve will be linear, otherwise the skew is proportional to the amount of inequality. To summarize this amount the *Gini Coefficient* G is used. Mathematically this is “the area between the line of perfect equality and the observed Lorenz curve, as a percentage of the area between the line of perfect equality and the line of perfect inequality”[48]. In figure 2.14 this is the area named A . The Gini Coefficient is calculated as follows:

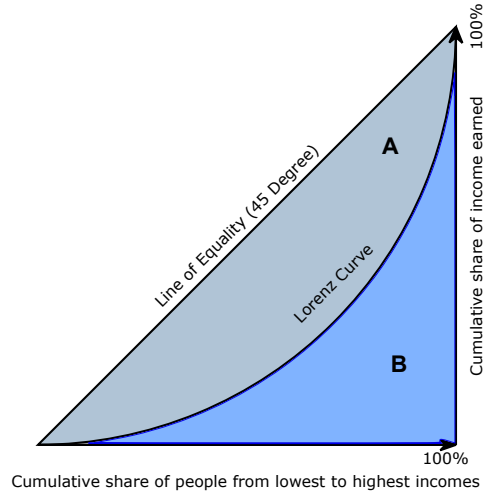


Figure 2.14.: Example of a Lorenz Curve [48]

$$G = \frac{\sum_{i=1}^n (2 \cdot i - n - 1) \cdot r(d_i)}{(n - 1) \sum_{j=1}^n r(d_j)} \quad \text{with } n = |D| \quad (2.20)$$

n is the number of documents in the collection that are sorted by their $r(d)$ value. G will be a value between 0 and 1, so the higher G is the more biased the underlying retrieval system is. This is also proportional to the area above the Lorenz Curve in figure 2.14 (area A).

To proof that the splitting of the corpus in low and high retrievable documents improves recall, several experiments are done. The first test is done over the complete corpus using all queries (single term and 2-term queries) in order to sort the documents according to their retrievability. Three prominent retrieval models are evaluated:

- TFIDF - term frequency-inverse document frequency (cf. section 2.2.2)
- BM25 - the OKAPI retrieval function (cf. section 2.2.3)
- LM - a Language Modeling approach based on Dirichlet Smoothing (cf. section 2.2.3)

The test shows that TFIDF is missing a few thousand and BM25 does not find a few hundred documents whereas LM does not have any unretrievable document and is therefore the least biased retrieval system. In the second experiment it is tested if longer queries still lead to low retrievability by 20 documents that have the highest respectively lowest average term frequencies. It is shown that the retrievability score ranking is almost identical.

The third test builds on the results of the former and checks if this behavior also occurs for a random 5% subset of the corpus and not only for extrema. For this experiment the corpus gets split into three subsets: Documents with high, medium and low retrievability

get divided into particular subsets. In order to do so, the retrievability of each document has to be calculated. Since this would be a way too extensive task for the corpus used, Bashir and Rauber apply another approach: They apply a simple set of document features, similar to the approach in [9] (compare section 3.1.1). As said before the corpus gets divided into three subsets afterwards. These subsets, respectively the high and low retrievable set, build the training instances for the model training. This is done using a 10-fold cross-validation training a Naïve Bayes classifier to determine an optimal split of the corpus. For each of the three retrieval models a particular *top* and *bottom* split is calculated in order to get three areas. On the resulting partitions queries are processed independently and afterwards the result sets get merged to one single result set, in order to ensure that eventually also poorly retrievable documents will occur in this very set. Regarding merging principles, the authors use

- *equal_size* - from each result subset (low/high retrievable) an equal number of documents gets selected for the final result set
- *partition_size* - from each result subset the number of documents selected is relative to the size of the subset

As expected, the retrieval inequality decreases for all three retrieval models when the corpus gets split before processing queries. It is also shown that the Gini Coefficient decreases as the rank cut-off factor increases. So if the user is willing to proceed further down the list, the system bias is less obvious.

In the fourth and last experiment it is evaluated if higher retrievability also improves accuracy. As said the retrievability is much higher when using a split corpus, whereas regarding accuracy it merely indicates that the accuracy improves.

2.5.2. Result set merging including different retrieval engines

The main goal of the work of Zenz et al. [59] is to analyze the effects of different indexing technologies and retrieval models on the ranking of the result sets. For this purpose three open source search engines - Solr, Lemur and Terrier, which are also used in this work for indexing and retrieval (see also sections 2.4.3, 2.4.1 and 2.4.2) - are investigated in regard to their indexing and retrieval functionality. Because the focus is only on full-text parts, metadata of the documents like the International Patent Classification (IPC) codes are ignored. Furthermore only English parts of the patent documents are indexed and the first claim of a patent document is used as topic because claims provide a representative set of terms for finding documents relevant to the invention. The same approach is used in this work, see section 4.1.2. The researchers used the ASPIRE patent collection, which is a subset of MAREC and counts 400,000 documents. This is a comparable amount of documents used in this very project.

For the indexing part there is no stemming or stopword removal used and before processing the XML markup is entirely removed for all frameworks. Also term length limitation is disabled. Regarding some other options (for instance “omit tokens with more than 4 digits”) not all three frameworks are indexing with the same adjustments.

The result of the indexing part is interesting as far as the unique number of terms is concerned: Due to different tokenization mechanisms, Solr has more than twice as much terms as Terrier (about 4 Million compared to about 2 Million), Lemur on the other hand can be found almost exactly between the two (about 3 Million). The reason for the poor performance of Terrier can be found in its special treatment of numbers. Terms with more than four consecutive digits are discarded which causes a massive reduction of terms in the patent corpus which has a high amount of numbers in its text.

For the test collection corpus the documents of the ASPIRE data set are ordered by priority date and then split by the targeted number of topics and the size of the corpus into two parts: the document collection, which are documents with priority date before January 2003, and potential topic documents with a later priority date. These topics are selected regarding certain criteria. Only patent applications are used which have English claims because the topics are formed out of the first English claim. This claim is used because it represents the content of the patent document to the most extend. Furthermore all World Intellectual Property Organization (WIPO) documents are excluded from the topic documents because of erroneous XML markup. From the remaining documents those with at least three relevance judgments are selected. All of these constraints were also used in the experiments done in this thesis, see section 4.1.2.

Eventually there are 15 experimental runs with the three retrieval frameworks selected. These have rather low MAP values due to the setup of the procedure (using the first claim as query and the average number of relevant documents per topic is about 4). The other results, which can be compared by the recall value at 10, 100 or 1000 results, show that the results of all runs are very similar for one exception. So all three frameworks perform much the same with respect to Solr which has the best results using its *MoreLikeThis* feature. In contrast to its normal ranking results, the *MoreLikeThis* functionality “computes a tf-idf score for all terms in the original query [and] the 25 terms with highest scores are taken as a query submitted to the index” [59]. Then the individual terms are connected using the Boolean OR operator.

For the result set merging different merging algorithms are tested, but none shows an all-over superior result compared to the other. Nevertheless the desired outcome, to increase the recall by merging the results, can be proved. With Solr and the *MoreLikeThis* feature run combined with the Lemur and KL-divergence run a higher number of relevant documents can be found among the first 1000 results.

Summary

The goal of this chapter was to provide an overview of this work’s background. It was defined what intellectual property means and how patents can help to maintain the right on an idea. Different ways to this, like the *first-to-file* and *first-to-invent* approaches result in different methods to obtain a patent. The part about IR provided a description of how a retrieval system works. For this, concepts like *tokens* or *postings* – steps that lead to an index – were explored. Then it was described how a retrieval system determines which terms are more important than others with models like *tfidf* and the

VSM. The most important retrieval system comparison approaches like precision/recall or *MAP* were explained. Because this work makes use of it, the *Okapi BM25* model as well as the concept of language models were emphasized as well. The patent search section provided information on different search types and when they are used as well as it described how to do a classification, a full-text or a citation search. The chapter also focused on the three different IR systems that were used: Lemur, Terrier and Solr. Each system's advantages and disadvantages as well as their functionality were described. The last section presented the work of Bashir and Rauber about corpus partitioning in low and high retrievable documents and the resulting bias decrease as well as the work of Zenz et al. about result set merging of different result sets provided by three retrieval engines.

3. Related work

The retrievability of single documents occurs as a new way of looking at the broad area of patent searching. During the patenting process it is crucial for patent examiners to find each and every patent document related to the application in order to proof its patentability. However, patent retrieval researchers found out that some patent documents are hard to find, some even cannot be found by any retrieval system at all. In order to get a handle on this important problem they developed certain mechanisms to cope with this issue, one of them picked up from the transportation planning domain, others come from the information retrieval domain itself. The following section 3.1 provides an overview of work that is related to the topic of this thesis. This includes publications on retrievability and how it was derived from accessibility in section 3.1.1, work on prior art search in 3.1.2 as well as retrievability and precision in section 3.1.3. Section 3.2 summarizes how these publications are related to this thesis' topic and where they will find application in the experiments later.

3.1. Categorization of methods

3.1.1. Accessibility, findability, retrievability

The concept of *accessibility* is a main component in the work of Azzopardi and Vinay [4]. They focus on accessibility of documents in a collection of IR systems. To do so they draw parallels between the *land use and transportation planning* domain and information retrieval. The former uses the term accessibility as “a measure of potential opportunities for interaction with resources like employment, schooling, shopping, dining, etc.” [4]. Information retrieval can be compared to a certain extent to this domain. The physical space of the transportation systems can be seen as an analogy to the information space or document collection, and the transportation system is similar to an *Information Access System* (like a browsing mechanism). So accessibility, which was previously a way of how to get from one space to another within the physical space via a transportation system, is now a way of how to receive information from a document within a collection. Or, if precisely compared to the definition of accessibility, “the opportunities are the documents in the information space, and we wish to capture the *potential of documents for retrieval*” [4]. So the measures of accessibility become measures for the retrievability potential of documents.

In the transportation domain two different measures are used to calculate accessibility:

- *Cumulative Opportunity Measures* count how many opportunities can be reached within a certain time of travelling.

- *Gravity Based Measures* implement a widely used, general method for calculating accessibility and differ from the former measure by including a cost function within the calculation.

For measuring the accessibility of a document instead of a certain location, the authors propose two measurements adapted from the equations used in the transportation planning domain. As an adaption of the *Cumulative Opportunity Measure* they suggest:

$$A(d) = \sum_{q \in Q} o_q \cdot f(c_{dq}, \theta) \quad (3.1)$$

o_q is the probability of expressing query q from all queries that are in Q . The function $f(c_{dq}, \theta)$ denotes a generalized utility/cost function where c_{dq} denotes the rank of d in the result set of query q , θ is a (set of) parameter(s) denoting the type of measure. This function returns the value 1 if $c_{dq} \leq \theta$, otherwise 0. For a *Gravity Based Measure* they adapt a function where a document's accessibility is inversely proportional to a document's rank:

$$f(c_{dq}, \beta) = \frac{1}{(c_{d,q})^\beta} \quad (3.2)$$

Accessibility of documents in a collection can be examined from the system side, in which case it is denoted as *retrievability*, and from the user side, the *findability* [3]. So high or low retrievability depends on the underlying retrieval system of the search process. Usually this retrievability imbalance occurs when the retrieval system prefers some documents in the retrieval part. The term *bias* is often used in such processes and refers to situations when a group of documents is favored over others.

For further investigation on that issue, Azzopardi and Vinay [5] adapt the equation mentioned above:

$$r(d) = \sum_{q \in Q} o_q \cdot f(k_{dq}, c) \quad (3.3)$$

where c is now the maximum rank a user will look down the ranked list. This equation is the basis for Bashir and Rauber in e.g. [8] (also compare section 2.5.1), except for o_q which is a weight and indicates how likely it is that a user will issue this query to the information retrieval system.

The authors also find out in [5] that the effectiveness of the retrieval system is not affected by removing less retrievable documents from the collection due to the fact that the probability that they are retrieved is very low. Even when using queries that are especially designed to retrieve documents with low $r(d)$, these documents are more difficult to find or cannot be found at all.

Working with the retrievability measurement adapted from the transportation planning domain, Bashir and Rauber first consider the approach of dealing with sets of relevant and irrelevant queries processed for each document in [8], which can be compared to the way recall oriented users create queries during their search process. The authors state that all retrieval systems have a certain bias towards certain document features and that the effects of this retrieval system bias have to be analyzed before using a retrieval system, especially when working in a recall oriented domain. So they

analyze the retrievability of all documents in the collection and if each document $d \in D$ has about the same retrievability score, the corresponding retrieval system is called best retrieval. They make experiments calculating the retrievability considering all queries or only relevant or irrelevant queries, respectively. For the query generation they use ideas from invalidity search, like combining for each document single frequent terms into two- and three-term combinations. Relevant and irrelevant query sets for documents are determined whether the query terms originated from the respective document or not. It is found out that even though relevant queries seem to be the most important queries for the accessibility of each document, these documents are not highly retrievable through their relevant query set but through all types of queries (irrelevant and relevant).

In [10] the same authors approach another problem that makes findability difficult: wrongly captured and interpreted context of short queries. A remedy for this can be document selection based on clustering for pseudo-relevance feedback. This can help to increase the findability as well as to decrease the bias of the retrieval system. In order to do that the strengths of different query expansion approaches are analyzed. Therefore they use an improved clustering based resampling method for pseudo-relevance feedback selection: Based on the intra-cluster similarity, clusters are accepted or rejected for pseudo-relevance feedback. With this selecting procedure, their approach tops other systems because individual documents are better findable. They also state that as the rank cut-off parameter increases, the Gini coefficient slowly decreases, so if users only have a look at the top ranked documents, the degree of retrieval bias is greater and decreases with their willingness to search deeper down in the ranking.

Bashir and Rauber further investigate the problem of findability/retrievability in [9] where they use various content-based features to classify the documents of a corpus into low and high retrievable ones. This can be used to automatically divide the corpus in low and high retrievable documents. The content-based features they use are:

- rare terms ratio (RTR),
- average terms frequencies (ATF),
- frequent terms count (FTC),
- average terms probabilities in related patents (ATPrd),
- average terms probabilities in whole set (ATP) and
- patent length (PL).

Other features are also investigated but do not bring useful results. For their experiments they use different retrieval models, of which TFIDF and JM show the least retrievability inequality, except for the RTR feature where they perform worst, which leads to the assumption that this feature can be used to identify low findable documents when working with these retrieval models. Usually the document's length is considered to be the reason for low retrievability of the document. However, regarding these experiments the authors find out that the low retrievability in a particular retrieval system can also have

other reasons and that retrieval systems can still be useful for particular retrieval tasks, for instance for finding patents with frequently used *rare terms* (if the retrieval system is biased towards this feature). Eventually, it can be shown that content-based features can be used to automatically classify and separate - with reasonable accuracy - low and high retrievable patents.

3.1.2. Prior art search

Prior art search is the task of finding all information that can be relevant to proof a patent's novelty, therefore it is one of the main search tasks in patent retrieval. Identifying prior art is part of the patentability (or novelty) search and is the most commonly used search type. It has an important part in the (in-)validity searches as well.

In [12] Bashir and Rauber deal with improved retrievability in prior art search by expanding prior art search queries that were generated from query patents by using query expansion with pseudo-relevance feedback. This approach is used to help finding patents where the writers of the patent application developed their own terminology in order to pass the patent examination process, which is a common problem. Interestingly, in the summary and abstract fields the performance results are much better than in the other fields, because there the patent writers stick to a certain language and try to keep it short. Bashir and Rauber use the presented method to cope with the problem that prior art queries sometimes do not extract all relevant terms from query patents, so these terms can then be extracted from the newly found pseudo-relevance feedback documents. In order to do so they develop an approach for finding more relevant patents for pseudo-relevance feedback: The documents "are identified based on their similarity with query patents via specific terms"[12], these specific terms are also called *subset of terms*. It can be shown that their novel approach eventually increases the retrievability of single patent documents.

Another methodology to work with in prior art search, that is also used in the paper of Zenz et al. (cf. section 2.5.2), is proposed by Graf and Azzopardi in [21]. The authors propose a method for creating topics for prior art search. Their goal is to develop a methodology for all kinds of patent sources, also for different languages. According to the authors the basic steps for creating topics for prior art search are as follows:

1. Define a certain set of documents that form the corpus.
2. Define the potential information needs that are a pool of documents that do not have the same time period as the corpus documents.
3. From this pool one patent document gets selected.
4. Then the patent document's references to prior art are extracted.
5. The references within the pool are identified.
6. The documents of existing references get a relevant flag.
7. A topic gets defined by using

- the **patent application** without references or a **subset of its text** as a *query*,
- the **extracted references** as *relevance assessments* and
- the **relevant prior art definition** as *information need*.

8. Repeat steps 3-7 for each document in the pool from step 2.

In order to justify this kind of reverse engineering relevance assessment from the references, certain requirements have to be fulfilled: the examiner at the patent office has to be qualified for subject and legal expertise with respect to the prior art search or examination manuals have to provide additional guidance for the interpretation of references. The authors argue that with a low number of relevant documents per topic, which is the case for many patent documents, the results of the method may be inaccurate. Another problem may be the different formats in which patent documents are provided. These are an obstacle concerning structure and bibliographic data as well as the valid evaluation measures.

3.1.3. Retrievability and precision

As said before, in recall oriented domains the precision of the result set does not need to be high in order to have a good retrieval system. As long as the recall improves, the precision value can be disregarded. Nevertheless, in some experiments the precision measurement is also investigated. Azzopardi and Bache look at that matter more closely in [3] to see if accessibility, represented by the Gini measurement, and effectiveness, represented by the precision measurement, are compatible. The findings of the authors suggest that there appears to be a relationship between these two measurements. It turns out that if parameters are selected that maximize the access, the retrieval performance is still acceptably good. This also leads to the suggestion that the configuration of a system with missing relevance information can also be done reasonably by providing the user easily accessible documents.

In the work of Bashir and Rauber [11] (see section 2.5.1 for more details) there is also one experiment to evaluate if improving retrievability leads to higher or lower accuracy on prior art search task patents. Though the accuracy rates of their experiment cannot be compared to other retrieval engines because the experiments were designed for high recall and high retrievability, they indicate that the splitting of documents in high and low retrievable classes improves not only the retrievability of the patent documents but also the accuracy of the result set. The authors argue that the improvement of accuracy depends on the retrieval model as well as on the used query generation process.

3.2. Assessment of methods

The presented methods provide reasonable results in specific tasks. For this work particular attention will be paid on one hand on the retrievability measurement, on the other hand on prior art search. The former will be useful when calculating low and high

retrievability of documents in order to split the corpora in two classes (see section 2.5.1). For that content-based features like in [9] of the documents will be used to quickly provide the partitioning. The latter, prior art search, is used in the work of Zenz et al. (see [59] or section 2.5.2 for more details). The authors use the method presented by Graf and Azzopardi in [21] in order to create the test collection.

Summary

This chapter provided an overview over existing research topics in the patent retrievability field, respectively in the retrievability domain. Adapting calculations from the land use and transportation planning domain, the terms accessibility, findability and retrievability were examined. It was also explained how to see retrievability, the term most authors agreed on, as the bias of some retrieval systems towards certain documents. Later on different approaches towards improved retrievability were shown. There were approaches like document selection based on clustering for pseudo-relevance feedback or automatic document categorization with content-based document features. Another field, the prior art search, was also investigated. The most prominent work was a reverse engineering relevance assessment from references. The last section investigated the compatibility of retrievability and effectiveness, respectively precision. Finally a short assessment of the presented methods was made.

4. Methods

A theory is something nobody believes, except the person who made it. An experiment is something everybody believes, except the person who made it.

Albert Einstein

The previous chapters explained methods or approaches to improve crucial measures of the patent search domain, namely the split of a corpus or the merge of result sets. These are now going to be the conceptual framework for the subsequent experiments. At first, a system set-up in section 4.1 has to be done to provide a basic structure for experimenting. It describes hardware and software used, the data collection and preprocessing steps that were necessary for working with three different retrieval engines, how topics and relevance assessments were created as well as which parameters the single retrieval engines used for indexing and retrieval steps. Eventually, the baseline results generated with these configurations are discussed. Section 4.2 describes how the corpus split was prepared and executed and provides information on the high/low retrievable distribution. An analysis of the split corpora is shown in 4.3, which also includes first information about the subsequent merging. Finally, the result set merging section 4.4 contains information on the implementation done for combining a high and low result set file, which methods were realized, which difficulties were met and which additional features were brought in.

4.1. System set-up

This section is assigned to describe the preliminary work that was done prior to doing experiments regarding the topic of this work. First of all a short overview is given of the hardware on which the retrieval engines were run. Then the data collection used, the parser, which processed the raw data and produced a usable text form, as well as how topics and relevance judgments were obtained are described. The last part of this chapter contains an overview of the individual indexing and retrieval steps that were done by each of the three used retrieval engines in order to receive the baseline retrieval values with which the values of the experiments will be compared to.

4.1.1. Hardware and software versions

Two hardware setups were used during the indexing and retrieval process, the software used was similar:

- **Supercomputer mdc**¹: provided by the IRF; includes two *IBM x3950M2* server,

¹as of May 2011, it is not available anymore

32 Cores (four quad core *Intel Xeon* 2.93GHz per node), 256 GB main memory; a production cluster for Java and serial code; Linux operating system using *Java 1.6.0_07*

- **PC:** *AMD Phenom II X4*, 4 GB main memory; *Ubuntu 10.04 LTS* using *Java 1.6.0_24*

On both machines the same versions of the three retrieval engines were used:

- Lemur: Lemur version 4.12, respectively Indri version 2.12
- Terrier: version 3.0
- Solr: version 3.1 with Lucene 3.1

Only Lemur had to be installed on the systems by compiling the source code, the two others ran out of the box.

4.1.2. Creation of the test collection

In order to obtain an index, the documents of the data collection had to be pre-processed and transformed into a format which the retrieval engines could index. Then the documents of the data collection had to be divided into the corpus, which was indexed, and the potential topics. From the latter ones the documents complying to specific requirements (cf. section 2.5.2) were sorted out and used as topics. Details on the data collection, the parser and how corpus, topics and relevance judgments were obtained, follows.

Data collection

The data collection used in this work is named *MAREC-400k*, which is a random subset of 400,000 documents of the *MAREC* collection. *MAREC* again is a subset of a larger repository, *Alexandria*, and includes patents from the four main patent offices - EPO (subsequently referred to as EP), WIPO (WO), USPTO (US) and JPO (JP) (see also 2.1.2). It consists of over 19 million patent documents, which include text in English, French and German and which are formatted in a highly structured XML format. The XML fields include dates, countries, languages, references, person names, companies as well as IPC codes. The US, EP and WO documents include applications and granted patents, the JP patents are only applications. The distribution of documents of each patent office in the collection can be seen in figure 4.1. The *MAREC-400k* contains 100,000 documents of each patent office. [25]

For the experiments in sections 4.2 and 4.4 as well as for the baseline created prior to that not all documents of the *MAREC-400k* were used. According to [59] and because they do not contain any description, JP patents were excluded as well as all non-English documents. The language was checked in each XML file in the *patent-document* tag's attribute *lang*. In [59] it is stated that the XML markup of WO documents had erroneous

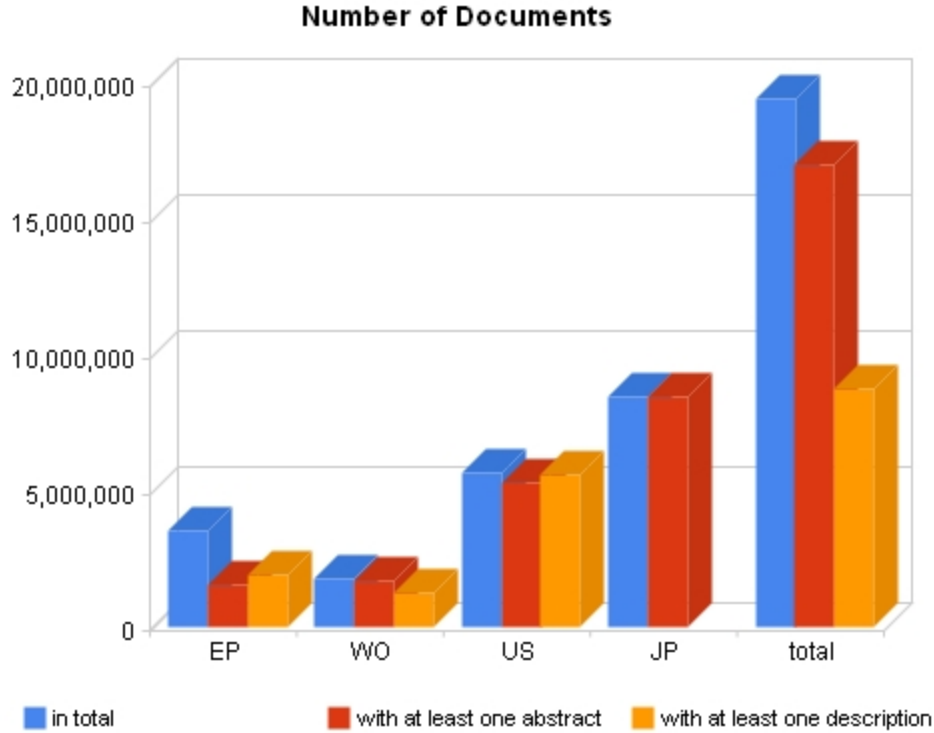


Figure 4.1.: Distribution among MAREC documents on abstract and description [26].

claim numbers. This could be verified, for the patents included all claims in the tag that refers to the first claim and therefore could not be parsed properly. Thus, WO patents were indexed but were excluded from the list of topics.

Parser

The first parser used was part of a project done prior to this work. It uses *StAX* (Streaming API for XML) which is an API used for reading and writing XML documents in Java. *StAX* belongs to the event-based XML APIs like *SAX* that read in the document in pieces and if a certain element is encountered an event gets triggered. APIs like these are also called *push-API*. *StAX* on the contrary is a *pull-API* where the next part of the document gets requested actively. It provides an *iterator* and *cursor* processing model, of which the former one was chosen to parse the files because it is more flexible. [32]

The parser takes MAREC-style files, parses them, and writes out a TREC-style file in which the `<TEXT>` tag contains contents from the MAREC files. An example TREC-style file can be found in listing 4.1.

For using the parser in this work, the section about writing out the path and id of images had to be removed in order to work with the second parser more easily. The first parser provides one file which includes many sections like *bibliographic data* or *drawings*

Listing 4.1: Example TREC file

```
<DOC>
<DOCNO>US-20050205757-A</DOCNO>
<TEXT>
Patent document text
</TEXT>
</DOC>
```

that do not have to be indexed for this work. Because the generated file has to be parsed again, working with one file is faster. It is also useful to write only one file in the end because Lemur works better with fewer files to index.

The second parser takes the result of the first step and only keeps the following sections that build the default basis of a common index:

- title
- abstract
- description
- claims

The output is written into two files. The first one still contains TREC-style markup and tags related to the sections mentioned above and is used by Terrier and Lemur. An example is given in listing 4.2. The different markups were not indexed into fields but

Listing 4.2: Example markup for Terrier and Lemur

```
<DOC>
<DOCNO>US-5141493-A</DOCNO>
<TEXT>
<title>Peritoneal dialysis system</title>
<abstract> A peritoneal dialysis system is disclosed for ... </abstract>
<description> BACKGROUND OF THE INVENTION This invention relates to ...
  </description>
<claims> I claim: 1. A peritoneal dialysis system for administering ...
  </claims>
</TEXT>
</DOC>

<DOC>
<DOCNO>US-5141409-A</DOCNO>
...
```

created in case they are needed for experiments later on. The second output is used for indexing with Solr and is similar to the example given in listing 4.3. For Solr, all text is put into one field to index like it is done for the other engines. Additionally, because

Listing 4.3: Example markup for Solr

```
<add>
<doc>
<field name="id">US-5141493-A</field>
<field name="toIndex"> Peritoneal dialysis system
  A peritoneal dialysis system is disclosed for ...
  BACKGROUND OF THE INVENTION This invention relates to ...
  I claim: 1. A peritoneal dialysis system for administering ...
</field>
</doc>

<doc>
<field name="id">US-5141409-A</field>
...
```

Solr needs valid xml text to index, certain special characters (&, <, >, \, ') had to be changed (e.g. replacing & with &).

Topics and relevance assessments

As mentioned above, only a subset of the MAREC-400k corpus was used. Patents from the JP subset and documents with no *lang*="EN" attribute in their *patent-document* tag were excluded. This leads to 219,379 documents that build the corpus of potential topics and collection on which the topics are issued in the retrieval step, according to [59].

For generating the **topics**, the methodology presented in [21] and used in [59] was applied. This included:

- ordering all documents in the collection by priority date
- splitting the collection at January 2003 (as in [59])
- documents with a date prior to the split date form the corpus for the experiments, documents with a date after the split build the pool of potential topics

From this pool of potential topics only patents that fulfill certain requirements were chosen. The requirements were:

- only applications (document kind "A", see section 2.1.3)
- only patents that have English claims
- only patents that have at least three relevance judgments

In order to check the requirements and split the data a metadata file, which contained information about *document kind*, *publication date*, *has English description*, *has English abstract* and *has English claims* for each patent was generated by the IRF and kindly

Table 4.1.: Test collection summary

Test collection	description	#	empty
Corpus	priority date <i>before</i> Jan. 20003	122.690	90
Topics	priority date <i>after</i> Jan. 2003; only US,EP; only applications; at least 3 relevance judgments from SEA, EXA	255	22 ²
Relevance judgments	direct and extended citations	254.924	-

provided for this work. After generating the relevance judgments (see below), the patents with at least three relevance judgments were selected and their first claim was extracted as topic. For this, three files were generated to evaluate the result sets with TREC evaluation tools:

- *qrrels file*: The *Relevance Judgments File List*, a file containing information about which topic is relevant for which document number. The columns of the file are *TOPIC*, *ITERATION*, *DOCUMENT NUMBER* and *RELEVANCY*, where the latter is always 1 because only relevant documents are calculated out of the citations. [35]
- *topicNumberUcid file*: A simple file that maps the topic number to the patent ucid where the topic came from. For troubleshooting purposes.
- *topics file*: Contains the topic number (also in both other files) and the topic text in a suitable markup for Terrier and Lemur (see also sections 4.1.3 and 4.1.4). For Solr, the Terrier topics file was used to extract the topics (see section 4.1.5).

For the **relevance judgments** all necessary direct and extended citations from the collection were created by the IRF. Extended citations were obtained by expanding the citations along simple patent families. This huge amount of about 44 million citations also returned citations that were not contained in the collection so it had to be checked for suitable documents. From the remaining citations only the patents that fulfilled the requirements mentioned above and that were assigned by the search report (SEA) or the examiner (EXA) were chosen for topic extraction. The last attribute is the only one not mentioned in [59], but because the search report and examiner have higher relevance than an applicant when selecting a citation this decision was made.

Finally (see also table 4.1), the number of documents in the collection that were indexed by each retrieval system is 122,690, including 90 empty documents (according to Terrier). There were 277 topics extracted of which only 255 were chosen because the others did not contain any text in their first claim and can be regarded as empty. In general, empty documents contain only bibliographic data and appear to happen when

²19 were automatically sorted out because they had erroneous markup in the first claim; 3 were manually sorted out because they only contained the word “Canceled”

for instance a patent is found that has two versions (e.g. B1 and A3) and one of them is empty (e.g. A3, which is the publication of a search report). Because the number of empty documents in the corpus was insignificant, they were not sorted out.

4.1.3. Terrier

The downloadable file of Terrier was uncompressed and used with small modifications of the properties files in the `./etc/` folder.

Indexing

The indexing was done according to the parameters selected in the work of Zenz et al. [59]. Therefore no stemming and stopword removal was done. As mentioned in section 4.1.2 no field indexing was done. Terrier has a maximum term length limit of 20 which was set to 2000 to allow indexing longer words on one hand as well as to prevent indexing strings that may occur because of parsing shortcomings and that are not likely to be present more often, on the other hand.

For the indexing part, only the *terrier.properties* file had to be modified. The selected properties³ were:

- *terrier.home* - path to the topmost terrier directory
- *terrier.index.path* - path to the index directory; per default inside the terrier home directory
- *collection.spec* - path to a file that contains the files to be indexed; this file is generated by the parser and contains only one entry
- *trec.collection.class* - which collection object to use to index
- *max.term.length* - set to 2000 as mentioned above
- *TrecDocTags.doctag/idtag* - the name of the tag that sets the start of a document, respectively sets the document id

Example parameters of the *terrier.properties* file for the indexing part are shown in listing 4.4. Indexing was done by simply invoking `./bin/trec_terrier.sh -i` without additional arguments.

Retrieval

For retrieval, the *terrier.properties* file also needs:

- *trec.topics* - path to the file which contains all topics for the run (cf. section 4.1.2); for Terrier a tagging as shown in listing 4.5 was used

³More information on terrier properties: <http://terrier.org/docs/v3.0/properties.html>

- *TrecQueryTags.doctag* - name of the tag that sets the start of a query (by this Terrier can discriminate different queries in one file)
- *TrecQueryTags.process* - list of tags to process
- *TrecQueryTags.idtag* - name of the tag that sets the query id
- *ignore.low.idf.terms* - set to *false* for some retrieval models (labeled with * in section 4.1.6)

Terrier writes out 1000 results per default, so no property had to be set for that.

Listing 4.4: terrier.properties file

```

1 # for indexing:
2 terrier.home=/path/to/terrier-3.0
3 terrier.index.path=/path/to/terrier-3.0/index
4 collection.spec=/path/to/parser/temp/indexlist.txt
5 trec.collection.class=TRECCollection
6 max.term.length=2000
7 TrecDocTags.doctag=DOC
8 TrecDocTags.idtag=DOCNO
9
10 # for retrieval:
11 trec.topics=/path/to/terrier-3.0/_topics/topics.txt
12 TrecQueryTags.doctag=query
13 TrecQueryTags.process=query,number,text
14 TrecQueryTags.idtag=number
15 ignore.low.idf.terms=false # for some retrieval models

```

Listing 4.5: Terrier example query

```

<query>
<number>294</number>
<text>
1 A system for providing automated translations of documents
[... ]
</text>
</query>

<query>
...
</query>

```

Terrier may need additionally to the above mentioned properties file:

- *./etc/trec.qrels* - file containing the query relevance assessments file (cf. section 4.1.2, the qrels file), if evaluation should be done with Terrier

- *./etc/trec.models* - file containing the retrieval models to use; can also be put into *terrier.properties*

The retrieval was invoked by calling `./bin/trec_terrier.sh -r` without additional arguments.

Because Lemur (cf. section 4.1.4) was not able to work with queries using symbols like *point(.)*, *comma(,)*, *semicolon(,)*, Terrier was tested with two different types of topics: one containing symbols, one without symbols, where the latter was generated out of the former by replacing symbols with whitespace.

4.1.4. Lemur

In contrast to the other two toolkits Lemur had to be installed before using it. Therefore the properties files could be saved in a folder of choice.

Indexing

The same basic properties as in Terrier were used for indexing. Thus no stemming and stopword removal was done, all text was indexed into one field. Lemur does not seem to have a term length limit so no parameters were set for this. Only one file had to be provided for the indexing step which is shown in listing 4.6. The used properties are⁴:

- *index* - path to the folder where the index will be created or files will be added to an existing index
- *memory* - how much memory will be used to index
- *indexType* - the type of index, can be *key* (KeyfileIncIndex) or *indri* (LemurIndriIndex)
- *corpus* - parameters related to the corpus
- *corpus/path* - path to the file that has to be indexed
- *corpus/class* - the file type of the file(s) of the previous line
- *field* - element specifying the fields to index as data
- *field/parserName* - contains the name of the parser to use; WebParser handles documents with html markup; used if any markup is left from parsing shortcomings

Lemur was started by invoking `IndriBuildIndex parameter_file`.

⁴More information on Lemur properties:

[http://sourceforge.net/apps/trac/lemur/wiki/IndriBuildIndex Parameters](http://sourceforge.net/apps/trac/lemur/wiki/IndriBuildIndex%20Parameters)

Listing 4.6: Lemur properties for indexing

```

1 <parameters>
2   <index>/path/to/lemur/index</index>
3   <memory>1G</memory>
4   <indexType>indri</indexType>
5   <corpus>
6     <path>/path/to/parser/temp/onefile_essentials.trec</path>
7     <class>trectext</class>
8   </corpus>
9   <field>
10    <parserName>WebParser</parserName>
11  </field>
12 </parameters>

```

Retrieval

Lemur did not need any parameter settings regarding retrieval, only a properly formatted *topics file*. An example tagging can be found in listing 4.7.

Listing 4.7: Lemur example query

```

<parameters>
<query>
<number>294</number>
<text>
#combine( 1 A system for providing automated translations of
documents [...] )
</text>
</query>

<query>
...
</query>
</parameters>

```

The *#combine* statement, which was only used for Indri runs (see section 4.1.6), defines a simple query in which terms are equally weighted and get *OR* combined. For other Lemur runs the queries looked similar to Terrier queries (see listing 4.5). Because Lemur did not understand queries that had any symbols, like *point(.)*, *comma(,)*, *semicolon(,)*, all of them were removed before the run.

Retrieval was invoked by calling

```

IndriRunQuery topics.txt -count=1000 -index=index/ -trecFormat=true >
  results/results.res

```

where the **count**-argument denotes that Lemur should write out 1000 results per query, the **index**-argument denotes the path to the index to use and the **trecFormat**-argument denotes to write out the results in a specific format usable for evaluating TREC-style results. The output then has the format shown in listing 4.8. For *tfidf* and *okapi* retrieval the additional argument **-baseline=tfidf** respectively **-baseline=okapi** was used.

Listing 4.8: TREC formatting

```
<queryID> Q0 <DocID> <rank> <score> <runID>
```

4.1.5. Solr

Solr could be used by simply uncompressing the downloaded file and using the examples provided, including the Java Servlet Container *Jetty* and the existing `./example/exampledocs/post.sh` file for adding files to the index. Other files in `./example/` could also be used with only small modifications.

Indexing

For indexing the valid xml file generated by the parser (cf. section 4.1.2), a *schema.xml* file has to be provided because the xml file has to comply to the schema. The schema file used can be found in listing A.1⁵. Only two data types are used, namely *string* and *text*. The former is only used for the id, which will be the ucid of each patent. The latter is used for the text to index and the queries and hence includes two analyzers, which pre-process text during indexing or searching. Both analyzers use the *solr.StandardTokenizerFactory* that strips irrelevant characters and sets token types. The *solr.LowerCaseFilterFactory* lowercases all letters and strips everything that is not a letter. The *fields* section of the schema includes two fields: one for the id (ucid) and one for the text to index. Both fields are indexed, because only then can they be searched too, and stored to make the field value retrievable. From the *toIndex* field the term vectors are stored additionally because this helps the *MoreLikeThis* functionality of Solr regarding efficiency. A *uniqueKey* tells Solr that the ucid of each patent is unique and if a document with the same id is added, the old document is deleted. The *defaultSearchField* mentions the field to do retrieval on if no specific field is used in the query.

Additionally to the schema.xml, the Solr server listening port had to be changed to not interfere with other Solr instances when working on the mdc. For this, in file `./etc/jetty.xml` the line `<Set name="port"><SystemProperty name="jetty.port" default="8983"/></Set>` was changed to port 8980. Also file `./example/exampledocs/post.sh` had to be adapted accordingly.

The Solr server was started by calling the `./example/start.jar` file, the files were added by calling `./example/exampledocs/post.sh fileToAdd.xml`.

Retrieval

Retrieval with Solr is more complex compared to the other two retrieval engines. First of all a client that sends the queries to the Solr server has to be implemented. For this there are already implemented clients⁶ or one can implement a simple HTTP request. The latter method was used in this work because with this it was possible to write the results in a TREC-style format which could then be used for the evaluation. The code

⁵More information on the parameters: <http://wiki.apache.org/solr/SchemaXml>

⁶See <http://solrclient.com/> for more information.

fragment in listing A.2 shows the HTTP request for standard queries, whereas listing A.3 demonstrates the request for mlt queries. The client reads in a topic file that is similar to the one Terrier uses (cf. listing 4.5) and reads out the query number and text. The text is then encoded in a URL compliant string and sent to the Solr server. The server returns a result set for the query which is then brought into a TREC-style format and written out to the final result set. Two components of the client have to be discussed in more detail:

- **standard and mlt queries:** A standard query request for a query includes `select/`. By sending the line with this statement, Solr invokes the standard request handler from *solrconfig.xml*, which can be found in listing 4.9 together with an example standard query for the word “schema”⁷. To get 1000 results back, the number of `rows` has to be changed accordingly. For the *mlt* functionality, the query request contains `mlt/`, which invokes the Solr *MoreLikeThisHandler*, which is not included by default in *solrconfig.xml* and can be found in listing 4.10 together with an example mlt query. The mlt handler operates on the first result for the initial query by default and searches for more documents that are similar to this one. The `rows` attribute is set to 999 because the returned result set includes the first result of the initial query too.
- **usage of xslt file:** In order to receive a TREC-style formatted output (cf. listing 4.8), an xslt file had to be used, which was kindly provided by the IRF. The content of this file is shown in listing A.4. Including this style sheet in the HTTP request is done by adding `stylesheet=` and `tr=SOLR2TREC.xsl` to the url.

Listing 4.9: Solr standard query and request handler

```

1 http://localhost:8980/solr/select/?stylesheet=&q=schema&fl=score&wt=xslt&tr
   =SOLR2TREC.xsl
2
3 <requestHandler name="standard" class="solr.SearchHandler" default="true">
4   <lst name="defaults">
5     <str name="echoParams">explicit</str>
6     <int name="rows">1000</int>
7   </lst>
8 </requestHandler>

```

Additionally to the client a few changes had to be done to cope with the length of some queries or other problems:

- in *./example/etc/jetty.xml*:
 - Line `<Set name="headerBufferSize">65535</Set>` was added to `//Call name="addConnector"/Arg/New` because Jetty had problems working with long queries.

⁷More about SolrRequestHandlers can be found in the wiki

Listing 4.10: Solr MoreLikeThis handler

```

1 http://localhost:8980/solr/mlt/?stylesheet=&q=schema&mlt.fl=toIndex&fl=
  score&wt=xslt&tr=SOLR2TREC.xsl
2
3 <requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
4   <lst name="defaults">
5     <int name="rows">999</int>
6   </lst>
7 </requestHandler>

```

- in *solrconfig.xml*:
 - The file contains a bug⁸ that also sometimes disturbs Solr retrieval, so the request handler name *search* was changed to *standard*.
 - **maxBooleanClauses** were doubled (set to 2048) to deal with longer queries.
 - Within the *XSLTResponseWriter* the value of **xsltCacheLifetimeSeconds** was changed to 1000 to improve performance.

As with Terrier, Solr used two different topic sets for retrieval: one using symbols, one without symbols. Additionally, the retrieval with each topic set was done one time with the standard request handler, one time with the mlt request handler, both without any attributes as seen above in listings 4.9 and 4.10.

4.1.6. Baseline

The baseline consists of the values to which the results of further experiments will be compared to. Thus it includes manifold runs with each retrieval system to allow for a wide range of comparable values. These values are calculated with *trec_eval 8.1*⁹, which is the standard evaluation tool for the TREC community that computes common measures like *MAP* or *recall@X* by reading in the results file (in a standardized format) and a relevance judgments file.

Each retrieval engine was tested with its standard and most common weighting models. These were:

- Terrier (t.< *model* >)[45] (* labels models with parameter *ignore.low.idf.terms=false*):
 - **BB2**: Bose-Einstein model for randomness, the ratio of two Bernoulli's processes for first normalization, and normalization 2 for term frequency normalization
 - **BM25***: BM25 probabilistic model

⁸see <http://whomwah.com/2011/04/05/fixing-the-400-bad-request-problem-with-solr-3-1-0-and-ruby-solr/>

⁹http://trec.nist.gov/trec_eval/trec_eval.8.1.tar.gz

- **DFR_BM25***: DFR version of BM25
 - **DLH**: DLH hyper-geometric DFR model
 - **DLH13***: improved version of DLH
 - **Hiemstra_LM***: Hiemstra’s language model
 - **IFB2**: Inverse Term Frequency model for randomness, the ratio of two Bernoulli’s processes for first normalization, and normalization 2 for term frequency normalization
 - **In_expB2***: Inverse expected document frequency model for randomness, the ratio of two Bernoulli’s processes for first normalization, and normalization 2 for term frequency normalization
 - **In_expC2***: Inverse expected document frequency model for randomness, the ratio of two Bernoulli’s processes for first normalization, and normalization 2 for term frequency normalization with natural logarithm
 - **InL2***: Inverse document frequency model for randomness, Laplace succession for first normalization, and normalization 2 for term frequency normalization
 - **LemurTF_IDF***: Lemur’s version of the *tfidf* weighting function
 - **PL2**: Poisson estimation for randomness, Laplace succession for first normalization, and normalization 2 for term frequency normalization
 - **TF_IDF***: *tfidf* weighting function, where *tf* is given by Robertson’s *tf* and *idf* is given by the standard Sparck Jones’ *idf*
- Lemur (l.< *model* >)¹⁰:
 - **indri**: standard weighting model; needs formatting: `#compare(query)`
 - **TFIDF**: *tfidf* weighting; no formatting; invoked by adding the baseline argument `-baseline=tfidf` which has default values $k1 = 1.2$, $b = 0.75$
 - **okapi/BM25**: BM25 probabilistic model; no formatting; invoked by adding the argument `-baseline=okapi` which has default values $k1 = 1.2$, $b = 0.75$, $k3 = 7$
 - Solr (s.< *model* >):
 - **standard**: using the `select/` statement in the HTTP request
 - **mlt** : using the `mlt/` statement in the HTTP request

To give an overview of the performance of each run, the following measures were chosen: MAP, recall@10, recall@100, recall@1000, precision@10, precision@20, precision@30, precision@100, precision@1000 and the number of relevant-retrieved documents (RR). The close precision values were chosen to show the difference in the top k results

¹⁰Note: other models (e.g. run in [59]) were deprecated in the version used.

compared to the merged results, which will be the subject of discussion in chapter 5. Table B.2 with *Topics I*, the topic set that does not include symbols, and table B.3 with *Topics II*, the topics set with symbols, show the values for the baseline runs.

Terrier returned in both topic sets almost everytime the same number of documents. Where it was not exactly the same, the numbers did not differ much. The MAP value was slightly better for the *Topics I* on average, so were the *recall@k* values. The *precision@k* value tended to be better for *Topics I*, but only with small *k*. Regarding the topics without special characters, Lemur performed better according to the MAP value (with *tfidf* and *indri*), but the recall values of the Terrier.Hiemstra model were slightly higher. Lemur’s okapi model did not perform that well, especially when it comes to precision values. Section 4.3 also discusses the Lemur.okapi model and its relevant documents curiosity. Compared to other Terrier models, Solr’s values were higher. But when looking at the different topic sets, Solr retrieved a different number of documents which is also reflected in MAP and recall values with higher difference compared to the Terrier results. Regarding its precision values there was not much difference between *Topics I* and *Topics II*.

Because the topic sets did not differ much for Terrier and Solr, the subsequent experiments were done with *Topics I* only.

4.2. Corpora split

For further experiments the indexed corpus of the *MAREC-400k* collection was split in a low and a high corpus based on the retrievability findings. For this, existing retrievability values generated for the high/low split research by Bashir (cf. section 2.5.1 and [11]) were used. The patents of the dataset were divided up into six different partitions, which gradually assign a document to a low or high findable partition. The grades were almost equally distributed:

	bm25	tfidf
<i>total</i> ¹¹	219,379	219,379
<i>0</i>	36,569	36,569
<i>1</i>	36,562	36,562
<i>2</i>	36,562	36,562
<i>3</i>	36,562	36,562
<i>4</i>	36,562	36,562
<i>5</i>	36,562	36,562

For the experiments in this work only low or high findability was considered, so the six grades were reduced to two, each containing three former grades. An example of the original parameter file is given in listing 4.11. Retrievability was calculated for two retrieval models, the standard TFIDF model and the BM25 model (see also sections

¹¹The low/high partitioning was only done for documents used from the *MAREC-400k* collection that build the main collection of this work, see section 4.1.2.

Listing 4.11: Example findability file

```
MAREC.400000\ref_100000_US\US\000000\00\H3\10\US-H310-H.xml 0
MAREC.400000\ref_100000_US\US\000000\00\H3\40\US-H340-H.xml 1
MAREC.400000\ref_100000_US\US\000000\00\H4\27\US-H427-H.xml 5
```

2.2.2 and 2.2.3)¹². Only indexed documents were considered for the split. This led to four document lists, a low and high list for each classifier (bm25/tfidf). With these, four different indexes for each retrieval engine were obtained with the same parameters used as in section 4.1. These indexes contained a different number of documents (with the number of empty files in brackets):

	low (empty)	high (empty)
<i>tfidf</i>	58,222 (11)	64,468 (79)
<i>bm25</i>	57,181 (10)	65,509 (80)

The ratio of low to high documents was very close, nevertheless did the low indexes take up about twice as much disk space. This could be due to having much more text in them. More information on the retrievability classification can be found in [11].

4.3. Analysis of split corpora

For each retrieval engine, retrieval runs were done on the four indexes with the same retrieval models as described in section 4.1. Because the differences between *Topics I* and *Topics II* in section 4.1.6 were insignificant the runs were only done with *Topics I*. An analysis of the result sets showed that all retrieval engines in almost all models gave high retrievable documents in the new, split result set a higher score compared to the baseline result sets. The low retrievable documents on the contrary received in almost all cases a lower score compared to the baseline results. This was tested for both relevant and non-relevant documents, where each document in the low respectively high result set got compared to the adequate result set line in the baseline result set. Only the following retrieval models ranked more low documents higher than they ranked high documents lower in both (bm25 and tfidf) new result sets:

- terrier.Hiemstra_LM
- terrier.DLH13

This might be an indication to which model is more applicable to the corpora split and merging (see chapter 5 for results and 6.1 for discussion). Interestingly, Hiemstra implements a language model function, which was mentioned in [11] (see section 2.5.1) to be least biased compared to other models.

¹²For a better differentiation between split and models, the bm25/tfidf split will be lowercased whereas the BM25/TFIDF models will be uppercased in this thesis.

Another analysis dealt with the distribution of relevant documents within the low and high result sets. For this purpose the number of relevant documents on each rank was counted throughout the result sets for all queries of each retrieval model and additionally over all low or high result sets. As an example, which is representative for both splitting methods and for almost all retrieval models of each retrieval engine, the distribution of relevant documents in Terrier models with the tfidf split is given in figure 4.2 for the low result sets and in figure 4.3 for the high result sets. The comparison shows that the

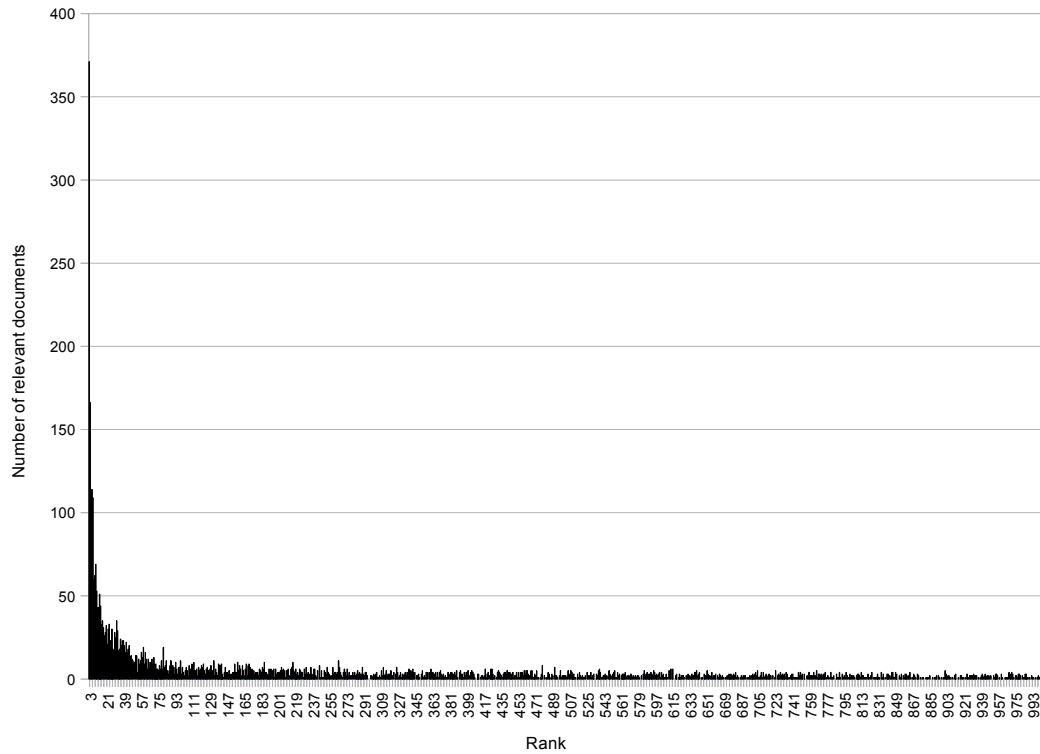


Figure 4.2.: Frequencies of relevant documents in result sets of Terrier.tfidf.low

low results contain on average more relevant documents on the first ranks. Figure 4.4 plots the sum of relevant documents up to a certain rank, from which can be seen that the low results only perform better in the first 50 ranks. From rank 59 up the sum of relevant documents in the high results is higher than in the low results. The frequency figures show that from there on the number of relevant documents in both result sets remain static, though the high results contain on average more relevant documents per rank than the low results.

From these findings, the following two assumptions were made:

1. Because of the mismatch of scores (documents get higher scores in high result sets and lower scores in low result sets), low and high result set scores have to be normalized before the merging step to compensate this fact.

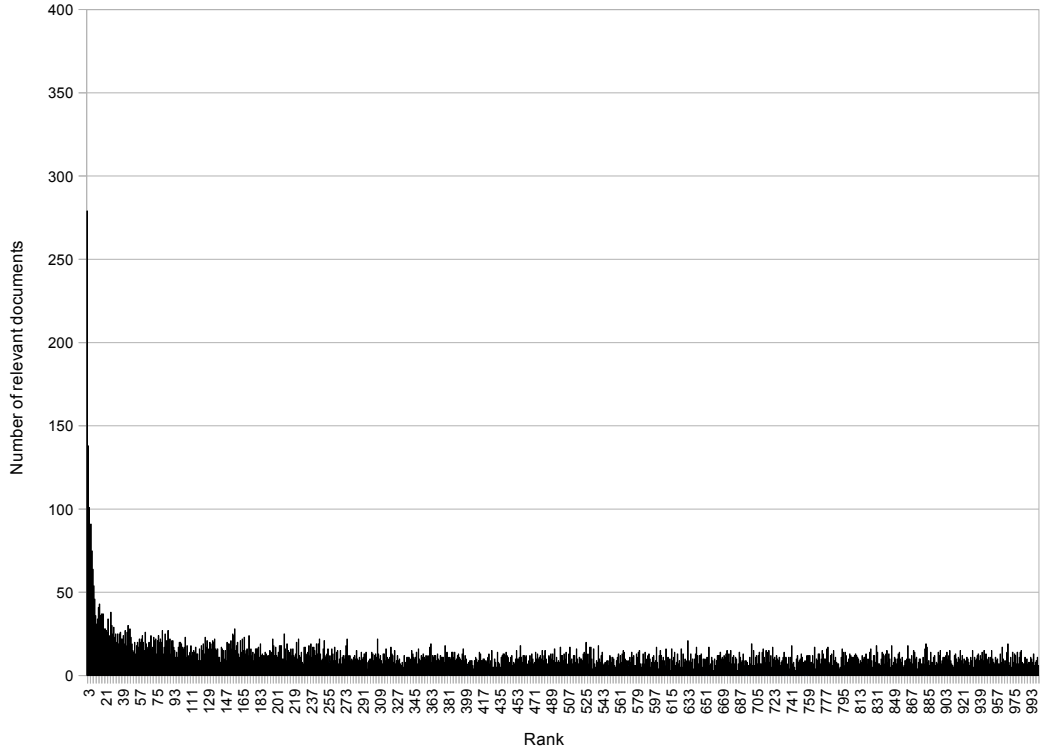


Figure 4.3.: Frequencies of relevant documents in result sets of *Terrier.tfidf.high*

2. The relevant documents are not equally distributed. Compared to [12] (see also section 2.5.1) where the amount of low and high results taken was relative to the overall ratio of low/high documents, it could make a difference to see for each retrieval model up to which rank most relevant documents show up:
 - a) Only a small amount of result lines from the low result set will be merged with the high result lines, again up to 1000 results.
 - b) This small amount has to contain more relevant documents than the high result lines that are cut off in the merging:

$$\#(relDocs_{highBottom}) \ll \#(relDocs_{lowTop})$$

To comply with point 2 of the assumptions, the low result sets were analyzed regarding their peak in relevant documents. One idea of working with an average score - average for a model or for a query - were not further investigated because the average rank for an average score was always close to position 400 and for a few cases higher. The two most interesting methods were:

- **Median:** The median of relevant documents in the low corpus was calculated using the relevant document distribution described above. Because relevant documents

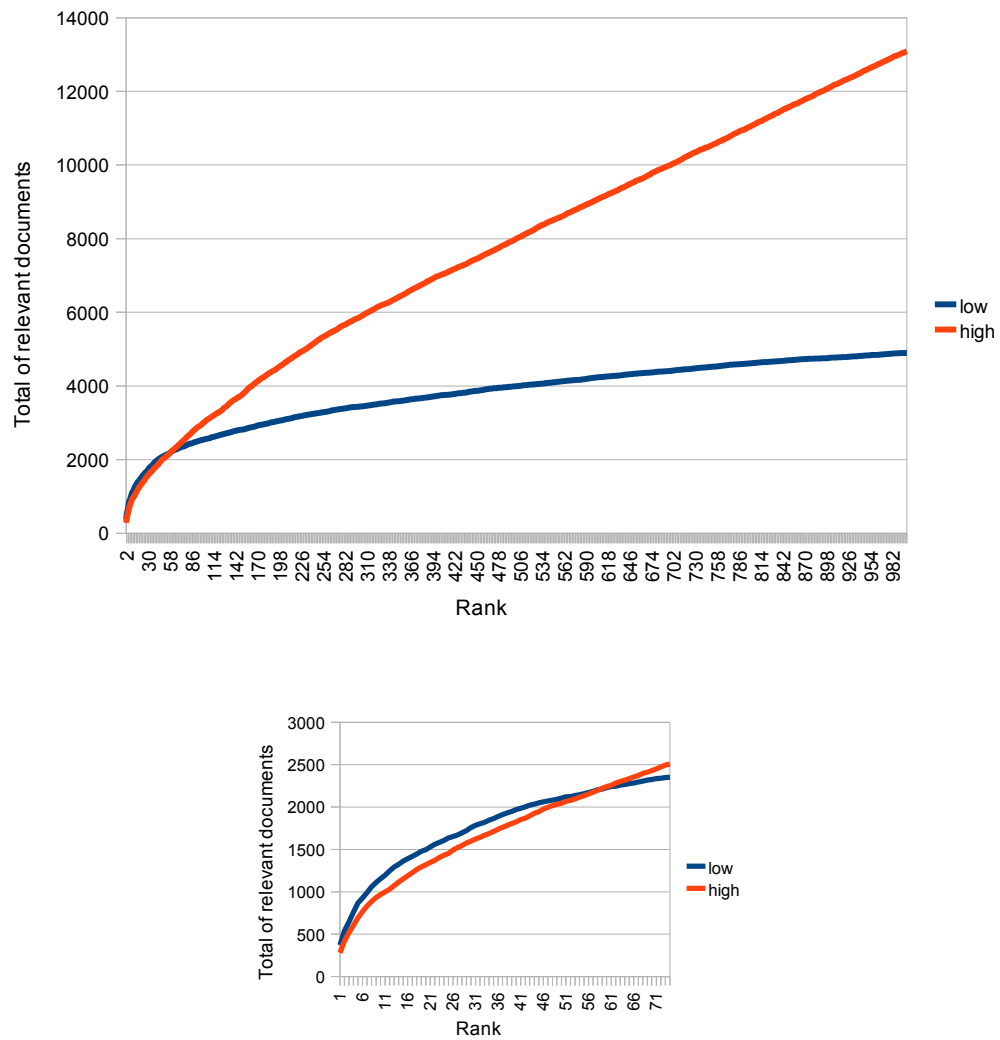


Figure 4.4.: Total of relevant documents per rank for Terrier.tfidf *low* and *high* - overview and detail

in the low distribution tend to be high in the beginning and stagnate with higher rank, the median provides on average a suitable point in the top ranks up to which many relevant documents occur.

- **Maximum relevant retrieved (maxRR):** To fulfill the requirement stated in point 2.b above, experiments were made that calculated the position in the low result list on which a merging will produce a maximum amount of relevant documents retrieved.

The found cut-off ranks of these two methods are illustrated in table B.1. The median can be determined for a single position, whereas the maximum of relevant retrieved documents can be found on multiple positions in the result set. These values were used in the experiments with the method described in section 4.4.

It is worth mentioning that most of the *maxRR* values were close to the *median* values except for a few outliers. Amongst those were also the Terrier models that had to be run with *ignore.low.idf.terms=true*, namely PL2, IFB2, DLH, BB2. However, median and maxRR rank positions of some other retrieval models also diverged widely. Only for Lemur.BM25 an answer for this behavior could be found: The relevant document distribution in both splittings was unusual, as can be seen in figure 4.5. Virtually no relevant documents were found on positions 1 to 66 (bm25), respectively 1 to 44 (tfidf), which stands in contrast to a small set of ranks on which the relevant documents accumulated. These findings might be of interest for further research.

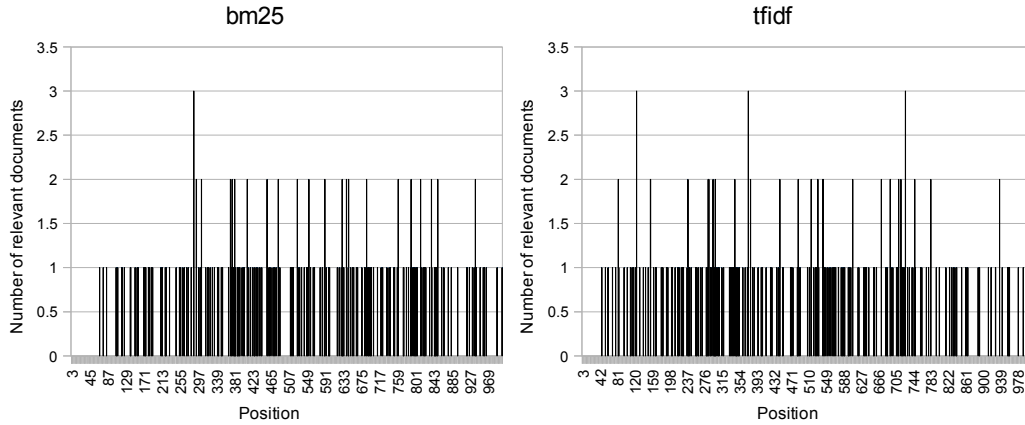


Figure 4.5.: Relevant document distribution of Lemur’s BM25 model for both splits

4.4. Result set merging

The result set merging started off by implementing an *equal_size* and *partition_size* merging algorithm similar to the methods used in [11] (see also section 2.5.1). In [11] the *partition_size* method used the size of the low/high distribution for its partitions. Because in this work the number of low/high documents is almost the same for both splittings (see above, section 4.2), this approach is not much different from the *equal_size* method. Therefore a simple *equal_size* merging method (*eqS*) was implemented along with a *partition_size* similar approach (*parS*) that combines the result sets in a ratio specified by the user. Both approaches merge on a rank based approach; eqS is a kind of *Round Robin Merging*.

Additionally to these methods, the program implemented for this work contains other optional features. Below the main features and the program's layout are described. Listings with code fragments shall provide information about the basic algorithms used for the program features. They include Java syntax because the program was written in Java, but the functionality is not bound to any programming language.

4.4.1. CombineTrecResults

The main program part is called *CombineTrecResults*. A state diagram describing the application flow is shown in figure 4.6.

Options

The program implemented can be called by using the following arguments, which should partly also be used with another program implemented for the same purpose:

- eqS** *Combine with equalSize* - Takes 500 result lines of the low and high result file from each query. If a query does not contain that much result lines the missing lines are filled up by the other result file.
- parS** *Combine with partitionSize* - Needs the **-high N** and **-low M** arguments provided. It takes *N* result lines from the high result list and *M* result lines from the low result list of each query.
- all** *Combine with equalSize and partitionSize*
- highSim** *Combine with highestSimilarity method*
- normalization** *Normalize scores before sorting*
- opMap** *Use optimalMap feature*
- high N** *N* is the number of lines taken from the high result file
- low M** *M* is the number of lines taken from the low result file

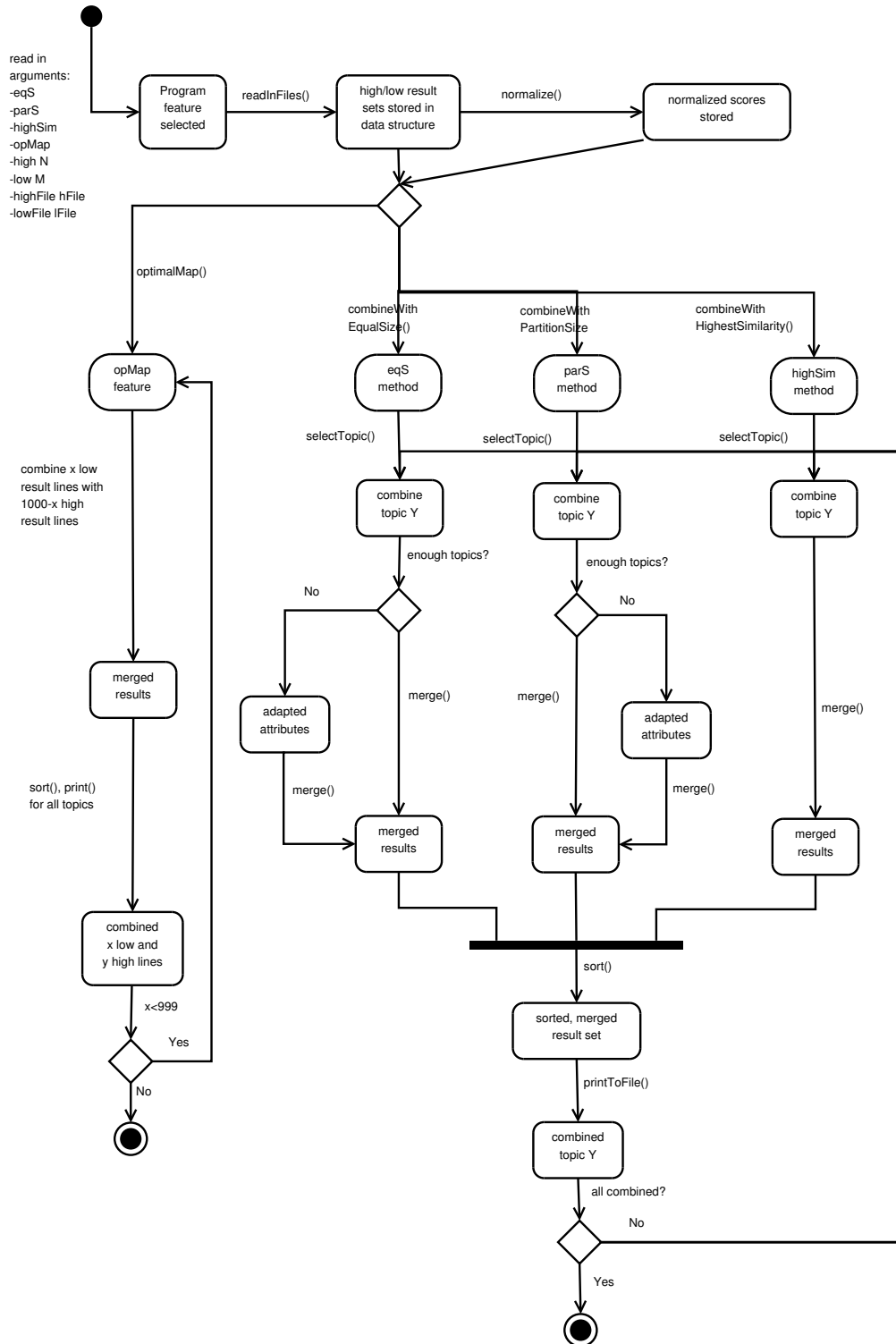


Figure 4.6.: CombineTrecResults application flow

-highFile highFile *highFile* is the path to the result set file that was generated from the high corpus

-lowFile lowFile *lowFile* is the path to the result set file that was generated from the low corpus

4.4.2. Methods of CombineTrecResults

Equal size method (eqS)

A constant, that is set to 500, determines how many result lines have to be taken from each result set. Because some queries returned less than 500 queries or none at all, certain validations were implemented:

- If a low or high result set does not contain any queries, this query is sorted out and a corresponding message is printed.
- If one file list contains less than 500 entries, the missing lines are taken from the other one.
- However, if both file lists contain less than 500 entries, the query is sorted out.

The code in listing A.5 shows the merging and fill-up (equivalent to the second point above) part. For merging, a for-loop puts an equal number of *ResultLine* elements from a low and high result set in a list. If the fill-up part gets invoked, meaning if one result list does not contain enough elements, the already partly merged list gets further elements by specifying at which position the first loop stopped and from which element list the missing result lines are taken.

After merging regarding these requirements, the result set gets sorted by its scores, because for the evaluation with *trec_eval* the scores have to be in order. The new result set is written to an output file.

Partition size method (parS)

For each query N result lines of the high result set and M lines of the low set are combined. Again, to cope with queries that return less than N respectively M results, the missing lines are taken from the other file. If both result sets contain less than the required number of result lines, the query is sorted out. This also happens if one result set does not contain any result line at all. Sorting out can be avoided by taking less result lines, because the parS method does not necessarily return 1000 (merged) result lines. This was implemented on purpose to provide a method that only returns e.g. 100 result lines, which are more realistic to be searched through by a human person than 1000.

In contrast to the eqS method, the parS method checks first if enough results from high/low are available and adapts the amount of result lines taken from each list accordingly. The lines for the merging are shown in listing A.6. After the merging the result set also gets sorted by its scores and the new result set is written to an output file as well.

Highest similarity method (highSim)

The *highestSimilarity* method combines all results of the high and low lists, then sorts these results by score in descending order. Afterwards it cuts off at 1000 results and outputs these to a new result set. With this, only the documents with the highest scores are put into the final result set, regardless how many are high and low. This method was implemented to maximize the number of documents that score very well. The source code for this method can be found in listing A.7.

Optimal MAP feature (opMap)

The *optimalMap* method combines the low and high result set with every combination possible, starting with 999 high results and 1 low result up to 1 high result and 999 low results. For each combination it calls *trec_eval* and reads out MAP, r@10, r@100, r@1000, p@10, p@100, p@1000 and RR values, which are written into an output file. A code fragment for this containing the main algorithm can be found in listing A.8. The output text file is not further formatted, so its columns are according to the value enumeration and its lines correspond to the number of low result lines (e.g. line 58 is the evaluation of a merge with 58 low and 942 high result lines). With this feature it is possible to follow the progress of specific values throughout different merging combinations.

Normalization methods

Experiments were made with several score normalization methods. There are two basic normalization methods with which experiments were made, both of which are applied to either the entire model or each query, which makes four normalization possibilities:

- Standard normalization: As in [34] this normalization distributes the scores between 0 and 1, which is done according to [6] by the formula: $s' = \frac{s - \min}{\max - \min}$ with s being the original score, s' the new, normalized score, and \max/\min the maximal/minimal score throughout the model or query. This was used in methods:
 - *normalizeScoresPerModel* - Normalization is done for the entire model, meaning over all topics of a run, as suggested in [6].
 - *normalizeScoresPerQuery* - Normalization is done for one query only, meaning that the scores of each query are then distributed between 0 and 1.

Because the worst performing document scores are cut off, entries with score 0.0 are not in the final result set.

- Sum normalization: As in [34] this normalization produces a minimum score of 0 and all other scores sum up to 1. For this the list of all scores is summed up and the normalized score is calculated with $score' = (\frac{1}{sum}) * score$, with sum being the sum of the original scores. This was used in methods:
 - *normalizeScoresSumPerModel* - Normalization is done for the entire model, meaning over all topics of a run, as suggested in [6].

– *normalizeScoresSumPerQuery* - Normalization is done for one query only.

Again the worst performing document scores are cut off, so entries with score 0.0 are not in the final result set.

An example code fragment for the score normalization per query of high result set lines can be found in listing A.9.

Summary

This chapter provided information on the methods applied for the goal of this work. For this it described the *MAREC-400k* data collection and how topics and relevance assessments out of direct and extended citations were generated. Furthermore, the parsers used to produce text files that could easily be indexed by all three retrieval engines were explained. The parameter files that were used for Terrier, Lemur and Solr were discussed and it was shown which arguments were used for indexing and that for retrieval the TREC format was crucial. Afterwards the results of many retrieval models produced by applying these parameters were provided, to give a baseline to compare to later on. Then the focus was on how to split a corpus with different split models and what characteristics the resulting low and high indexes, respectively result sets, had in order to work with them later on in the merging step. This merging, along with the implemented program parts that were used, was also discussed and highlighted the obstacles and versatile methods implemented to retrieve a better result set than with normal merging.

5. Experimental results

The experimental results provide the aggregation of all previous work, including the splitting of the corpus, the merging and the evaluation with previously generated judgments to create a final assessment to compare to baseline results. For all this, the first step in section 5.1 is to provide a short recap of the methods used and to give an overview of the models used in the experiments. Preceding the actual experiments, tests with different normalization methods will be reported on in section 5.2. Then the equalSize and highestSimilarity result sets will be discussed in sections 5.3 and 5.4. Eventually section 5.5 contains the most comprehensive experiments that were made with the partitionSize feature, including a comparison with the baseline as well as with the eqS and highSim feature.

5.1. Overview

Experiments for result set merging were performed using the methods described in section 4.4.2, namely:

- **eqS** - equalSize method
- **parS** - partitionSize method, with median and/or maximum relevant retrieved (*maxRR*) positions (cf. section 4.3), depending on model
- **highSim** - highestSimilarity method
- **norm** - normalization methods combined with the above mentioned methods
- **opMap** - optimalMap feature; was used to give an overview of all merging possibilities if the progress of certain values was interesting

Because not all methods could be tested with all retrieval models of the three retrieval engines, especially not with four different normalization methods that could be applied to each of them, a few models were picked and will be referred to as the *standard (test) set*:

- *bm25.BM25* - bm25 split and BM25 model; because a split in high/low was especially done for this model, this combination should provide good results; can be done with Terrier and Lemur, but was dropped for Lemur most of the time (see below for more information)
- *tfidf.TFIDF* - tfidf split and TFIDF model; again, a split in high/low was done for this model so good results are expected; can be done with both Terrier and Lemur

- *terrier.Hiemstra_LM* - one of the models that ranked low results higher, see also section 4.3; the tfidf split provided better results in pre-tests, so the bm25 split will not be taken into account
- *terrier.DLH13* - the second model that ranked low results higher; the tfidf split provided better results in pre-tests, so the bm25 split will not be taken into account
- *Solr.standard* - the standard Solr model, because it performed better than the mlt functionality in pre-tests

In general these models were used to provide an overview of the result set merge for each method. Additional models will be used in the parS method. Although Lemur's BM25 model did not seem to be working correctly (see section 4.3), it was also used in the parS method because the results were interesting to compare. In contrast to the baseline results, where two different topic sets were used, the experiments were only done with one topic set (*Topics I*).

5.2. Normalization

The normalization with different methods is demonstrated at first to give a basis for choosing the best performing normalization method. During development several experiments using different merging strategies were already done with the four normalization methods. `NormalizeScoresPerQuery` performed best most of the time. To test this proposition the *standard set* was used to give an overview of the normalization methods. The experiments were performed for each model with the eqS method to avoid the problem of choosing a specific position as it is necessary in parS. Table B.4¹ contains the results for several runs, each named in format *retrievalEngine.split.model.normalization*, where *normalization* can be (cf. section 4.4.2):

- *normM* - `normalizeScoresPerModel`
- *normQ* - `normalizeScoresPerQuery`
- *normSumM* - `normalizeScoresSumPerModel`
- *normSumQ* - `normalizeScoresSumPerQuery`

On average the normQ normalization achieved the best results again. Single values were sometimes better in other normalizations but overall it could score better than the other methods or its values were not far away from the better performing ones. Additionally to the results of the preceding tests, normQ seems to be the optimal choice in most cases and will therefore be the method of choice for further experiments.

¹The column *relevant retrieved* is not listed because only the order of the documents differ for each normalization, not the number of retrieved documents.

5.3. equalSize method

As described in section 4.4.2 the eqS method combines 500 low and 500 high result lines of each query to an output result set. Table B.5 contains the result values of the standard set combined with this method. The format of each model's name is again set to *retrievalEngine.split.model<.normalization>*.

Emphasized values in table B.5 highlight better results compared to the baseline results in table B.2. Only two models, *t.Hiemstra* and *t.DLH13* which stood out from the other models regarding their low and high scoring, showed better MAP values. Hiemstra also achieved better results with low precision/recall@*k* values, but on the other side it found less relevant documents compared to the baseline, which is the only model in this comparison. All other models found more relevant documents in total, nevertheless their r@1000 value is lower compared to the baseline. This characteristic can be found if only a few queries score much better whereas many others perform worse. The r@*k* value is calculated per topic and averaged over all topics, so only a few good performances will not distort the result. This problem is inherent in the RR value which seems to be much better than in the baseline results but has to be contrasted with r@1000 in this case. A discussion on this behavior can be found in section 6.1. On average the p@1000 values were lower in the baseline, although the MAP values did not perform that well. p@100 also did not achieve better results in the merged result set. This implies that between rank 100 and 1000, or for the tfidf models between 30 and 1000, the precision increased but not to an extent that could affect the MAP value.

5.4. highestSimilarity method

Details to the highestSimilarity method can be found in section 4.4.2. It basically combines all low and high result lines, sorts them and returns the best thousand entries. In table B.6 the results of the standard set with highSim combination can be found. As before, emphasized values highlight better results compared to the baseline.

Again as in the results of eqS, DLH13 and Hiemstra stand out having a better average precision (MAP) in the non-normalized run. Hiemstra can once again score better results in lower precision/recall@*k* fields, but the number of relevant documents that were found is lower. With higher p@1000 and RR values on average the highestSimilarity method is similar to the equalSize method's performance. Again it is important to look at the RR values in contrast to r@1000, because a high number of found documents in total does not imply a high average recall. In section 6.1 this behavior is discussed further. As seen in the eqS results in section 5.3 the p@1000 values of almost all models are higher compared to the baseline results. Again this implies that the precision above rank 100 is higher on average.

5.5. partitionSize method and optimalMap

The partitionSize method parS combines a predefined number of low and high result lines (cf. section 4.4.2). Because of the analysis of the split corpora in section 4.3, the two methods *median* and *maximum relevant retrieved (maxRR)* were found to be the basis for the ratio of a low/high merge. Values for merging are always related to the number of low documents, as seen in table B.1. The number of high documents is calculated by subtracting the number of low documents from 1000, so that the common result set for a query is 1000 again.

Tests were made with both methods together with the *optimalMap* feature to follow the progress of specific values, mainly MAP and r@1000. As expected the feature showed that ratio values that are close together (e.g. low/high: 100/900 and 150/850) do not perform much different from each other and the values are roughly the same or equal. Therefore three constraints were made:

1. If a model has more than one maxRR position, then the position with the highest distance to the median is used.
2. Median and maxRR positions that are close together will not both be listed in the resulting table unless the values are remarkably. Only the better performing method is listed.
3. If the difference of median and maxRR positions is high but the resulting values are almost equal, only the (slightly) better performing is listed.

Because the parS method is the most comprehensive feature and provides many merging possibilities all models from the three retrieval engines are listed, for each split (bm25/tfidf), with the original score (referred to as *raw score*) and one normalized score (normQ).

The result tables use the format *model.[med/max].position<.normalization>*, which indicates if the method used is median or maxRR, at which position the low result set was cut off (cf. table B.1) and if the run was done with normalization. Emphasized cells denote higher values compared to the baseline runs.

5.5.1. Comparison to baseline and different splits

The results of the different retrieval models can vary widely, as for normalization, split and median/maxRR method. Therefore a generalizing evaluation of the results per retrieval engine is substituted by a detailed explanation of each model's performance.

Terrier models

The results of all Terrier models can be found in table B.7 for the bm25 split and table B.8 for the tfidf split. Below are descriptions of the results for each model.

DLH13 Only the median position of DLH13 was used because the maxRR position was too close for the bm25 split, respectively the values were not much different for the tfidf split. It is the only model, respectively one of two models, that could achieve a better MAP value compared to the baseline results and also one of the two models suggested to be more promising than the others. In both splits the normQ feature is inferior to the raw score. According to the MAP and RR values the tfidf split seems more suitable for this model, but when including other measures a preference for one split cannot be made.

DLH Both median and maxRR positions showed interesting results for DLH. The bm25 split's most interesting values can be found in high $p@k$ measures and RR. Nevertheless, $r@1000$ is not higher than the baseline value, which implies the existence of only a few, maybe exceptionally, better performing queries. The tfidf split on the other hand can score much better with $p@k$ for low k values, although interestingly $p@20$ makes an exception for all runs. Deciding if bm25 or tfidf is the better split cannot be made by looking at the MAP and RR values or other precision/recall values because these are too diverse. The differences between normalization and raw score are also not significant enough to recommend normQ.

Hiemstra Hiemstra was suggested to be promising compared to other models because of its outstanding low/high scores in the split corpora (cf. section 4.3). Looking at the tfidf result table the difference to the others can be seen. It is one of only two models achieving a higher MAP value compared to the baseline results, but only for one split (DLH13 achieved higher MAP values in both splits). The tfidf split seems to be more preferable because there $r@10$ as well as $p@20$ or $p@30$ are higher than in the baseline. Also most of the other values are slightly higher than in the bm25 split. Regarding normalization, the raw score seems to work better with this model.

BB2 The values of BB2 for median and maxRR positions were almost identical, so only median was used. In both splits normQ is superior to the raw score method and the normalization also helps to increase the values of $p@k$ for $k \geq 30$ compared to the baseline results. The tfidf split seems to be performing slightly better than the bm25 split, especially when comparing MAP and RR values.

BM25 The bm25 split was actually done for this model, so it was expected to perform really good in this split and poor in the tfidf split. Interestingly though, the values of almost all measures are higher in the tfidf split, even the number of RR. Regarding normalization, normQ is superior to the raw score method in both splits. Altogether the model does not perform well compared to the baseline, only $p@1000$ and RR are higher which is a constant for all Terrier models.

DFR_BM25 The *Divergence from Randomness* for BM25, as the pure BM25 model, was also expected to have better results in the bm25 split. Because the median and

maxRR positions were calculated per split and the position values as well as the values of all measures for this model were close together, only the maxRR position for the bm25 split and the median position for tfidf were put into the result list. Both splits do not exceed the values of the baseline, except for the usual p@1000 and RR measure. Interestingly, almost all values of the tfidf split are again higher than in the bm25 split. In both splits normQ is superior to the raw score method.

IFB2 The IFB2 model's values are diverse regarding the bm25 split and the median respectively maxRR positions. The median position results in better r@1000 values, whereas the maxRR position has better RR values as expected. Both splits have in common superior precision values for low k for the normalization method compared to the baseline. Regarding a preference for one split, the differences are too diverse to recommend one of it.

InL2 The InL2 model was merged at the median position in split bm25 and at one maxRR position for the tfidf split. The normalization normQ is the superior method used here, resulting in higher values for low p/r@ k measures and equal values in higher measures. The tfidf split seems to perform slightly better than bm25 for this model. Other values than the usual p@1000 and RR values are not higher compared to the baseline result.

In.expB2 This model was also run with different position methods: split bm25 with maxRR position, split tfidf with median position. Comparison of both splits shows that except for measure r@1000 the tfidf split seems to be preferable for this model. It also includes two measures that performed better in the merging than in the baseline results. However, the values do not differ much. Regarding normalization, normQ can be recommended for this model.

In.expC2 This model does not exceed the baseline result values by merging low and high result sets, except for the usual p@1000 and RR measures. The tfidf split performs on average slightly better than bm25. Normalization with normQ can be recommended for this model because it outperforms the raw score method.

LemurTF.IDF As this model implements Lemur's TFIDF model it was expected to perform much better with the tfidf split. MAP and low r@ k values can support this theory, however the precision values are equal between bm25 and tfidf or only slightly different in favor of tfidf. Both splits include better results compared to the baseline for r@100 and p@100, p@1000 and RR are higher in the merged result set as usual. Normalization can be recommended for the model, especially for low p/r@ k measures.

PL2 For PL2 only the median positions were used because the results of the maxRR positions were not much different. Comparison of both splits shows tfidf is superior to bm25 for this model. Especially r@100 is higher than in the baseline results. On

the other hand, normalization cannot be recommended in general. For higher precision measures the raw score method performs slightly better, in both splits.

TF_IDF As counterpart to the BM25 model, this model was expected to perform much better in the tfidf than in the bm25 split. This is true for MAP and RR values, but when looking at the recall values, it only performs slightly better. Low precision measures are even inferior in the tfidf split. Nevertheless, this split achieves one better result ($p@100$) compared to the baseline. Other values are not remarkable. Regarding normalization normQ can be recommended for this model because it increases $p@k$ and $r@k$ for low k .

Lemur models

The results of all Lemur models can be found in table B.9 for the bm25 split and table B.10 for the tfidf split. Below are descriptions of the results for each model.

BM25 As mentioned in section 4.3 the distribution of relevant documents in both splits was unusual for Lemur’s okapi model. Because of this a similar and extraordinary performance was expected. The model’s median and maxRR positions, which are much higher compared to other models, can be explained by its unusual behavior. Nevertheless, BM25 showed an impressive increase of the recall value. $r@1000$ in the baseline is only 0.1687, in the merged result set 0.3031 for the tfidf split. However, the baseline value is very low compared to all other models and engines, so this suggests that the okapi model did not work well in the baseline already. Both splits also show that almost all precision values are higher than in the baseline. Again, this could be due to problems with the model itself. The model’s performance is slightly better in the tfidf split and on average the raw score performs better than the normalization method. However, this information does not have much significance due to the distribution problems of relevant documents.

Indri Indri performed very well in the baseline results, especially regarding its high RR value, which is a multiple compared to most Terrier models. As unusual as that, its median and maxRR positions are also very peculiar: For the maxRR method only 5, respectively 3 documents (bm25/tfidf) from the low result list are taken, the rest are high results. With this ratio the RR value exceeds all Terrier and Solr results and the tfidf split, which returns 4176 relevant documents compared to 2501 in the baseline, achieves a plus of 67% relevant documents. Nevertheless, as seen before a high RR does not necessarily imply a high recall value. Except for the $p@100$ and $p@1000$ values the baseline can not be topped. Regarding all other measures, the median value seems to be better for this model. For many measures the normalization performs better, but when looking at the MAP values the raw score is definitely a better recommendation.

TFIDF TFIDF was as usual expected to perform better for the correspondent split. For most of the measures this is the case but for higher k in $r@k$ and $p@k$ it turned:

$r@1000$ is higher in the *bm25* split, $p@1000$ is equal. The MAP value on the other side is higher in the *tfidf* split, especially when comparing raw score values. In both splits the $p@100$ value is higher compared to the baseline, other values are not superior. Regarding normalization, TFIDF performs slightly better or equal when using normQ.

Solr models

The results of all Solr models can be found in table B.11 for the *bm25* split and table B.12 for the *tfidf* split. Below are descriptions of the results for each model.

Standard Solr’s standard retrieval model performs averaged compared to most other retrieval models. The usual measures, $p@1000$ and RR, are higher as in the baseline, all others do not top the baseline results. Comparing both splits the model achieves similar values, so a recommendation for one split cannot be given. Regarding normalization it seems that the higher k in $p@k$ and $r@k$ gets, the more preferable normQ gets. MAP values are almost equal between normQ and raw score.

Mlt In [59] the mlt model outperformed all other models tested there. In the baseline results its performance was quite average, with exception of the RR measure which was second best after Lemur’s *indri* model. In the merged result set on the other hand it is the only model with not a single higher value compared to the baseline. Tables B.11 and B.12 show that between standard and mlt model there is not much difference, the poor result only comes from a much better baseline. The *tfidf* split seems to produce slightly better results, but for higher k in $p@k$ and $r@k$ the differences get smaller. Normalization is only superior when looking at the MAP value, all other values are too diverse to give a recommendation.

5.5.2. Comparison to eqS and highSim

To compare the implemented methods among each other, only their merged results were taken into account albeit they may not be better than the baseline. Normalization will not be a subject because the discussion in section 5.5.1 above showed that this can only be considered for each model separately. Comparison is made with table B.5 for the *equalSize* method and table B.6 for the *highestSimilarity* method.

Interestingly both methods compared to the *parS* method show similar behavior. The *lemur.tfidf.TFIDF* model’s values do not show much difference between eqS/highSim and *parS* results, only $r@1000$ is higher for eqS/highSim whereas $p@1000$ and RR are superior in the *parS* method. A similar picture can be found for all other models as well: In *solr.tfidf.standard*, *terrier.bm25.BM25* and *terrier.tfidf.TFIDF* $r@1000$ is higher in eqS/highSim, $p@1000$ and RR are higher in *parS*. The two outstanding models *terrier.tfidf.DLH13* and *terrier.tfidf.Hiemstra* show the same behavior and additionally in all methods the MAP values are higher as in the baseline.

Summary

This chapter included the most important information of this work. First a short overview of the methods tested was presented as well as a standard test set was introduced. Because the normalization methods did not perform that different, the decision on one method, normQ, was made by comparing this method to the other ones in a few example runs. Afterwards the method eqS was tested with the standard set and compared to the baseline results. As second method highSim was also tested with the standard set and compared to the baseline. Both methods performed similar. Eventually the parS method was tested with all available models and with different low/high ratios for the merging, either with the median or maximum-relevant-retrieved position. For each model its performance in both bm25 and tfidf splits as well as compared to the baseline was discussed. Recommendations whether to use normalization or not were subject too and were not always determinable. The last comparison was made between the different methods, eqS, highSim and parS with the interesting result that both highSim and eqS performed very similar in comparison to parS.

6. Conclusions

The main part of this work was to provide merging strategies to combine the result sets of previously split corpora to a new result set. One distinctive feature that distinguishes this merge from common data fusion or collection fusion approaches is that the documents were first divided in low and high retrievable documents. Afterwards they got indexed separately and also retrieval was done individually. This produced non-overlapping result sets that had to be combined in a way that leads to having more relevant documents retrieved at lower ranks compared to the normal, non-merged results.

To come to this final result sets several preliminary steps were necessary first. The setup for the experiments was made based on another, similar work (cf. section 2.5.2) which worked with different retrieval engines and no relevance judgments and topics available initially. Relevance assessments and topics were generated out of the data collection to obtain a completely functional test collection to work with. After some adjustments and formatting of the data collection, indexes were made with three different retrieval engines, namely Terrier, Lemur and Solr. With these a baseline was built for comparison after the actual experiments. The single corpus was then split based on a retrievability evaluation of its documents in a low and a high corpus. After analyzing the split corpora regarding its relevant document distribution, several combining methods were presented with which experiments were made afterwards. One method named `partitionSize`, which combines low and high result sets in a predefined ratio, was tested more precisely because it could merge the documents in a ratio that was discussed to perform better than other methods in the analysis section. Though this method performed on average similar to the other methods presented, some models with particular features could achieve slightly better results compared to the non-merged result set. This behavior shows a tendency for certain models, which will be explained and discussed below.

6.1. Discussion and interpretation of results

It is difficult to look at the results as a whole because of the diversity of the outcomes. A detailed description of each retrieval model with regard to the `partitionSize` method can be found in section 5.5.1. The following deals mainly with the results of the `partitionSize` method in general.

r@1000 and RR

In the `partitionSize` results, two circumstances occur in almost every model: measures `p@1000` and `RR` (relevant retrieved) are higher and `r@1000` is lower than in the baseline

results. A higher $p@1000$ means that between rank 100 and 1000 (for most of the models, some also have higher values in $p@100$ or lower) the precision is higher on average than in the baseline. For models like Terrier.DLH in the tfidf split all $p@k$ values, except for $p@20$, are higher than in the baseline, so merging low and high split documents seems to increase at least the precision values for this model. However, for a recall-oriented domain like the patent domain high recall values are much more important. So the RR values combined with the $r@k$ values are more important to be higher. In the result sets that were generated by the merging methods this behavior only occurs infrequently. Usually the RR values are higher or much higher than in the result set whereas all or most of the $r@k$ values are lower. For this two explanations have been found.

First, the distribution of relevant documents in the merged result sets was analyzed. Figure 6.2 shows the distributions for the Terrier Hiemstra model in the tfidf split, without normalization. The equalSize and partitionSize method have more relevant documents on the first rank, highestSimilarity performs worse compared to all others. On average the partitionSize method has more relevant documents per rank, for it also contains the most relevant documents compared to the other methods and the baseline. For the first approximately 200 ranks it is equal to the other methods regarding the total number of relevant documents per rank. Then this number increases more for the partitionSize method than for the others; equalSize and highestSimilarity are even equal for about 800 ranks. A figure plotting this development can be found in diagram 6.1.

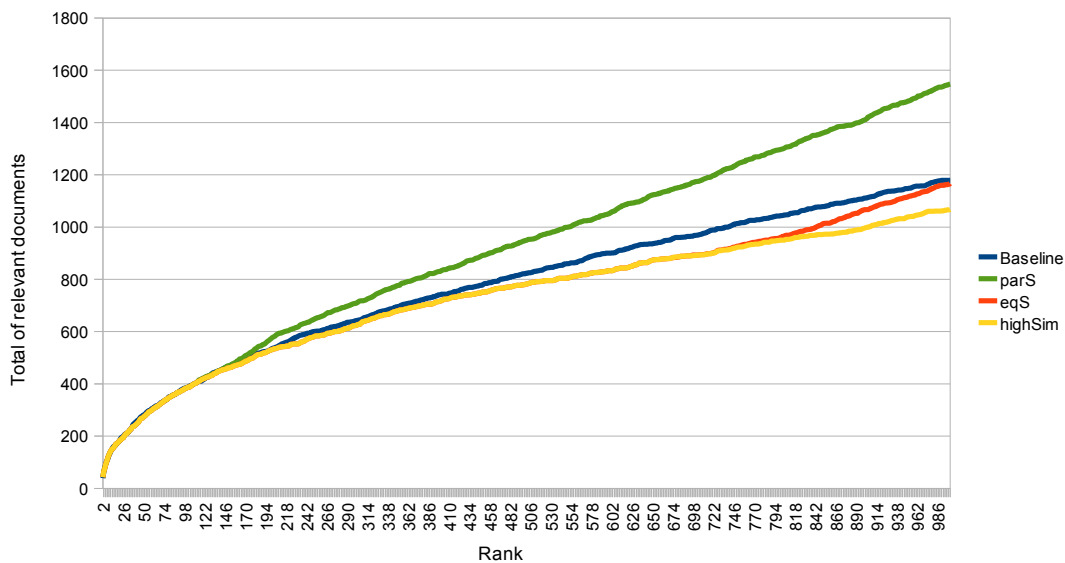


Figure 6.1.: Terrier.tfIdf.Hiemstra - Total number of relevant documents per rank

In contrast to this progression of total relevant documents per rank, the median of relevant documents can also be examined. For the baseline of Terrier.Hiemstra the median was found on rank 238, so above and below the same number of relevant documents can be found. For the partitionSize method the median could be found further down on rank

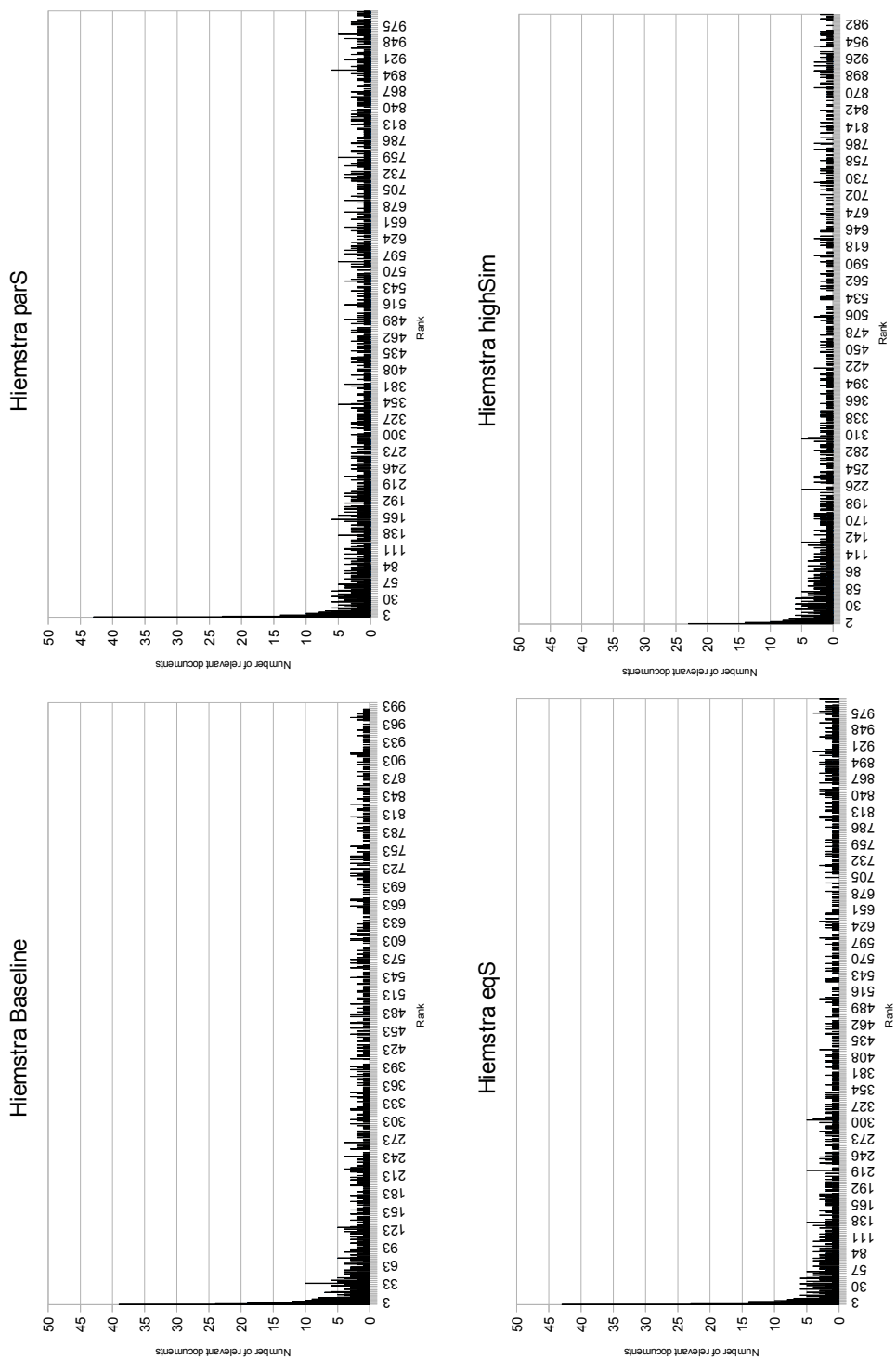


Figure 6.2.: Terrier.tfidf.Hiemstra - Overview of relevant documents distribution for baseline and different merging methods

346. So the relevant documents are spread across a wider area. The equalSize method performed better with the median being found on rank 251, whereas the highestSimilarity method could even top the baseline and had its median on rank 202. In the latter methods Hiemstra could achieve better MAP and recall values compared to the other merging methods, whereas only RR and p@1000 values were better in the partitionSize method.

As second explanation for the low r@1000 and high RR values a few, very good performing queries and many poor performing ones were found to increase one measure while leaving the other low. To test this theory, *trec_eval* was run with the option to print detailed values for each topic. Again, the tested model was Terrier.tfidf.Hiemstra without normalization. The MAP values of the baseline was 0.0753, the r@1000 value 0.4700. The following table shows how many topics score better or worse than the baseline MAP/r@1000 value, respectively how many topics had a MAP/r@1000 of 1 or 0 in Hiemstra results:

		Merge	Baseline
MAP	< 0.0753	173	172
	≥ 0.0753	59	60
	$= 1$	0	0
	$= 0$	41	39
r@1000	< 0.47	131	107
	≥ 0.47	101	125
	$= 1$	34	50
	$= 0$	17	11

Regarding the MAP value and number of topics there is not much difference between merging and baseline. However, the r@1000 distribution shows that less topics in the baseline scored below the average 0.47, whereas more topics scored better. The merging distribution shows quite the contrary. The baseline contains about five times more topics with $r@1000 = 1$ than $r@1000 = 0$. In the merging on the other side there are only twice as many topics that retrieved all relevant topics after 1000 ranks than there are topics that retrieved none.

Normalization

Regarding normalization, different normalization methods were tested and one was found to be performing slightly better than the others. Apart from the poor result of normSumQ and normSumM in Lemur.tfidf.indri the different methods performed similar. Although normalization was thought to be the superior method for merging, because most of the documents in low result sets were scoring lower compared to the baseline whereas documents in the high result set scored higher, it cannot be named the method of choice for all models. It seems to depend on the model and certainly its attributes, but this was not further examined and might be subject to further work. Only Hiemstra and DLH13 were expected to work better when using the original score because both models assigned lower scores to high retrievable documents and higher scores to

low retrievable documents, compared to the baseline. And also for these two normQ sometimes achieved better results, e.g. for the $r@1000$ measure in highSim.

maxRR and median

For the partitionSize method two positions were found to be interesting as cut off for the low result set: the median of relevant documents in the low corpus and the position in the low result list on which a merging will produce a maximum amount of relevant retrieved documents. During testing it was found out that two positions that are close together did not produce very different merged result sets. Because many maxRR and median positions are close together only one of them was put into the final result set because the values of different measures were very close or equal. Terrier's DLH model shows this consistency. Thus using the one method over the other cannot be supported, both performed similar or equal.

Promising models

Two types of promising models were found: First of all the TFIDF and BM25 models for which the splits were done originally. On the other hand Terrier's Hiemstra and DLH13 models were the only ones showing the expected scoring for low respectively high classified documents.

The first group did not perform well compared to other models. BM25 models achieved better results in tfidf splits and although for many other models the tfidf split was superior, for TFIDF the split was inferior for some measures. Maybe the method used for the split should get more analysis to learn more about it. Lemur's BM25 model can not be evaluated here because there seem to be some problems with it (cf. section 4.3).

The second group, Hiemstra and DLH13, performed much better compared to the first group. At least in one split both could increase the MAP and $r@10$ value albeit not much. However their achievement in highSim and eqS merging was considerably better: MAP, $r@10$ and $r@1000$ could be improved in some cases. An explanation for Hiemstra's performance might be the usage of language models. Bashir and Rauber also mentioned in [11] that a language modelling approach used there is the least biased retrieval system.

Summary of highSim, eqS and parS

None of the merging methods provided could improve the measures important for a recall-oriented domain more than the other. In general, the methods could only provide slightly better results for certain models regarding recall values. Some were able to increase precision values, mainly $p@1000$ achieved way better results. The problem is that although working with 1000 results is common in information retrieval experiments, humans who have to productively work with this result set might not be looking at results after rank 100. Nevertheless, the total number of relevant retrieved documents was higher for almost all models. So one important task is to get these documents in the top ranks. Ideas on how this might be achieved are subject in the next section.

6.2. Future prospects

As mentioned before, the merging method `partitionSize` probably can outperform the others if the relevant documents that are listed on low ranks can be brought to the top 100. To do so, the potential of several concepts should be further examined. First the method used for dividing high and low documents in separate corpora should be further analyzed. In this work it was only used as a black box without much insight into its operation methods. The six different grades of low and high that were provided might also be considered as basis to future work. The top and bottom 30% of high and low retrievable documents could contain more potential to divide a corpus. Because the baseline results were on average quite good, a possible experiment could be based on using the most distant low/high documents merged with the baseline. This would include having to combine documents with a data fusion technique like COMB*, as described by Zenz et al. in [59].

Another field that can be considered problematic, that forms the basis of all the experiments made, is prior-art search. A newly presented publication shows that extracting the first claim as topic, as done in this paper, might not be the best choice. Mahdabi et al. experimented in [27] with different fields of patent documents as queries and reasoned that using the description of a patent is best for extracting query terms.

A last field that can be especially problematic with a setup using smaller collections, like the low/high corpora in this work, are issues with the very specific vocabulary used in patents. As stated by Harris et al. in [23] there are several problems that come with keyword search, like words that have different meanings in other fields (homonyms), as well as synonyms on the opposite side. In a small collection, coming across this problem once can decrease important values exceptionally. As seen in section 6.1 the number of topics for which not a single relevant document was found increased in the merged, smaller result set compared to the larger baseline, whereas the number of topics for which all relevant documents were found decreased.

The sometimes uncommon usage of vocabulary in the very special domain of patent information and its consequences are difficult to evaluate. However they should always be considered to influence experiments made within this field. As Atkinson mentioned in [2] the algorithms used in retrieval tools, like Terrier, Lemur or Solr, were not primarily made for the special domain of patent information. Therefore one cannot assume they will deal with patents in patentese correctly, simply because “patentese is the language used in patents that may appear to be English” [2].

A. Listings

Listing A.1: Solr schema.xml

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2
3 <schema name="solrForSplitCorpora" version="1.3">
4
5 <types>
6
7   <fieldType name="string" class="solr.StrField" sortMissingLast="true"
8     omitNorms="true"/>
9   <fieldType name="text" class="solr.TextField" positionIncrementGap="100">
10
11     <analyzer type="index">
12       <tokenizer class="solr.StandardTokenizerFactory"/>
13       <filter class="solr.LowerCaseFilterFactory"/>
14     </analyzer>
15
16     <analyzer type="query">
17       <tokenizer class="solr.StandardTokenizerFactory"/>
18       <filter class="solr.LowerCaseFilterFactory"/>
19     </analyzer>
20   </fieldType>
21 </types>
22
23 <fields>
24   <field name="id" type="string" indexed="true" stored="true" required="
25     true"/>
26   <field name="toIndex" type="text" indexed="true" stored="true"
27     termVectors="true"/>
28 </fields>
29
30 <uniqueKey>id</uniqueKey>
31
32 <defaultSearchField>toIndex</defaultSearchField>
33
34 </schema>
```

Listing A.2: Solr client - standard http request

```
1 // for standard queries:
2 // e.g. http://localhost:8980/solr/select/?stylesheet=&q=schema&fl=score&wt
3 // =xslt&tr=SOLR2TREC.xsl
4 URI preUrl = new URI( "http", null, "localhost", 8980, "/solr/select/",
5   "stylesheet=&q=" + URLEncoder.encode( query, "UTF-8" ) +
6   "&fl=score" +
```



```

6 "&wt=xslt&tr=" + xslFile ,
7 null);
8
9 URL url = preUrl.toURL();
10 urlStream = url.openStream();
11 urlReader = new BufferedReader( new InputStreamReader( urlStream ) );

```

Listing A.3: Solr client - mlt http request

```

1 // for more like this functionality:
2 // e.g. http://localhost:8980/solr/mlt/?stylesheet=&q=schema&mlt.fl=toIndex
   &fl=score&wt=xslt&tr=SOLR2TREC.xsl
3 URI preUrl = new URI( "http", null, "localhost", 8980, "/solr/mlt/",
4 "stylesheet=&q=" + URLEncoder.encode( query, "UTF-8" ) +
5 "&mlt.fl=toIndex" +
6 "&fl=score" +
7 "&wt=xslt&tr=" + xslFile ,
8
9 URL url = preUrl.toURL();
10 urlStream = url.openStream();
11 urlReader = new BufferedReader( new InputStreamReader( urlStream ) );

```

Listing A.4: Solr to TREC xslt file

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="
   1.0">
3   <xsl:output method="text" indent="no" />
4   <xsl:template match="/">
5     <xsl:variable name="topic" select="/response/lst[@name='responseHeader
   ']/lst[@name='params']/str[@name='topic']"/>
6     <xsl:variable name="run" select="/response/lst[@name='responseHeader']/
   lst[@name='params']/str[@name='run']"/>
7     <xsl:for-each select="response/result/doc">
8       <xsl:value-of select="$topic"/><xsl:text> Q0 </xsl:text>
9       <xsl:value-of select="str[@name='id']"/><xsl:text> </xsl:text>
10      <xsl:value-of select="position()"/><xsl:text> </xsl:text>
11      <xsl:value-of select="float[@name='score']"/><xsl:text> </xsl:text>
12      <xsl:value-of select="$run"/><xsl:call-template name="Newline"/>
        xsl:call-template>
13    </xsl:for-each>
14  </xsl:template>
15
16  <xsl:template name="Newline">
17    <xsl:text>&#10;</xsl:text>
18  </xsl:template>
19 </xsl:stylesheet>

```

Listing A.5: equalSize - merging and fillup

```

1 ArrayList<ResultLine> mergedList = new ArrayList<ResultLine>( 1000 );

```

```

2
3 for ( int n = 0; n < high_eq; n++ )
4 {
5     ResultLine resLine_H = fileHighList.get( n );
6     ResultLine resLine_L = fileLowList.get( n );
7
8     mergedList.add( resLine_H );
9     mergedList.add( resLine_L );
10 }
11
12 // second part of fillup
13 if (fillup)
14 {
15     ArrayList<ResultLine> fileList = null;
16
17     if (fileHighList.size() < CONST_EQ_SIZE)
18     {
19         // fillup from fileLowList
20         fileList = fileLowList;
21     }
22     else if (fileLowList.size() < CONST_EQ_SIZE)
23     {
24         //fillup from fileHighList
25         fileList = fileHighList;
26     }
27
28     int fill = 1000 - 2*high_eq;
29
30     for (int m = high_eq; m < fill; m++)
31     {
32         ResultLine resLine = fileList.get(m);
33         mergedList.add(resLine);
34     }
35 }

```

Listing A.6: partitionSize - merging

```

1 ArrayList<ResultLine> mergedList = new ArrayList<ResultLine>();
2
3 for ( int n = 0; n < high_par; n++ )
4 {
5     ResultLine resLine_H = fileHighList.get( n );
6
7     mergedList.add( resLine_H );
8 }
9
10 for ( int n = 0; n < low_par; n++ )
11 {
12     ResultLine resLine_L = fileLowList.get( n );
13
14     mergedList.add( resLine_L );
15 }

```

Listing A.7: highestSimilarity

```

1  /*
2  * for each topicNumber/query in the high result set data structure:
3  */
4  for ( Integer i : resultMapHigh.keySet() )
5  {
6      // combine all high + low lists
7      // then sort them after their normalized score
8      // cut off at 1000 results => output result set
9
10     ArrayList<ResultLine> fileLowList , fileHighList ;
11
12     fileHighList = resultMapHigh.get( i );
13     fileLowList = resultMapLow.get( i );
14
15     /*
16     * merge together both resultlines
17     */
18     ArrayList<ResultLine> mergedList = new ArrayList<ResultLine>();
19
20     for ( ResultLine rl : fileHighList )
21         mergedList.add( rl );
22
23     for ( ResultLine rl : fileLowList )
24         mergedList.add ( rl );
25
26     /*
27     * sort the results
28     */
29
30     // sort in reverse order , because we want the highest score on top
31     Comparator<ResultLine> comparator = Collections.reverseOrder();
32     Collections.sort( mergedList , comparator );
33
34     ArrayList<ResultLine> resultList = new ArrayList<ResultLine>();
35
36     for ( int n = 0; n < 1000; n++)
37     {
38         resultList.add( mergedList.get( n ) );
39     }
40
41     for ( ResultLine rs : resultList )
42     {
43         out.write( rs.topicNr + " Q0 " + rs.ucid + " " + rs.rank + " " + rs.score
44             + " " + rs.run + "\n" );
45     }
46     outInfo.write( "Sucessfully combined topic " + i + "\n" );
47 }

```

Listing A.8: Loop for optimal MAP feature

```

1  int h = 999;
2  for ( int l = 1; l < 1000; l++ )

```

```

3 {
4   FileWriter writer = new FileWriter( "tempResSet" );
5   BufferedWriter out = new BufferedWriter( writer );
6
7   for ( Integer i : resultMapHigh.keySet() )
8   {
9     ArrayList<ResultLine> highlist = resultMapHigh.get( i );
10    ArrayList<ResultLine> lowlist = resultMapLow.get( i );
11
12    ArrayList<ResultLine> mergedList = new ArrayList<ResultLine>();
13
14    for ( int n = 0; n < h; n++ )
15    {
16      if ( highlist.size() <= n )
17        break;
18
19      mergedList.add( highlist.get( n ) );
20    }
21
22    for ( int n = 0; n < l; n++ )
23    {
24      if ( lowlist.size() <= n )
25        break;
26
27      mergedList.add( lowlist.get( n ) );
28    }
29
30    Comparator<ResultLine> comparator = Collections.reverseOrder();
31    Collections.sort( mergedList, comparator );
32
33    for ( ResultLine rs : mergedList )
34    {
35      out.write( rs.topicNr + " Q0 " + rs.ucid + " " + rs.rank + " " + rs.
36        score + " " + rs.run + "\n" );
37    }
38    out.close();
39
40    [...]
41    // call trec_eval for temporary file
42    // read + write out map, r@X, p@X, rr values
43    [...]
44
45    h--;
46  }

```

Listing A.9: Example normalization per query

```

1 for ( Integer topicNr : resultMapHigh.keySet() )
2 {
3   // for each query determine max and low scores and normalize with them all
4   // scores of that query
5   ArrayList<ResultLine> allTopicResults = resultMapHigh.get( topicNr );
6   ArrayList<Double> scoreHelper = new ArrayList<Double>();

```

```
6
7  for (ResultLine rl : allTopicResults )
8      scoreHelper.add( rl.score );
9
10 maxHighScore = Collections.max( scoreHelper );
11 minHighScore = Collections.min( scoreHelper );
12
13 for ( ResultLine rl : allTopicResults )
14 {
15     rl.normScore = ( rl.score - minHighScore ) / ( maxHighScore -
16         minHighScore );
17 }
```

B. Tables

Table B.1.: Position in low result set for *median* and *maximum of relevant retrieved*

Split	Model	low position of median	low position of maxRR
<i>Terrier bm25</i>	TFIDF	71	84, 85, 88, 89
	PL2	133	45, 48, 52, 57
	Lemur TFIDF	77	43
	InL2	73	76, 77, 84
	InExpC2	75	65, 66, 67
	InExpB2	77	94, 95, 96, 111, 112, 113
	IFB2	120	58, 60
	Hiemstra	79	67
	DLH13	83	72
	DLH	114	65
	DFR_BM25	73	101
	BM25	76	79, 91
	BB2	98	58, 59, 60, 61, 62
<i>Terrier tfidf</i>	TFIDF	78	79, 80, 81, 82, 83, 84, 86, 87, 88, 89, 90, 91, 93
	PL2	133	47
	Lemur TFIDF	81	59, 60
	InL2	76	82, 90, 92, 93, 94, 128
	InExpC2	80	56, 57, 58, 60, 67, 68, 69
	InExpB2	77	58
	IFB2	118	61, 62
	Hiemstra	79	51, 52, 60, 61, 63
	DLH13	78	44
	DLH	111	47, 48, 49
	DFR_BM25	71	75
	BM25	73	78, 81, 82
	BB2	107	69, 71, 72
<i>Lemur bm25</i>	BM25	526	443, 444, 445, 446
	indri	86	5, 7
	TFIDF	92	108, 109, 110, 112, 113, 115, 116
<i>Lemur tfidf</i>	BM25	482	566, 599, 600, 607, 608, 609
	indri	95	3
	TFIDF	98	100, 109, 112
<i>Solr bm25</i>	standard	74	99
	mlt	117	102, 103
<i>Solr tfidf</i>	standard	76	60, 61, 62
	mlt	124	72, 85, 87, 89

Table B.2.: Baseline runs - Topics I

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
t.BB2	0.0533	0.0795	0.2180	0.3803	0.0429	0.0298	0.0239	0.0122	0.0036	832
t.BM25	0.0741	0.1064	0.2454	0.4147	0.0554	0.0371	0.0295	0.0138	0.0033	774
t.DFR_BM25	0.0739	0.1055	0.2463	0.4125	0.0545	0.0369	0.0295	0.0138	0.0033	759
t.DLH	0.0575	0.0894	0.2201	0.3711	0.0476	0.0337	0.0263	0.0140	0.0051	1191
t.DLH13	0.0706	0.1064	0.2413	0.3930	0.0554	0.0395	0.0312	0.0141	0.0039	905
t.Hiemstra_LM	0.0753	0.1184	0.2949	0.4700	0.0605	0.0386	0.0306	0.0165	0.0051	1179
t.IFB2	0.0503	0.0859	0.2193	0.3799	0.0464	0.0298	0.0242	0.0130	0.0037	872
t.In_expB2	0.0663	0.0985	0.2476	0.4304	0.0536	0.0373	0.0293	0.0142	0.0037	864
t.In_expC2	0.0672	0.0972	0.2536	0.4262	0.0536	0.0386	0.0299	0.0149	0.0037	872
t.InL2	0.0688	0.1064	0.2421	0.3973	0.0545	0.0358	0.0288	0.0128	0.0031	727
t.LemurTFIDF	0.0653	0.0974	0.2503	0.4265	0.0549	0.0388	0.0299	0.0141	0.0035	817
t.PL2	0.0552	0.0860	0.2159	0.3724	0.0446	0.0320	0.0252	0.0124	0.0035	812
t.TFIDF	0.0698	0.1068	0.2313	0.3931	0.0545	0.0365	0.0283	0.0124	0.0031	712
l.indri	0.0701	0.1018	0.2278	0.3754	0.0567	0.0380	0.0300	0.0148	0.0107	2501
l.TFIDF	0.0762	0.1117	0.2464	0.4158	0.0597	0.0397	0.0305	0.0134	0.0034	795
l.BM25	0.0403	0.0631	0.1158	0.1687	0.0300	0.0182	0.0139	0.0060	0.0011	263
s.standard	0.0671	0.1004	0.2375	0.3933	0.0515	0.0354	0.0280	0.0135	0.0033	768
s.mlt	0.0623	0.0912	0.2183	0.3817	0.0472	0.0343	0.0270	0.0153	0.0073	1700

Table B.3.: Baseline runs - Topics II

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
t.BB2	0.0531	0.0785	0.2180	0.3793	0.0425	0.0298	0.0239	0.0122	0.0036	831
t.BM25	0.0739	0.1053	0.2454	0.4147	0.0549	0.0373	0.0295	0.0138	0.0033	774
t.DFR_BM25	0.0738	0.1045	0.2463	0.4125	0.0541	0.0371	0.0295	0.0138	0.0033	759
t.DLH	0.0571	0.0884	0.2201	0.3711	0.0472	0.0335	0.0262	0.0140	0.0051	1191
t.DLH13	0.0706	0.1074	0.2413	0.3930	0.0558	0.0393	0.0312	0.0141	0.0039	905
t.Hiemstra_LM	0.0751	0.1173	0.2949	0.4700	0.0601	0.0384	0.0306	0.0165	0.0051	1179
t.IFB2	0.0500	0.0859	0.2193	0.3788	0.0464	0.0298	0.0242	0.0130	0.0037	871
t.ln_expB2	0.0660	0.0985	0.2476	0.4294	0.0536	0.0373	0.0293	0.0142	0.0037	863
t.ln_expC2	0.0670	0.0972	0.2536	0.4251	0.0536	0.0386	0.0299	0.0149	0.0037	871
t.lnL2	0.0692	0.1053	0.2421	0.3973	0.0541	0.0358	0.0289	0.0128	0.0031	727
t.LemurTFIDF	0.0646	0.0963	0.2503	0.4265	0.0545	0.0391	0.0299	0.0141	0.0035	817
t.PL2	0.0550	0.0849	0.2159	0.3724	0.0442	0.0318	0.0250	0.0124	0.0035	812
t.TFIDF	0.0702	0.1057	0.2313	0.3931	0.0541	0.0365	0.0285	0.0124	0.0031	712
s.standard	0.0622	0.0999	0.2346	0.3856	0.0511	0.0339	0.0270	0.0130	0.0032	753
s.mlt	0.0525	0.0750	0.1887	0.3349	0.0416	0.0300	0.0255	0.0152	0.0074	1735

Table B.4.: Normalization results for several runs

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000
l.tfidf.indri.normM	0.0684	0.0997	0.2215	0.3777	0.0524	0.0352	0.0279	0.0147	0.0083
l.tfidf.indri.normQ	0.0577	0.1005	0.2275	0.3777	0.0532	0.0373	0.0280	0.0144	0.0083
l.tfidf.indri.normSumM	0.0012	0.0000	0.0096	0.3777	0.0052	0.0039	0.0037	0.0045	0.0083
l.tfidf.indri.normSumQ	0.0012	0.0000	0.0081	0.3777	0.0039	0.0034	0.0034	0.0043	0.0083
s.bm25.standard.normM	0.0634	0.0908	0.2257	0.3983	0.0502	0.0352	0.0270	0.0134	0.0034
s.bm25.standard.normQ	0.0556	0.0889	0.2283	0.3983	0.0476	0.0345	0.0279	0.0133	0.0034
s.bm25.standard.normSumM	0.0653	0.0976	0.2290	0.3983	0.0515	0.0352	0.0278	0.0133	0.0034
s.bm25.standard.normSumQ	0.0652	0.0976	0.2290	0.3983	0.0511	0.0350	0.0276	0.0133	0.0034
t.bm25.BM25.normM	0.0398	0.0648	0.1692	0.4001	0.0373	0.0270	0.0217	0.0110	0.0035
t.bm25.BM25.normQ	0.0566	0.0912	0.2261	0.4001	0.0485	0.0330	0.0272	0.0132	0.0035
t.bm25.BM25.normSumM	0.0567	0.0871	0.2069	0.4001	0.0421	0.0294	0.0236	0.0120	0.0035
t.bm25.BM25.normSumQ	0.0580	0.0873	0.2048	0.4001	0.0421	0.0292	0.0230	0.0118	0.0035
t.tfidf.TFIDF.normM	0.0430	0.0651	0.1942	0.3886	0.0365	0.0262	0.0209	0.0114	0.0033
t.tfidf.TFIDF.normQ	0.0590	0.0974	0.2260	0.3886	0.0511	0.0341	0.0268	0.0126	0.0033
t.tfidf.TFIDF.normSumM	0.0557	0.0873	0.2208	0.3886	0.0472	0.0328	0.0263	0.0123	0.0033
t.tfidf.TFIDF.normSumQ	0.0558	0.0880	0.2208	0.3886	0.0476	0.0330	0.0263	0.0123	0.0033
t.tfidf.Hienstra.normM	0.0550	0.0904	0.2492	0.4639	0.0502	0.0361	0.0300	0.0164	0.0050
t.tfidf.Hienstra.normQ	0.0614	0.1041	0.2876	0.4639	0.0549	0.0380	0.0309	0.0167	0.0050
t.tfidf.Hienstra.normSumM	0.0734	0.1116	0.2897	0.4639	0.0584	0.0408	0.0319	0.0170	0.0050
t.tfidf.Hienstra.normSumQ	0.0734	0.1111	0.2886	0.4639	0.0584	0.0406	0.0322	0.0169	0.0050

Table B.5.: eqS results for the standard set

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
l.tfidf.TFIDF	0.0513	0.0852	0.2139	0.4121	0.0481	0.0311	0.0256	0.0132	0.0036	834
l.tfidf.TFIDF.normQ	0.0604	0.1038	0.2401	0.4121	0.0554	0.0382	0.0289	0.0138	0.0036	834
s.tfidf.standard	0.0576	0.0958	0.2136	0.3999	0.0511	0.0326	0.0262	0.0128	0.0034	798
s.tfidf.standard.normQ	0.0568	0.0950	0.2283	0.3999	0.0498	0.0345	0.0273	0.0132	0.0034	798
t.bm25.BM25	0.0352	0.0574	0.1424	0.4001	0.0318	0.0234	0.0186	0.0100	0.0035	813
t.bm25.BM25.normQ	0.0566	0.0912	0.2261	0.4001	0.0485	0.0330	0.0272	0.0132	0.0035	813
t.tfidf.DLH13	0.0718	0.1041	0.2391	0.3918	0.0554	0.0384	0.0309	0.0139	0.0041	949
t.tfidf.DLH13.normQ	0.0620	0.0927	0.2349	0.3918	0.0502	0.0373	0.0293	0.0139	0.0041	949
t.tfidf.Hiemstra	0.0760	0.1189	0.2905	0.4639	0.0609	0.0386	0.0306	0.0165	0.0050	1167
t.tfidf.Hiemstra.normQ	0.0614	0.1041	0.2876	0.4639	0.0549	0.0380	0.0309	0.0167	0.0050	1167
t.tfidf.TFIDF	0.0502	0.0765	0.2133	0.3886	0.0416	0.0292	0.0245	0.0121	0.0033	776
t.tfidf.TFIDF.normQ	0.0590	0.0974	0.2260	0.3886	0.0511	0.0341	0.0268	0.0126	0.0033	776

Table B.6.: highSim results for the standard set

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
l.tfidf.TFIDF	0.0512	0.0852	0.2139	0.3787	0.0481	0.0311	0.0256	0.0132	0.0040	927
l.tfidf.TFIDF.normQ	0.0604	0.1038	0.2401	0.4138	0.0554	0.0382	0.0289	0.0138	0.0036	833
s.tfidf.standard	0.0575	0.0958	0.2136	0.3786	0.0511	0.0326	0.0262	0.0128	0.0038	888
s.tfidf.standard.normQ	0.0568	0.0950	0.2283	0.3972	0.0498	0.0345	0.0273	0.0132	0.0034	800
t.bm25.BM25	0.0346	0.0574	0.1424	0.2609	0.0318	0.0234	0.0186	0.0100	0.0038	876
t.bm25.BM25.normQ	0.0565	0.0912	0.2261	0.3956	0.0485	0.0330	0.0272	0.0132	0.0035	823
t.tfidf.DLH13	0.0718	0.1041	0.2391	0.3947	0.0554	0.0384	0.0309	0.0139	0.0038	876
t.tfidf.DLH13.normQ	0.0620	0.0927	0.2349	0.3913	0.0502	0.0373	0.0293	0.0139	0.0041	952
t.tfidf.Hiemstra	0.0760	0.1189	0.2905	0.4638	0.0609	0.0386	0.0306	0.0165	0.0046	1068
t.tfidf.Hiemstra.normQ	0.0614	0.1041	0.2876	0.4663	0.0549	0.0380	0.0309	0.0167	0.0050	1172
t.tfidf.TFIDF	0.0501	0.0765	0.2133	0.3793	0.0416	0.0292	0.0245	0.0121	0.0034	799
t.tfidf.TFIDF.normQ	0.0589	0.0974	0.2260	0.3857	0.0511	0.0341	0.0268	0.0126	0.0033	775

Table B.7.: Terrier bm25 split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
DLH13.med.83	0.0707	0.1046	0.2408	0.3451	0.0567	0.0386	0.0306	0.0140	0.0051	1186
DLH13.med.83.normQ	0.0598	0.0911	0.2316	0.3451	0.0515	0.0373	0.0289	0.0139	0.0051	1186
DLH.max.65	0.0500	0.0786	0.2041	0.3049	0.0453	0.0315	0.0251	0.0144	0.0066	1522
DLH.max.65.normQ	0.0472	0.0806	0.2015	0.3049	0.0453	0.0308	0.0254	0.0132	0.0066	1522
DLH.med.114	0.0501	0.0786	0.2041	0.3189	0.0453	0.0315	0.0251	0.0144	0.0064	1494
DLH.med.114.normQ	0.0473	0.0806	0.1998	0.3189	0.0453	0.0308	0.0254	0.0130	0.0064	1494
Hiemstra.med.79	0.0741	0.1177	0.2926	0.3998	0.0601	0.0384	0.0305	0.0165	0.0065	1513
Hiemstra.med.79.normQ	0.0617	0.1056	0.2835	0.3998	0.0549	0.0371	0.0305	0.0164	0.0065	1513
BB2.med.98	0.0306	0.0503	0.1628	0.3225	0.0297	0.0224	0.0198	0.0119	0.0045	1043
BB2.med.98.normQ	0.0372	0.0691	0.2098	0.3225	0.0384	0.0287	0.0249	0.0129	0.0045	1043
BM25.med.76	0.0348	0.0574	0.1424	0.3306	0.0318	0.0234	0.0186	0.0100	0.0040	939
BM25.med.76.normQ	0.0562	0.0912	0.2261	0.3306	0.0485	0.0330	0.0272	0.0132	0.0040	939
DFR_BM25.max.101	0.0336	0.0563	0.1402	0.3416	0.0318	0.0230	0.0183	0.0099	0.0040	941
DFR_BM25.max.101.normQ	0.0558	0.0915	0.2231	0.3416	0.0481	0.0324	0.0270	0.0131	0.0040	941
IFB2.max.58	0.0325	0.0521	0.1698	0.3148	0.0332	0.0246	0.0200	0.0127	0.0050	1151
IFB2.max.58.normQ	0.0405	0.0758	0.2141	0.3148	0.0418	0.0310	0.0243	0.0136	0.0050	1151
IFB2.med.120	0.0325	0.0521	0.1698	0.3299	0.0332	0.0246	0.0200	0.0127	0.0048	1108
IFB2.med.120.normQ	0.0405	0.0758	0.2126	0.3299	0.0418	0.0310	0.0243	0.0134	0.0048	1108
lnL2.med.73	0.0397	0.0650	0.1814	0.3266	0.0378	0.0258	0.0212	0.0109	0.0039	899
lnL2.med.73.normQ	0.0564	0.0981	0.2276	0.3266	0.0506	0.0337	0.0269	0.0127	0.0039	899
ln_expB2.max.113	0.0359	0.0641	0.1769	0.3684	0.0378	0.0238	0.0195	0.0127	0.0047	1103
ln_expB2.max.113.normQ	0.0547	0.0922	0.2443	0.3684	0.0511	0.0348	0.0276	0.0145	0.0047	1103
ln_expC2.med.75	0.0374	0.0637	0.1796	0.3574	0.0386	0.0253	0.0203	0.0129	0.0048	1121
ln_expC2.med.75.normQ	0.0555	0.0935	0.2460	0.3574	0.0532	0.0352	0.0285	0.0145	0.0048	1121
LemurTFIDF.med.77	0.0390	0.0664	0.1928	0.3539	0.0399	0.0262	0.0222	0.0128	0.0044	1033
LemurTFIDF.med.77.normQ	0.0537	0.0950	0.2522	0.3539	0.0541	0.0358	0.0279	0.0148	0.0044	1033
PL2.med.133	0.0455	0.0731	0.1941	0.3172	0.0409	0.0280	0.0233	0.0123	0.0047	1089
PL2.med.133.normQ	0.0469	0.0732	0.1980	0.3172	0.0414	0.0280	0.0231	0.0120	0.0047	1089
TFIDF.med.71	0.0460	0.0766	0.2099	0.3233	0.0421	0.0288	0.0245	0.0118	0.0038	881
TFIDF.med.71.normQ	0.0573	0.0976	0.2208	0.3233	0.0511	0.0341	0.0260	0.0124	0.0038	881

Table B.8.: Terrier tfidf split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
DLH13.med.78	0.0715	0.1041	0.2391	0.3451	0.0554	0.0384	0.0309	0.0139	0.0052	1206
DLH13.med.78.normQ	0.0618	0.0927	0.2354	0.3451	0.0502	0.0373	0.0293	0.0139	0.0052	1206
DLH.max.47	0.0518	0.0818	0.2056	0.3008	0.0483	0.0332	0.0267	0.0153	0.0068	1572
DLH.max.47.normQ	0.0483	0.0795	0.2091	0.3008	0.0483	0.0325	0.0269	0.0144	0.0068	1572
DLH.med.111	0.0521	0.0818	0.2059	0.3216	0.0483	0.0332	0.0267	0.0153	0.0066	1530
DLH.med.111.normQ	0.0484	0.0795	0.2056	0.3216	0.0483	0.0325	0.0269	0.0137	0.0066	1530
Hiemstra.med.79	0.0755	0.1189	0.2912	0.4028	0.0609	0.0386	0.0306	0.0166	0.0066	1548
Hiemstra.med.79.normQ	0.0610	0.1041	0.2871	0.4028	0.0549	0.0380	0.0309	0.0167	0.0066	1548
BB2.med.107	0.0367	0.0601	0.1613	0.3281	0.0341	0.0241	0.0203	0.0119	0.0046	1065
BB2.med.107.normQ	0.0402	0.0721	0.2097	0.3281	0.0397	0.0295	0.0247	0.0127	0.0046	1065
BM25.med.73	0.0375	0.0627	0.1514	0.3329	0.0339	0.0247	0.0193	0.0103	0.0041	958
BM25.med.73.normQ	0.0587	0.0929	0.2287	0.3329	0.0489	0.0328	0.0269	0.0133	0.0041	958
DFR_BM25.med.71	0.0364	0.0613	0.1494	0.3302	0.0335	0.0242	0.0187	0.0102	0.0041	954
DFR_BM25.med.71.normQ	0.0578	0.0901	0.2253	0.3302	0.0468	0.0330	0.0268	0.0132	0.0041	954
IFB2.med.118	0.0377	0.0634	0.1718	0.3378	0.0375	0.0254	0.0210	0.0129	0.0049	1143
IFB2.med.118.normQ	0.0429	0.0757	0.2101	0.3378	0.0422	0.0313	0.0240	0.0132	0.0049	1143
lnL2.max.128	0.0446	0.0677	0.1867	0.3485	0.0378	0.0255	0.0215	0.0112	0.0039	910
lnL2.max.128.normQ	0.0578	0.1005	0.2286	0.3485	0.0511	0.0333	0.0272	0.0126	0.0039	910
ln_expB2.med.77	0.0400	0.0689	0.1797	0.3566	0.0395	0.0262	0.0202	0.0127	0.0047	1105
ln_expB2.med.77.normQ	0.0554	0.0895	0.2480	0.3566	0.0506	0.0350	0.0279	0.0148	0.0047	1105
ln_expC2.med.80	0.0406	0.0701	0.1811	0.3645	0.0403	0.0260	0.0210	0.0127	0.0048	1125
ln_expC2.med.80.normQ	0.0547	0.0890	0.2508	0.3645	0.0502	0.0352	0.0286	0.0146	0.0048	1125
LemurTFIDF.med.81	0.0417	0.0702	0.1990	0.3600	0.0416	0.0273	0.0222	0.0132	0.0045	1045
LemurTFIDF.med81.normQ	0.0533	0.0939	0.2535	0.3600	0.0532	0.0354	0.0282	0.0148	0.0045	1045
PL2.med.133	0.0498	0.0733	0.2015	0.3236	0.0427	0.0302	0.0249	0.0130	0.0049	1126
PL2.med.133.normQ	0.0472	0.0775	0.2086	0.3236	0.0440	0.0315	0.0246	0.0127	0.0049	1126
TFIDF.med.78	0.0499	0.0765	0.2133	0.3259	0.0416	0.0292	0.0245	0.0121	0.0038	891
TFIDF.med.78.normQ	0.0585	0.0974	0.2260	0.3259	0.0511	0.0341	0.0268	0.0126	0.0038	891

Table B.9.: Lemur bm25 split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
BM25.med.526	0.0329	0.0490	0.1100	0.2504	0.0279	0.0191	0.0160	0.0068	0.0019	446
BM25.med.526.normQ	0.0180	0.0202	0.0580	0.2504	0.0116	0.0062	0.0043	0.0032	0.0019	446
indri.max.5	0.0647	0.0958	0.1717	0.2468	0.0519	0.0354	0.0266	0.0154	0.0175	4072
indri.max.5.normQ	0.0552	0.1006	0.1717	0.2468	0.0545	0.0356	0.0269	0.0154	0.0175	4072
indri.med.86	0.0690	0.0986	0.2252	0.3396	0.0511	0.0352	0.0279	0.0145	0.0160	3727
indri.med.86.normQ	0.0579	0.0971	0.2218	0.3396	0.0528	0.0367	0.0279	0.0141	0.0160	3727
TFIDF.max.116	0.0481	0.0780	0.2112	0.3642	0.0459	0.0303	0.0240	0.0130	0.0041	951
TFIDF.max.116.normQ	0.0600	0.0985	0.2382	0.3642	0.0554	0.0376	0.0285	0.0136	0.0041	951

Table B.10.: Lemur tfidf split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
BM25.max.609	0.0356	0.0548	0.1228	0.3031	0.0313	0.0215	0.0176	0.0075	0.0021	487
BM25.max.609.normQ	0.0189	0.0202	0.0875	0.3031	0.0124	0.0069	0.0054	0.0049	0.0021	487
indri.max.3	0.0643	0.0957	0.1679	0.2497	0.0528	0.0343	0.0259	0.0158	0.0179	4176
indri.max.3.normQ	0.0538	0.0971	0.1679	0.2497	0.0532	0.0343	0.0259	0.0158	0.0179	4176
indri.med.95	0.0689	0.0992	0.2216	0.3505	0.0528	0.0350	0.0282	0.0145	0.0162	3785
indri.med.95.normQ	0.0579	0.1005	0.2275	0.3505	0.0532	0.0373	0.0280	0.0144	0.0162	3785
TFIDF.med.98	0.0511	0.0852	0.2139	0.3609	0.0481	0.0311	0.0256	0.0132	0.0041	961
TFIDF.med.98.normQ	0.0601	0.1038	0.2401	0.3609	0.0554	0.0382	0.0289	0.0138	0.0041	961

Table B.11.: Solr bm25 split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
standard.max.99	0.0559	0.0917	0.2046	0.3515	0.0502	0.0337	0.0259	0.0126	0.0040	926
standard.max.99.normQ	0.0554	0.0889	0.2283	0.3515	0.0476	0.0345	0.0279	0.0133	0.0040	926
mlt.med.117	0.0502	0.0757	0.1781	0.2927	0.0378	0.0258	0.0205	0.0103	0.0040	936
mlt.med.117.normQ	0.0466	0.0774	0.1822	0.2927	0.0386	0.0266	0.0219	0.0103	0.0040	936

Table B.12.: Solr tfidf split results

run	MAP	r@10	r@100	r@1000	p@10	p@20	p@30	p@100	p@1000	RR
standard.med.76	0.0573	0.0958	0.2136	0.3417	0.0511	0.0326	0.0262	0.0128	0.0040	929
standard.med.76.normQ	0.0564	0.0950	0.2287	0.3417	0.0498	0.0345	0.0273	0.0132	0.0040	929
mlt.max.72	0.0545	0.0834	0.1909	0.2936	0.0412	0.0290	0.0220	0.0110	0.0041	955
mlt.max.72.normQ	0.0482	0.0829	0.2018	0.2936	0.0399	0.0288	0.0237	0.0111	0.0041	955

Bibliography

- [1] AMATI, G. *Probability Models for Information Retrieval based on Divergence from Randomness*. PhD thesis, Department of Computing Science University of Glasgow, 2003.
- [2] ATKINSON, K. H. Toward a more rational patent search paradigm. In *Proceeding of the 1st ACM Workshop on Patent Information Retrieval* (New York, NY, USA, 2008), PaIR '08, ACM, pp. 37–40.
- [3] AZZOPARDI, L., AND BACHE, R. On the relationship between effectiveness and accessibility. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2010), SIGIR '10, ACM, pp. 889–890.
- [4] AZZOPARDI, L., AND VINAY, V. Accessibility in information retrieval. In *Proceedings of the IR Research, 30th European Conference on Advances in Information Retrieval*, ECIR'08. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 482–489.
- [5] AZZOPARDI, L., AND VINAY, V. Retrievability: an evaluation measure for higher order information access tasks. In *Proceeding of the 17th ACM Conference on Information and Knowledge Management* (New York, NY, USA, 2008), CIKM '08, ACM, pp. 561–570.
- [6] BALOG, K., AND DE RIJKE, M. The University of Amsterdam at WebCLEF 2006. In *Working Notes CLEF 2006* (September 2006), A. Nardi, C. Peters, and J. Vicedo.
- [7] BARRETT, M. *Emanuel Law Outlines: Intellectual Property*. Aspen Publishers Inc., 2008.
- [8] BASHIR, S., AND RAUBER, A. Analyzing document retrievability in patent retrieval settings. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications* (Berlin, Heidelberg, 2009), DEXA '09, Springer-Verlag, pp. 753–760.
- [9] BASHIR, S., AND RAUBER, A. Identification of low/high retrievable patents using content-based features. In *Proceeding of the 2nd International Workshop on Patent Information Retrieval* (New York, NY, USA, 2009), PaIR '09, ACM, pp. 9–16.
- [10] BASHIR, S., AND RAUBER, A. Improving retrievability of patents with cluster-based pseudo-relevance feedback documents selection. In *Proceedings of the 18th*

ACM Conference on Information and Knowledge Management (New York, NY, USA, 2009), CIKM '09, ACM, pp. 1863–1866.

- [11] BASHIR, S., AND RAUBER, A. Improving retrievability and recall by automatic corpus partitioning. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems II*, A. Hameurlain, J. Küng, R. Wagner, T. Bach Pedersen, and A. Tjoa, Eds., vol. 6380 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 2010, pp. 122–140.
- [12] BASHIR, S., AND RAUBER, A. Improving retrievability of patents in prior-art search. In *Advances in Information Retrieval*, C. Gurrin, Y. He, G. Kazai, U. Kruschwitz, S. Little, T. Roelleke, S. Rüger, and K. van Rijsbergen, Eds., vol. 5993 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 2010, pp. 457–470.
- [13] BROGLIO, J., CALLAN, J. P., AND CROFT, W. B. Inquiry system overview. In *Proceedings of a Workshop held at Fredericksburg, Virginia: September 19-23, 1993* (Morristown, NJ, USA, 1993), Association for Computational Linguistics, pp. 47–67.
- [14] COMINVENT AS. Solr architecture diagram. <http://www.cominvent.com/2011/04/04/solr-architecture-diagram/>. [Online; accessed 23-April-2011].
- [15] EUROPEAN PATENT OFFICE. EPO - Basic definitions. <http://www.epo.org/patents/patent-information/european-patent-documents/basic-definitions.html>. [Online; accessed 24-January-2011].
- [16] EUROPEAN PATENT OFFICE. EPO - History. <http://www.epo.org/about-us/office/history.html>. [Online; accessed 24-January-2011].
- [17] EUROPEAN PATENT OFFICE. Europäische Klassifikation (ECLA). http://ep.espacenet.com/help?topic=ecla&method=handleHelpTopic&locale=de_ep. [Online; accessed 24-January-2011].
- [18] EUROPEAN PATENT OFFICE. Patent families. <http://www.epo.org/patents/patent-information/about/families.html>. [Online; accessed 26-January-2011].
- [19] EUROPEAN PATENT OFFICE. European Patent Convention. [http://documents.epo.org/projects/babylon/eponet.nsf/0/7bacb229e032863dc12577ec004ada98/\\$FILE/EPC_14th_edition.pdf](http://documents.epo.org/projects/babylon/eponet.nsf/0/7bacb229e032863dc12577ec004ada98/$FILE/EPC_14th_edition.pdf), August 2010. [Online; accessed 24-January-2011].
- [20] FOGLIA, P. Patentability search strategies and the reformed IPC: A patent office perspective. *World Patent Information* 29, 1 (2007), 33 – 53.
- [21] GRAF, E., AND AZZOPARDI, L. A methodology for building a patent test collection for prior art search. In *Proceedings of the Second International Workshop on Evaluating Information Access (EIVA)* (2008), pp. 60–71.

- [22] HARRIS, C. G., ARENS, R., AND SRINIVASAN, P. Comparison of IPC and USPC classification systems in patent prior art searches. In *Proceedings of the 3rd International Workshop on Patent Information Retrieval* (New York, NY, USA, 2010), PaIR '10, ACM, pp. 27–32.
- [23] HARRIS, C. G., ARENS, R., AND SRINIVASAN, P. Using classification code hierarchies for patent prior art searches. In *Current Challenges in Patent Information Retrieval*, M. Lupu, K. Mayer, J. Tait, A. J. Trippe, and W. B. Croft, Eds., vol. 29 of *The Kluwer International Series on Information Retrieval*. Springer-Verlag, Berlin, Heidelberg, 2011, pp. 287–304.
- [24] HUNT, D., NGUYEN, L., AND RODGERS, M. *Patent searching: Tools & Techniques*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2007.
- [25] INFORMATION RETRIEVAL FACILITY. MAREC - Overview. <http://www.ir-facility.org/prototypes/marec>. [Online; accessed 20-May-2011].
- [26] INFORMATION RETRIEVAL FACILITY. MAREC - Statistics. <http://www.ir-facility.org/prototypes/marec/statistics>. [Online; accessed 20-May-2011].
- [27] MAHDABI, P., KEIKHA, M., GERANI, S., LANDONI, M., AND CRESTANI, F. Building queries for prior-art search. In *Multidisciplinary Information Retrieval*, A. Hanbury, A. Rauber, and A. de Vries, Eds., vol. 6653 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 2011, pp. 3–15.
- [28] MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. *An Introduction to Information Retrieval*. Cambridge University Press, April 2009. <http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>.
- [29] MARTNEZ, C. Patent families: When do different definitions really matter? *Scientometrics* 86 (2011), 39–63.
- [30] MAYER, R., AND AVERY, C. *Das US-Patent: Erwirkung und Durchsetzung unter besonderer Berücksichtigung der Rechtsprechung*. Carl Heymanns Verlag KG, 2003.
- [31] MCCANDLESS, M., HATCHER, E., AND GOSPODNETIC, O. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.
- [32] MCLAUGHLIN, B., AND EDELSON, J. *Java & XML*, third ed. O'Reilly Media, Inc., 2006.
- [33] METZLER, D., LAVRENKO, V., AND CROFT, W. B. Formal multiple-bernoulli models for language modeling. In *Proceedings of the 27th annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2004), SIGIR '04, ACM, pp. 540–541.

- [34] MONTAGUE, M., AND ASLAM, J. A. Relevance score normalization for metasearch. In *Proceedings of the tenth International Conference on Information and Knowledge Management* (New York, NY, USA, 2001), CIKM '01, ACM, pp. 427–433.
- [35] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Relevance Judgements File List. http://trec.nist.gov/data/qrels_eng/index.html. [Online; accessed 22-May-2011].
- [36] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Text Retrieval Conference (TREC) Overview. <http://trec.nist.gov/overview.html>. [Online; accessed 11-February-2011].
- [37] OUNIS, I., AMATI, G., PLACHOURAS, V., HE, B., MACDONALD, C., AND LIOMA, C. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)* (2006).
- [38] RICLAS. Vector space model. http://en.wikipedia.org/wiki/File:Vector_space_model.jpg, 2011. [Online; accessed 7-February-2011].
- [39] STROHMAN, T., METZLER, D., TURTLE, H., AND CROFT, W. B. Indri: a language-model based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis* (2005).
- [40] TAIT, J., LUPU, M., BERGER, H., RODA, G., DITTENBACH, M., PESENHOFER, A., GRAF, E., AND VAN RIJSBERGEN, K. Patent Search: An important new test bed for IR. In *Proceedings of the 9th Dutch-Belgian Information Retrieval Workshop (DIR 2009)* (Enschede, The Netherlands, February 2–3 2009), pp. 56–63.
- [41] THE APACHE SOFTWARE FOUNDATION. SchemaXml - Solr Wiki. <http://wiki.apache.org/solr/SchemaXml>. [Online; accessed 23-April-2011].
- [42] THE APACHE SOFTWARE FOUNDATION. SolrConfigXml - Solr Wiki. <http://wiki.apache.org/solr/SolrConfigXml>. [Online; accessed 23-April-2011].
- [43] THE APACHE SOFTWARE FOUNDATION. What is Solr? <http://lucene.apache.org/solr/index.html>. [Online; accessed 18-April-2011].
- [44] THE LEMUR PROJECT. The lemur project homepage. <http://www.lemurproject.org>. [Online; accessed 22-November-2010].
- [45] THE TERRIER PROJECT. Configuring retrieval in Terrier. http://terrier.org/docs/v3.0/configure_retrieval.html. [Online; accessed 26-May-2011].
- [46] THE TERRIER PROJECT. The terrier project homepage. <http://terrier.org/>, 2010. [Online; accessed 24-November-2010].

- [47] UNITED STATES SENATE - COMMITTEE ON THE JUDICIARY. A BILL. To amend title 35, United States Code, to provide for patent reform. <http://judiciary.senate.gov/legislation/upload/BillText-PatentReformAct.pdf>. [Online; accessed 4-February-2011].
- [48] WIKIPEDIA. Lorenz curve — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Lorenz_curve&oldid=399320448, 2010. [Online; accessed 18-December-2010].
- [49] WIKIPEDIA. Lorenz-Kurve — Wikipedia, Die freie Enzyklopädie. <http://de.wikipedia.org/w/index.php?title=Lorenz-Kurve&oldid=81411412>, 2010. [Online; accessed 17-December-2010].
- [50] WORLD INTELLECTUAL PROPERTY ORGANIZATION. Contracting Parties: Paris Convention. http://www.wipo.int/treaties/en/ShowResults.jsp?lang=en&treaty_id=2. [Online; accessed 17-January-2011].
- [51] WORLD INTELLECTUAL PROPERTY ORGANIZATION. Paris Convention for the protection of industrial property. http://www.wipo.int/treaties/en/ip/paris/trtdocs_wo020.html. [Online; accessed 07-April-2011].
- [52] WORLD INTELLECTUAL PROPERTY ORGANIZATION. Patent Cooperation Treaty PCT (1970). <http://www.wipo.int/pct/en/treaty/about.html>. [Online; accessed 17-January-2011].
- [53] WORLD INTELLECTUAL PROPERTY ORGANIZATION. WIPO Intellectual Property Handbook: Policy, Law and Use - Chapter 1. <http://www.wipo.int/export/sites/www/about-ip/en/iprm/pdf/ch1.pdf>. [Online; accessed 20-January-2011].
- [54] WORLD INTELLECTUAL PROPERTY ORGANIZATION. WIPO Intellectual Property Handbook: Policy, Law and Use - Chapter 2. <http://www.wipo.int/export/sites/www/about-ip/en/iprm/pdf/ch2.pdf>. [Online; accessed 20-January-2011].
- [55] WORLD INTELLECTUAL PROPERTY ORGANIZATION. World Intellectual Property Indicators 2010. <http://www.wipo.int/export/sites/www/shared/images/icon/new/pdf.gif>. [Online; accessed 31-August-2011].
- [56] WORLD INTELLECTUAL PROPERTY ORGANIZATION. List of PCT contracting states and map (July 2010). http://www.wipo.int/pct/en/list_states.pdf, 2010. [Online; accessed 17-January-2011].
- [57] WORLD INTELLECTUAL PROPERTY ORGANIZATION. Guide to the IPC (2011). http://www.wipo.int/export/sites/www/classifications/ipc/en/guide/guide_ipc.pdf, 2011. [Online; accessed 24-January-2011].
- [58] WORLD TRADE ORGANIZATION. TRIPS: Agreement on trade-related aspects of intellectual property rights - Preamble. http://www.wto.org/english/tratop_e/trips_e/t_agm1_e.htm. [Online; accessed 07-April-2011].

- [59] ZENZ, V., WURZER, S., DITTENBACH, M., AND AMBROSI, E. On the effects of indexing and retrieval models in patent search and the potential of result set merging. In *1st International Workshop on Advances in Patent Information Retrieval* (2010), AsPIRe'10.