NTNU

Norwegian University of
Science and Technology

Improving Performance of Biomedical Information Retrieval using Document-Level Field Boosting and BM25F Weighting

Jørgen Jervidalo

Master of Science in Informatics

Submission date: July 2010

Supervisor: Herindrasana Ramampiaro, IDI

Co-supervisor: Trond Aalberg, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Improving Performance of Biomedical Information Retrieval using Document-Level Field Boosting and BM25F Weighting

Jørgen Jervidalo

July 13, 2010

Abstract

Corpora of biomedical information typically contains large amounts of ambiguous data, as proteins and genes can be referred to by a number of different terms, making information retrieval difficult. This thesis investigates a number of methods attempting to increase precision and recall of searches within the biomedical domain, including using the BM25F model for scoring documents and using Named Entity Recognition (NER) to identify biomedical entities in the text. We have implemented a prototype for testing the approaches, and have found that by using a combination of several methods, including using three different NER models at once, a significant increase (up to 11.5%) in mean average precision (MAP) is observed over our baseline result.

Acknowledgements

I want to express my gratitude to Guro and my daughter Anna for their support while writing this thesis. Your patience have been invaluable to me.

I would also like to thank my supervisor, Heri Ramampiaro, and my fellow students at Sule, for great advice and feedback.

It has been five great years at NTNU.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The first automated information retrieval systems were introduced during the 1960s, and the field of Information Retrieval (IR) was born. As the amount of electronic information increased, not least because of the widespread adoption of World Wide Web during the 1990's, Information Retrieval has become a domain of great interest, and there has been much research and development in the field (Singhal, 2001). Even though most of this research also applies to IR systems for biomedical information, Ramampiaro (2010) describes several problems that are specific to this domain. One of the biggest challenges is that biomedical information typically contains large amounts of domain-specific terminology with high ambiguity (Krauthammer and Nenadic, 2004). The same protein or gene can use widely different terms, and the same term can have several meanings.

1.2 Motivation

The inconsistency and ambiguity described in the previous section can lead to both low recall and low precision in search results within the biomedical domain. Ramampiaro (2010) successfully addresses some of the challenges described in his BioTracer prototype, a search engine tailored for biomedical information. Johannsson (2009) tries to increase precision of a search by giving each term in a document a weight based on the context in which it is found, but concludes that his implementation has little or no effect on the results. The author suggests methods to combine term boosting with query expansion to increase both precision and recall of a search, as well as ways to improve the context weighting.

In this thesis, we will try to extend the research done by Johannsson (2009), and strive to achieve higher precision in search results from biomedical articles.

1.3 Outline of the solution

We have developed a prototype in Java that tests several retrieval models and weighting algorithms. It uses Named Entity Recognition to automatically identify biomedical terms in an attempt to increase the precision of searches within biomedical corpora. The prototype is evaluated using the document collection and topics from TREC 2004 Biomedical track. The idea behind the prototype is explained in Chapter 4, and details about the implementation are found in Chapter 5.

1.4 Thesis structure

The thesis is structured into the following chapters:

Introduction This chapter contains a brief background, the motivation and the problem statement.

Background information This chapter explains the basic concepts and definitions used in the thesis. This includes theory about information retrieval and the methods used to evaluate results.

Related work This chapter presents other work and research related to this thesis.

Our approach In this chapter, we explain the idea behind the prototype.

Implementation This chapter explains how the prototype is implemented.

Evaluation In this chapter we evaluate the prototype and present the results.

Conclusion and future work Contains interpretations of the results and the conclusions, as well as future work.

Chapter 2

Background information

In this chapter we will give a short introduction to information retrieval (IR), introduce some retrieval models, and describe some commons methods used to evaluate IR systems.

2.1 Information Retrieval

The term *information retrieval* can be thought of as a more academic synonym to the term *search*, and can be defined as (Mannning et al., 2008):

> 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).

A traditional IR system provides a way for a user to enter a *query*, typically using keywords, and the system will return the documents it deems relevant to the query from the document collection. This could be done by scanning the collection for matched terms each time a search is performed, but this is obviously a very time consuming process, even in relatively small document collections. That is why most IR systems use an *index*, often referred to as an *inverted index*. An inverted index is basically a list that maps a specific term or *metadata*, such as year or author, to one or more specific documents. The process of creating an index is time-consuming, but the index only needs to created once (and then updated when the document collection changes or at certain intervals).

During the index construction certain linguistic preprocessing tasks are executed and *stop words*[1] are usually removed. The result is a list of *normalised*[2] and often *stemmed* or *lemmatised*[3] terms with pointers to the documents in which they occur. More advanced indexes also stores the position(s) of the terms within each document. The same preprocessing tasks that are performed on the index must be done to all queries as well.

2.2 Information Retrieval models

Every time a user of an IR system creates a query, the system needs to return some results. How can a computer tell which documents are relevant to the query, and more important; which results are more relevant than others?

[1] Extremely common words such as *and*, *to* and *it*. Stop words will appear in virtually all documents, and therefore does not add any value to the index

[2] Creating equivalence classes so that for instance different spellings of the same word will map to the same term

[3] Stemming is the crude process of chopping off the end of a word to find some kind of base form, even if it is not necessarily grammatically correct (ponies $\rightarrow$ poni), while lemmatisation uses a vocabulary to return the true base form (ponies $\rightarrow$ pony)

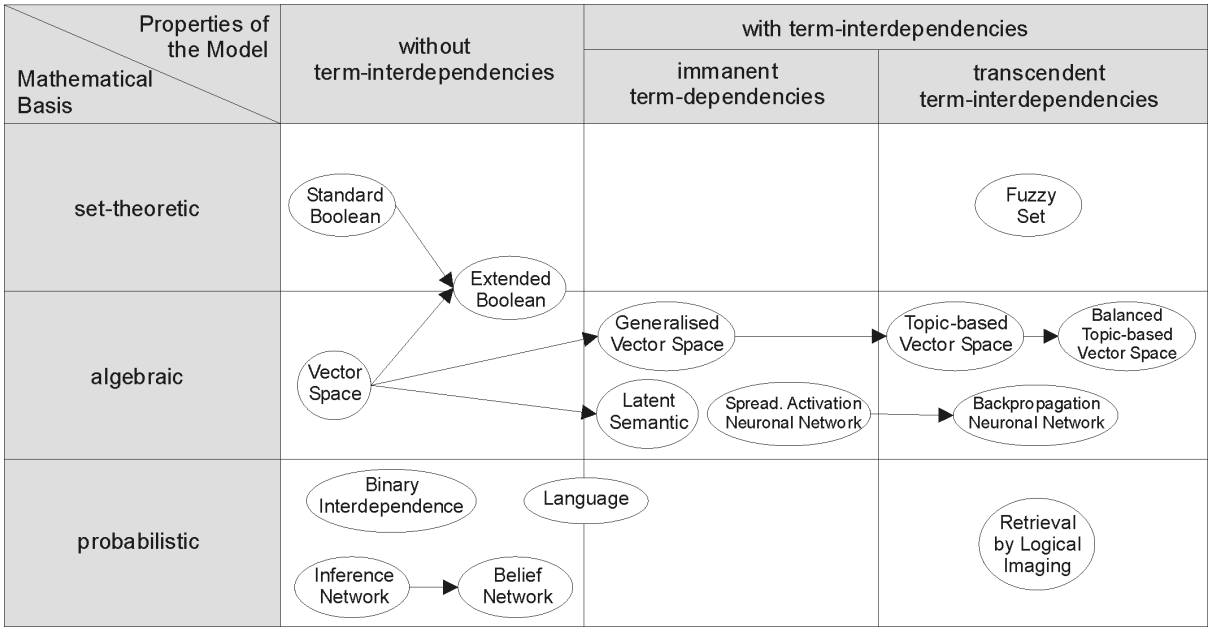

Figure 2.1: Information Retrieval model categories: The table shows the categories of some of the most common IR models. German original from Kuropka (2004), English translation by Thomas Hoffmann. Available from http://commons.wikimedia.org/wiki/File:Information-Retrieval-Models.png under the GNU Free Documentation License.

There is of course no definitive answer, but there are many approaches to solving the problem. We use different *Information Retrieval models* to represent the documents in the document collection, which can usually be divided into one (or more) of three categories; set-theoretic (including boolean) models, algebraic models, and probabilistic models, as seen in Figure 2.1 (Mannning et al., 2008). In this section we will first introduce the concept of term weighting, before we describe the three categories of retrieval models.

2.2.1 Scoring and term weighting

In the introduction, we asked the question *which results are more relevant*? To answer this, we need to assign some kind of a score that computes the similarity between a query and its results. There are several ways to do this, but a common denominator is that they often use the *term frequency-inverse document frequency* (tf-idf) model (described below) to create term weights.

2.2.1.1 tf-idf

tf-idf is a *bag of words*[4] weighting model used to give weights to the terms in a document collection by measuring how often a term is found within a document (*term frequency*), offset by how often the term is found within the entire collection (*inverse document frequency*). This means that a term t will have highest weight when it is found many times within a single document, and lowest when it is found in all documents.

The term frequency (tf), the inverse document frequency (idf) and tf-idf are defined as follows:

$$\mathrm{tf}(t,d) \quad = \quad \frac{n_{t,d}}{\sum_k n_{k,d}} \tag{2.1}$$

[4] Term order and grammar is ignored, what counts is the number of occurrences of each term. The documents *Peter is smarter than Jane* and *Jane is smarter than Peter* is considered equal in a bag of words model.

$$\text{idf}(t) = \log \frac{|D|}{\text{df}_t} \tag{2.2}$$

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \cdot \text{idf}(t) \tag{2.3}$$

where $n_{t,d}$ is the number of times the term t occurs in document d, $\sum_k n_{k,d}$ is the sum of all terms k in document d, $|D|$ is the number of documents in the collection and df_t is the document frequency (the number of documents in the collection in which the term t occurs) (Mannning et al., 2008).

2.2.2 Set-theoretic models

Models in this category uses sets of words or phrases to represent the documents, and uses some sort of set-theoretic operations to determine the relevance. The most common model is the Boolean model (described below), but methods such as Fuzzy retrieval also falls into this category.

2.2.2.1 Boolean model

The Boolean model is a very simple model where, based on a query, a document is either deemed relevant or irrelevant. When formulating a query, the user can include certain keywords; the most common being `AND`, `OR`, `XOR` and `NOT`. If we let D be a collection of documents, and let S_1, S_2 be the set of all the documents in D that contains the terms T_1 and T_2, respectively, we can define the boolean operators above as follows:

$$\texttt{NOT}\, T_1 = D - S_1 \tag{2.4}$$

$$T_1\, \texttt{AND}\, T_2 = S_1 \cap S_2 \tag{2.5}$$

$$T_1\, \texttt{OR}\, T_2 = S_1 \cup S_2 \tag{2.6}$$

$$T_1\, \texttt{XOR}\, T_2 = (S_1 \cup S_2) - (S_1 \cap S_2). \tag{2.7}$$

An example: Consider a set of documents $D = \{D_1, D_2, D_3\}$, where

$$D_1 = \{\text{lorem ipsum dolor sit amet}\} \tag{2.8}$$

$$D_2 = \{\text{consectetur adipisci velit}\} \tag{2.9}$$

$$D_3 = \{\text{duis aute irure dolor in reprehenderit}\}. \tag{2.10}$$

Let queries Q_1, Q_2 be:

$$Q_1 = \texttt{lorem AND ipsum} \tag{2.11}$$

$$Q_2 = \texttt{dolor OR reprehenderit}. \tag{2.12}$$

If R_1, R_2 is the results of Q_1, Q_2 respectively, we have $R_1 = \{D_1\}$ because D_1 is the only document that contains both `lorem` and `ipsum`, and $R_2 = \{D_1, D_3\}$ because both documents contains either `dolor` or `reprehenderit` (or both).

The boolean model has no concept of relevance; a document will either match or not match the user's query. That is; every document that matches the query is returned to the user in some kind of arbitrary order. In a large document collection, the record set might potentially become very cumbersome (if at all plausible) to sift through. As such, it is not very useful on its own, but it has been shown that it can be a very effective extension to another retrieval model (Ramampiaro, 2010; The Apache Lucene project, 2009). Many commercial IR systems also implements the boolean model to a certain degree.

2.2.3 Algebraic models

In this category, both documents and queries are represented as vectors, tuples or matrices, and relevance is computed by measuring the scalar distance between the vectors. In addition to the Vector Space Model (described below), methods based on matrix decompositions such as Latent Semantic Indexing is put in this category (Mannning et al., 2008).

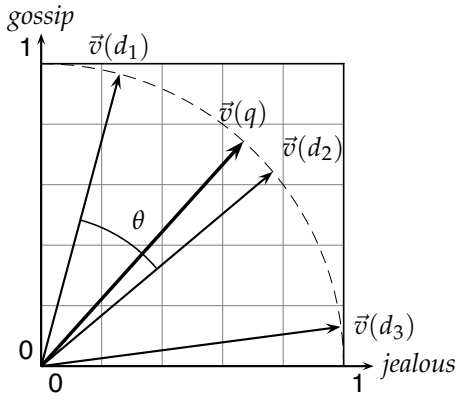

Figure 2.2: Cosine similarity illustrated: $\text{sim}\,(d_1, d_2) = \cos\theta$. Illustration from Mannning et al. (2008).

2.2.3.1 Vector Space Model

The Vector Space Model (VSM) is an algebraic model where both documents and query terms are represented as vectors in a vector space. The model was developed by Salton et al. (1975), and is still widely in use. It is for instance used as the default scoring model in the open source search engine Lucene[5] (The Apache Lucene project, 2009).

In VSM, we derive a vector $\vec{V}(d)$ from each document, where each dimension in the vector corresponds to a separate term from the document collection. Each term in the vector is given a relevance weight using a weighting scheme, often using the tf-idf formula described in Equation 2.3. A key step in the VSM model is to view the query itself as a vector in the same vector space as the documents, giving us $\vec{V}(q)$. We can then use for instance the *cosine similarity* between the query and each of the documents to compute the relevance (see Figure 2.2), given by (Mannning et al., 2008):

$$\text{sim}(d_1, d_2) \quad = \quad \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}. \tag{2.13}$$

Because we view the query as a "document" by itself, we can use this formula not just to compute the relevance between a query and a document, but also the relevance between two documents. This is useful if, for instance, a search engine wants to support a "show similar documents" feature.

2.2.4 Probabilistic models

Models in this category uses probability theory, such as Bayes' theorem, to estimate the likelihood that any given document is relevant to the user's information need. Models include the Binary Independence Model, Okapi BM25 and BM25F, different language models and more.

2.2.4.1 Okapi BM25

Okapi BM25 is a bag of words ranking model developed by Robertson and Jones (1994) used to compute the relevance between a query and the documents in the collection.

Given a document D and a query Q (containing terms $q_1 \ldots q_n$), the BM25 score is calculated as follows (Mannning et al., 2008; Robertson and Jones, 1994):

$$\text{BM25}(D, Q) \quad = \quad \sum_{i=1}^{n} \text{idf}(q_i) \frac{\text{tf}\,(q_i, D) \cdot k_1 + 1}{\text{tf}(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgl}_D}\right)} \tag{2.14}$$

[5] http://lucene.apache.org/java/docs/index.html

where k_1 ($k_1 \geq 0$) and b ($0 \leq b \leq 1$) are tuning constants, $|D|$ is the length of document D, and avgl_D is the average document length in the collection. A high value for k_1 means that the term frequency will have more impact, while the value b affects length normalisation; a value of 0 means that a document is long because it is multitopic, while a value of 1 means that it is long because it is repetitive. In the TREC workshops[6], these values were found to be most effective at $k_1 = 2$ and $b = 0.75$. The idf formula (defined in Equation 2.2) in the BM25 formula is sometimes replaced with alternative versions, for instance to achieve a level of smoothing or account for relevance feedback. Equation 2.15, using a low value for α, for instance $\alpha = \frac{1}{2}$, is an often used variation:

$$\text{idf}(q_i) = \log \frac{|N| - \text{df}(q_i) + \alpha}{\text{df}(q_i) + \alpha}. \tag{2.15}$$

Using BM25 ensures that the effect of the term frequency is not too strong and that for a term occurring just once in an average length document, the weight is the inverse document frequency (idf) of the term (Robertson and Jones, 1994). However, Robertson et al. (2004) have shown that when weighting structured documents (such as documents with title, abstract and content etc.), a linear combination of the scores for the different fields can lead to over-estimating the importance of a term, especially when there are big differences in the field lengths.

2.2.4.2 BM25F

BM25F is a variant of the Okapi BM25 algorithm (described above in Section 2.2.4.1) that is better suited than plain BM25 for ranking structured documents (Zaragoza et al., 2004; Robertson and Zaragoza, 2007). It uses a weighting algorithm to normalize the term frequencies for each of the fields in the document. To calculate this weight for term t in a document D containing the fields $f_1 \ldots f_n$, we use the following formula (Pérez-Iglesias et al., 2009):

$$\text{weight}(t, D) = \sum_{i=1}^{n} \frac{\text{tf}(t, f_i) \cdot \text{boost}_{f_i}}{1 - b_{f_i} + b_{f_i} \cdot \frac{l_{f_i}}{\text{avgl}_{f_i}}} \tag{2.16}$$

where $\text{tf}(t, f_i)$ is the term frequency for term t in field f_i, boost_{f_i} is a boost value given to the field, b_f ($0 \leq b_{f_i} \leq 1$) is a tuning constant related to field length normalisation (corresponding to the b value of the BM25 formula in Equation 2.14), l_{f_i} is the length of the field and avgl_{f_i} is the average length of the field.

Given a document D and a query Q (containing terms $q_1 \ldots q_n$), the BM25F algorithm is then calculated as follows:

$$\text{BM25F}(D, Q) = \sum_{i=1}^{n} \text{idf}(q_i) \cdot \frac{\text{weight}(q_i, D)}{\text{weight}(q_i, D) + k_1} \tag{2.17}$$

where $\text{idf}(q_i)$ is calculated as in Equation 2.15 and k_1 is a tuning constant, corresponding to the one used in the BM25 formula in Equation 2.14.

2.3 Evaluation of IR systems

There are two major approaches to evaluating an IR system; user-based evaluation and system evaluation. The first method, which is qualitative, tries to measure the user's satisfaction with the system, and is probably the optimal way of judging how well the IR system meets the information need of its users. It is

[6] See Section 2.3.1.2.

hard, however, to execute such an evaluation in a large scale because of the large number of representative users needed and the expenses involved. Instead it is usual to use system evaluation (Voorhees, 2002).

In this section, we will describe some of the concepts and methods one can use to evaluate or measure the performance of an information retrieval system.

2.3.1 Test collections

2.3.1.1 The Cranfield experiments

The Cranfield 2 experiment (Cleverdon, 1967) introduced a methodology for evaluating index languages, and the notion of using *test collections* quickly became the de facto way of evaluating IR systems, using concepts such as precision (see Section 2.3.2.2) and recall (see Section 2.3.2.2). A test collection consists of three parts; the documents, queries (statements of the information need), and relevance judgements (the "solution" to the queries, as evaluated by experts on the topic). The test collection used in the experiment is way to small for today's standard, but the methodology the experiment introduced is still used in more modern test collections.

2.3.1.2 Text REtrieval Conference (TREC)

TREC is a series of workshops run by the U.S. National Institute of Standards and Technology (NIST) every year since 1992. Each year there are one or more *tracks*, each with a specific challenge as well as ancillary document collections, definitions of information needs and sets of relevance judgements. In the more recent years, there have been specific tracks for several fields of study, including the genomics track which ran from 2003 to 2007[7]. In the prototype developed as part of this thesis, we have used the document collection from the genomics 2004/2005 track[8], and topics and relevance judgements from the genomics 2004 track. See Chapter 6 for more information.

2.3.1.3 Other test collections

There are several other test collections available, such as the large GOV2 collection used in the TREC Terabyte track[9] and test collections focused on Asian or European languages instead of the more English-centric TREC collections.

2.3.2 Measurements

In this section we present some common measures of system effectiveness.

2.3.2.1 Precision

The proportion of relevant documents among the documents retrieved in a search (see Figure 2.3), defined as

$$P = \frac{|\text{relevant items retrieved}|}{|\text{retrieved items}|}. \tag{2.18}$$

If ten documents were retrieved and six of these were relevant, we have precision $P = \frac{6}{10} = 0.6$.

[7] See http://ir.ohsu.edu/genomics/.

[8] The same document collection was used both years.

[9] See http://ir.dcs.gla.ac.uk/test_collections/.

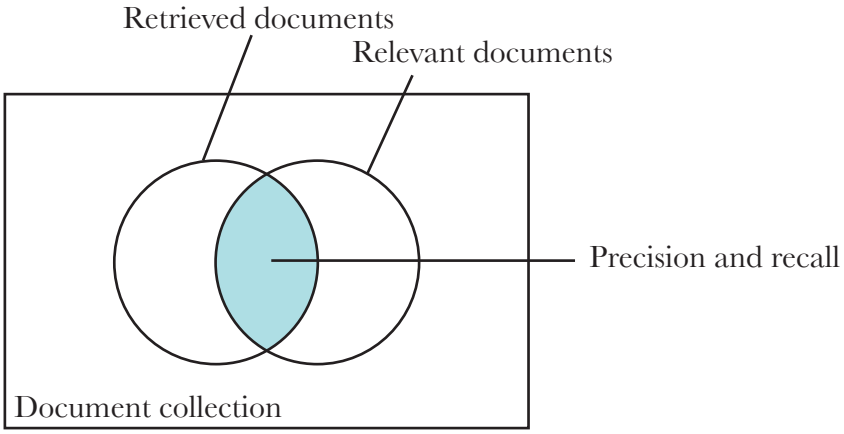

Figure 2.3: Precision and recall illustrated.

2.3.2.2 Recall

The proportion of relevant documents retrieved in a search among all the relevant documents in the collection (see Figure 2.3), defined as

$$R = \frac{|\text{relevant items retrieved}|}{|\text{relevant items}|}. \tag{2.19}$$

If six of the retrieved documents are relevant, and there are a total of 20 relevant documents in the collection (14 of which were not returned), we have recall $R = \frac{6}{20} = 0.3$.

2.3.2.3 Precision at k

Measures the precision as described below, but after k documents are retrieved. It is not very stable because the number of relevant documents in the collection strongly influences the measurement, especially if k is smaller than the number of relevant documents that exists for the topic (Mannning et al., 2008). Precision at k can still be of some interest because it mimics the way many people search on the Web; if a search result returns 10 results on each page, "precision at 10" resembles the fact that very few people look further than the first page of results (Eysenbach and Kohler, 2002; Silverstein et al., 1999).

2.3.2.4 R-Precision

R-Precision is the precision after R documents are retrieved, where R is the total number of relevant documents in the collection for this query. An R-Precision of 1 means that both precision and recall is perfect. If $k = R$, "Precision at k" and R-Precision are equal.

2.3.2.5 Mean Average Precision (MAP)

MAP is a more stable evaluation measure than Precision or R-precision, because it is based on much more information than the other methods. It also considers the order of which the documents are returned, giving a higher score if the relevant documents are near the top. Mannning et al. (2008) defines MAP as

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \tag{2.20}$$

where $\{d_1 \dots d_{m_j}\}$ is the set of relevant documents for an information need $q_j \in Q$ and R_{jk} is the set of ranked results from the top and down to document d_k.

The MAP returns a single score for the search, which might be a disadvantage because of the many factors that affects it. The score can be hard to interpret, as the same score can be achieved in a number of different ways.

2.3.2.6 Incomplete judgement sets and alternative measures

The Cranfield methodology introduced in Section 2.3.1.1 makes a few assumptions, one of which is that the relevance judgements are complete (which means that *all* relevant documents in the document collection is known). Evaluation measures that are derived from recall and precision, are not robust when the relevance judgements are incomplete (Voorhees, 2002).

In practice, for instance on the Internet, getting hold of complete judgement sets is obviously not possible. No one can know all web pages that are relevant to an information need when the document collection contains more than 10^{12} unique URLs (Alpert and Hajaj, 2008). As such, we can deduce that in a standard web search (using for instance Google), precision becomes much more important than recall. If the user performing the search can find some relevant documents at the first page of the search results (alas, high precision), he will probably be satisfied, and he will have no interest in knowing about the rest of the relevant documents that were not retrieved (low recall).

The test collections described in Section 2.3.1 are several orders of magnitude smaller than the World Wide Web, but most of them are still so large that getting complete relevance judgements is highly unlikely. Citing Voorhees (2002):

> Assuming a judgment rate of one document per 30 seconds, and judging round-the-clock with no breaks, it would still take more than nine months to judge one topic for a collection of 800,000 documents (the average size of a TREC collection).

Most test collections therefore use a technique called *pooling* to create a subset ("pool") of the document collection for each topic, which is then used in the judging process. All documents not in this pool are deemed to be irrelevant, even if they in reality are not. In the TREC conferences, documents are added to the pool by each of the participating groups. They submit the results of their runs to the organiser (NIST), and the top documents returned, for instance the first hundred results, are added to the pool to be judged (Voorhees, 2002). Because of the way documents are added, this tends to produce rather good pools, with most of the relevant documents for a topic being judged (Yilmaz and Aslam, 2006).

Finding more robust measures than the ones described in the previous sections, have been an area of much research (Buckley and Voorhees, 2004; Sakai and Kando, 2008; Yilmaz and Aslam, 2006). It has been shown that the Cranfield methodology is robust to small violations of the completeness requirement, and several alternative measures have been introduced. The most "successful"[10] of these measures is the *bpref* measure, introduced by Buckley and Voorhees. It have been shown to be more stable and robust than merely using recall and precision-based methods.

Johannsson (2009) chose to simplify the problem, and removed all unjudged documents from the document collection. As such, the test set was completely judged, and the author was able to use MAP in a robust way. To be able to easily compare our scores, we have chosen to follow Johannsson's lead and do the same. This is elaborated on in Chapter 6.

[10] The measure have been used, in addition to MAP, in many TREC runs since 2005, and is included in the official `trec_eval` tool (http://trec.nist.gov/trec_eval/).

Chapter 3

Related work

3.1 Strategies used to increase precision and recall

There are several strategies and methods introduced that tries to increase precision and/or recall in IR systems. Many of them are based around the issue of *synonymy*: The terms in a document collection are often ambiguous; the same concept can be referred to with different words, and conversely, the same word can relate to totally different concepts. The methods developed to address this problem can be divided into global and local methods. The global methods works on a global scale and tries to refine the query based on for instance a thesaurus, while the local methods looks at the documents returned in a search and tries to extract relevant information from these and use it to refine the query (Mannning et al., 2008).

3.1.1 Relevance Feedback

There are three basic approaches to relevance feedback:

User After a successful search, the system will give the user an opportunity to mark some of the returned documents as relevant, and then run the search again. The second run will use information about the marked, relevant documents to refine and reweight the query terms (for instance using the Rocchio algorithm), returning an even better search result.

Pseudo This method is similar to the regular user relevance feedback described above, but instead of letting the user mark documents as relevant, the system will do this automatically by assuming that the top k documents are relevant.

Indirect This method uses global analysis instead of the local analysis used by the two methods above. Rather than letting the user mark documents as relevant, this method uses data collected from a large number of users, marking popular documents as more relevant. The idea is that if a lot of users uses a document returned from a given term, this document is probably pertinent to that term.

3.1.2 Query Expansion

The two-step method for query expansion is:

- The user inputs a query.
- The system analyses the query and, using a thesaurus, it can either:
  - present some alternative or expanded queries to the user, along with the original results.
  - automatically expand the query and then present the user with the results of the new query.

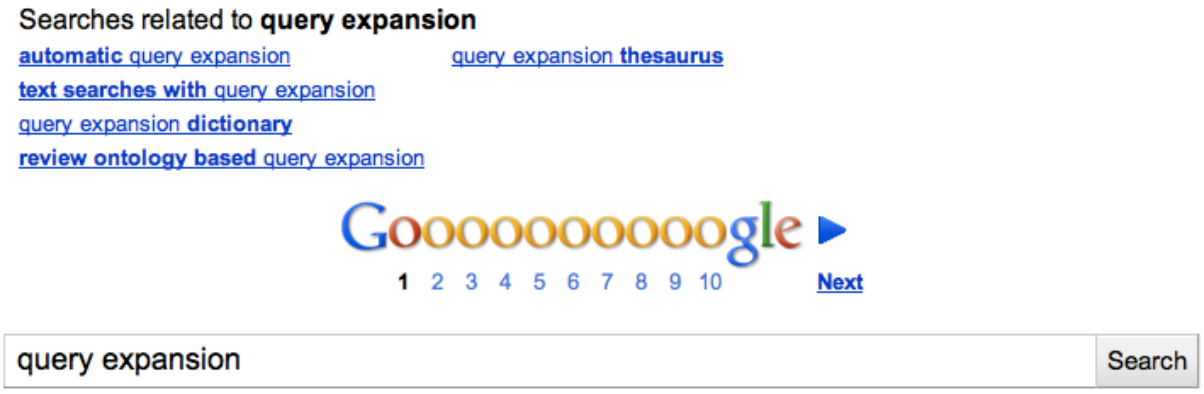

Figure 3.1: An example of query expansion on Google when searching for `query expansion`.

Query expansion is an effective tool when the user is searching for ambiguous terms or terms with a lot of synonyms. It is effective in increasing the recall of a search, but might also decrease the precision. A search for `charge`, for instance, is quite ambiguous, and a thesaurus might suggest synonyms that are unrelated to the user's intended meaning of the word (Mannning et al., 2008).

Query expansion is popular with web search engines, such as Google. An example can be seen in Figure 3.1.

The most common form of query expansion is global analysis using a thesaurus. The most popular methods for generating said thesaurus are (Mannning et al., 2008):

Controlled vocabulary This method "forces" query expansion by assigning one or more canonical terms for each concept. The vocabulary is controlled by human editors.

Manual thesaurus Similar to the controlled vocabulary, but without any canonical terms.

Automatic thesaurus generation This method tries to automatically generate a thesaurus from a collection of documents in a domain.

Query log mining This method focuses on the manual query expansions made by other users, by suggesting the most used expansions made by other users of the system.

3.1.3 MeSH Query Expansion/Relevance Feedback

The U.S. National Library of Medicine produces a controlled vocabulary/thesaurus of subject headings for biomedical and health-related information and documents called MeSH® (Medical Subject Headings)[1]. In 2010, it contains 25,588 descriptors organised hierarchically, and more than 172,000 entry terms that maps to these descriptors. Almost all articles in the MEDLINE database[2] are tagged with multiple[3] MeSH terms.

Using MeSH information in connection with query expansion or relevance feedback have been an area of much research[4]. Camous et al. (2006) uses pseudo-relevance feedback (described in Section 3.1.1) to select the top n MeSH headings ($5 \geq n \geq 40$) from the top 5 results of each topic, and uses these

[1] See http://www.nlm.nih.gov/mesh/.

[2] MEDLINE is a bibliographic database compiled the U.S. National Library of Medicine that contains over 18 million references to journal articles within medicine and biomedicine.

[3] 10-12 MeSH tags per article in average, according to Camous et al. (2006).

[4] Srinivasan (1996a,b); Shin et al. (2004); Bacchin and Melucci (2005); Camous et al. (2006); Kim and Chen (2007); Lu et al. (2009), among others.

MeSH terms to expand the query. They report an increase in MAP of up to 8.8%. Lu et al. (2009) reports a more modest improvement using a more classic form of query expansion, and notes that the increased retrieval performance in practice might not benefit end users. Others, such as Bacchin and Melucci (2005), even reports a decrease in the performance.

3.1.4 Weighting different document parts independently

When dealing with structured documents, such as MEDLINE citations, we can give different weights to each of the document parts (like the title or the abstract). The idea is that, for instance, if a term from a user's query is found in the title, the probability that the rest of the document is relevant to this term is higher than if the same term is found in the abstract. Ramampiaro (2010) found that weighting the title twice as much as the abstract produced good results.

3.1.5 Document-Level Term Boosting

As we briefly mentioned in Section 1.2, this thesis is continuing the work of Johannsson. In his thesis (2009), he described and implemented an IR system that gave a weight to all terms in a document based on the context of which the term is found. The context, in this case, was the number of times biomedical relevant keywords were mentioned within the same sentence as the term.

In practice, the author did the following to achieve document-level term boosting:

- Used sentence detection to extract each sentence of the document, using a library from a linguistic tool kit for Java called LingPipe[5].
- Used a text mining technique called Named Entity Recognition (NER) to identify biomedical information in the sentences. Two different models from the same tool kit (GeneTag and GENIA[6]) were used in this process.
- Weighted and normalised the sentence based on this information.
- The sentence weight was used as a boost in conjunction with standard ranking models; namely the Vector Space Model (described in Section 2.2.3.1) and the Okapi BM25 model (described in Section 2.2.4.1).

Johannsson concludes that the effect of the term boost as implemented in his prototype is minimal, but may be worth further research.

3.2 Implemented retrieval systems

3.2.1 PubMed

PubMed is the official (but not the only) search engine for inquiries to the MEDLINE database. It is developed and maintained by the U.S. National Center for Biotechnology Information (NCBI) at the U.S. National Library of Medicine (NLM). It provides citations and links to full text articles, and will automatically try to map a user's query to MeSH terms to increase the performance of the search. PubMed also offers some advanced features not found in most web search engines, but Herskovic et al. (2007) suggests that they are infrequently used.

[5] The LingPipe tool kit is available from http://alias-i.com/lingpipe/, and the sentence detection is described in detail on http://alias-i.com/lingpipe/docs/api/com/aliasi/sentences/HeuristicSentenceModel.html. LingPipe is discussed in Section 5.1.1.3.

[6] Both models are available from http://alias-i.com/lingpipe/web/models.html.

PubMed launched an updated and simplified interface in October 2009[7] that encourages the use of simple, Google-like queries instead of more complex boolean queries, which PubMed then tries to intelligently parse. Users can still access an advanced search page to use all the features manually.

3.2.2 BioTracer

BioTracer is the work of Ramampiaro (2010), and is a search tool prototype that implements many different strategies to improve the performance of searches within the BioMedical domain. The author uses methods such as different weighting for different document parts, user relevance feedback (see Section 3.1.1), extending tf-idf (see Section 2.2.1.1) and Okapi BM25 (see Section 2.2.4.1) to support more kinds of queries, such as boolean queries and wildcard (*) support. BioTracer has shown promising results against the TREC corpus, but has not yet been tested in a realistic environment.

3.2.3 GoPubMed

GoPubMed is a knowledge-based biomedical search engine that originally added the possibility to browse PubMed results using Gene Ontology (GO)[8], a structured vocabulary of genes, gene annotations and gene attributes (Doms and Schroeder, 2005). Later, MeSH support was added, and GO and MeSH are now both used.

3.2.4 Textpresso

Textpresso[9] is a text-mining engine that searches within specific corpora (currently 17 different literatures are supported). It supports full text search, so that the entire articles are searchable, and support relations and descriptions of biomedical concepts, using its own ontology. By using full text search, the authors claim to increase recall from 45% to 95% (Müller et al., 2004).

[7] See http://www.nlm.nih.gov/pubs/techbull/so09/so09_pm_redesign.html
[8] See http://www.geneontology.org/
[9] See http://www.textpresso.org/

Chapter 4

Our approach

This chapter elaborates on the ideas on which our prototype is based.

4.1 The idea

The goal of this thesis is to continue the work of Johannsson, and try to further improve on the results presented in his thesis (2009). To achieve this, we will try to combine several techniques and methods, including some changes to the original idea as well as some new ones.

The basic idea is to use data mining techniques to identify biomedical terms in all documents in a collection. The hypothesis is that a document with a high density of said terms is, on average, more relevant than a document with lower density. While Johannsson used sentence extraction to do the data mining on a per-sentence basis in the document's abstract, we have chosen to weight the abstract as a whole, and additionally use both the title field and the mesh fields found in MEDLINE citations in the weighting process. We propose that because abstracts generally are short and compact, each sentence will often contain a lot of biomedical information in a corpora such as MEDLINE. The sentence weighting done by Johannsson boosted the sentence weight if a query term matched a term in the sentence, and the sentence also contained biomedical relevant terms. In Figure 4.1 we have plotted all the boost values for the 8382 matched sentences used in his solution. We can see that the distribution is somewhat poor (most values varies only between 1.6 and 1.9), partially invalidating the effects of the boosting values, as a very high proportion of the sentences are boosted with almost the same value.

The basic idea we propose is actually a simplification of Johannsson's implementation, but it still brings about some implementational quirks. In addition, we test another approach that identifies biomedical terms in the query and maps these to MeSH terms (see Section 3.1.3 for more information on MeSH), and then boosts the weight of documents where the same MeSH terms are found in the MEDLINE citation.

In the following sections, the ideas are explained in depth.

4.2 Named Entity Recognition

Named Entity Recognition (NER) is the process of automatically recognising specific terms and concepts within natural language. Even though researchers have produced NER systems with near-human performance on "normal" English text (Marsh and Perzanowski, 1998), NER tasks within the biomedical domain have proved difficult (Cohen and Hersh, 2005). The main challenges lies with the fact that there are no complete dictionaries for most types of biological entities; the same word or phrase can mean different things based on the context; and many biological entities have multiple names or annotations that refers to it.

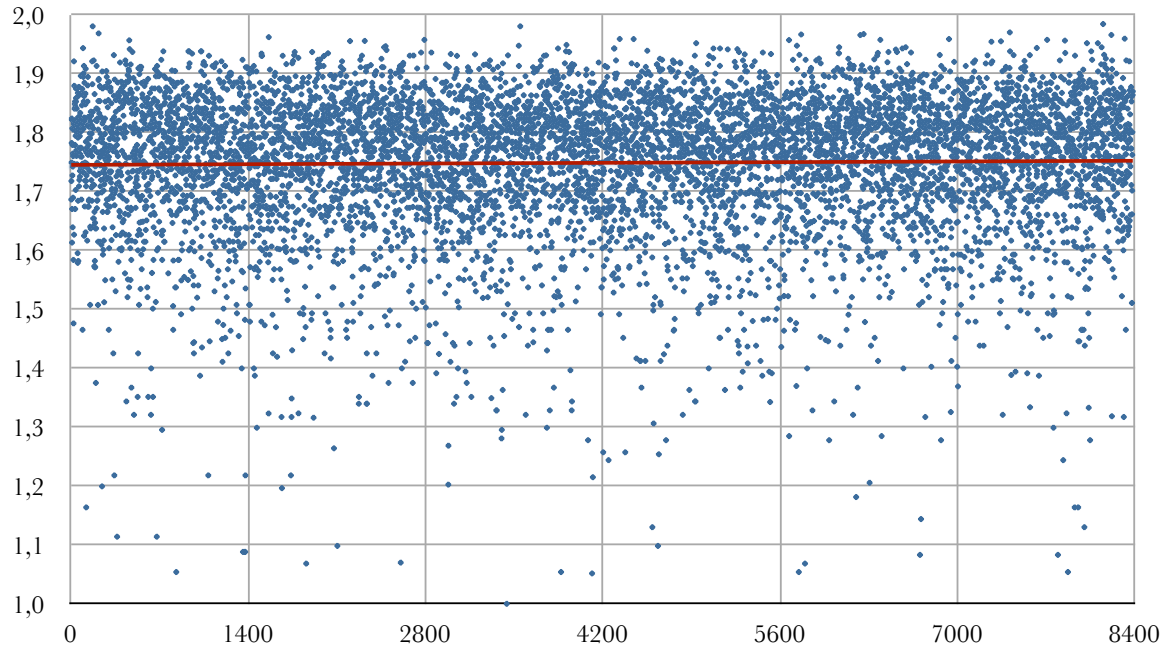

Figure 4.1: Distribution of the 8382 sentence boost values used during retrieval in Johannsson's prototype, using his Extended BM25 (GENIA) model. The red line is the average boost value (1.7485999). 4 sentences with a boost value of 0 are not shown.

There are four approaches to Named Entity Recognition (Cohen and Hersh, 2005; Wu et al., 2006; Mansouri et al., 2008):

Dictionary based Uses a static lexicon to identify known entities in the text.

Rule based Uses a set of human made rules to identify entities that matches.

Statistically based The systems look for patterns and relationships in already annotated training data, and using machine learning algorithms and statistical models it can detect similar concepts in new text. Requires a large amount of training data, but is very effective.

Hybrid Uses a combination of the above methods.

In our prototype we use three different NER models, but all are implemented using LingPipe[1]. Two of them are statistically based, namely *GeneTag* that uses a Hidden Markov Model[2], and *GENIA*, a token shaping model[3] that uses statistical machine learning to identify documents in the GENIA corpus[4]. Both models are available as pre-trained and pre-compiled "chunkers" from the LingPipe project (http://alias-i.com/lingpipe/web/models.html).

The last model uses a simple, dictionary based approach, where we have used the 2010 MeSH de-

[1] See http://alias-i.com/lingpipe/web/models.html.

[2] See http://alias-i.com/lingpipe/docs/api/com/aliasi/chunk/HmmChunker.html for implementation details.

[3] See http://alias-i.com/lingpipe/docs/api/com/aliasi/chunk/TokenShapeChunker.html for implementation details.

[4] A project that seeks to automatically extract information from the micro-biology domain. See http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/home/wiki.cgi for more information.

scriptors[5] to create the lexicon. We used a class from the LingMed project in the LingPipe sandbox[6] (com.aliasi.lingmed.mesh.MeshDictionaryCommand) to generate the chunker, which is compatible with the two previously mentioned chunkers.

4.2.1 Weighting the document fields

The chunkers described in the previous section all return a set of "chunks", each consisting of the term(s) that were identified as relevant and the type of the term. To use this information to calculate a weight for the current field, we use the size of the chunk set as the measure:

$$\text{fw}(f) \quad = \quad |R_f| \tag{4.1}$$

where $\text{fw}(f)$ is the field weight for the field f, R_f is the set that resulted from the chunking of the field, and $|R_f|$ is the size of the set. To avoid that longer fields dominate the weighting, we use the following normalisation formula, derived from Okapi BM25 (see Section 2.2.4.1) and Johannsson (2009):

$$\text{nfw}(f) \quad = \quad \frac{(k_1+1)\cdot(\text{fw}(f)+1)}{(\text{fw}(f)+1)+k_1\left(1-b+b\left(\frac{|f|}{\text{avgl}_f}\right)\right)} \tag{4.2}$$

where $\text{nfw}(f)$ is the normalised field weight, $|f|$ is the length and avgl_f is the average length of the field f. As in the Okapi BM25 formula, we set the constants to $k_1 = 2$ and $b = 0.75$. The field weight is 0 if no relevant terms are found, so we increment it by 1 to avoid that the weight zero out the document score.

We do the weighting during indexing (as the chunking is a somewhat time-consuming process) and sets the weight independently for each field of each document. This score is then multiplied with a length normalisation score and is saved in the index. When scoring queries during a search, we simply multiply the saved weight with the normal score calculated by the ranking model in use.

In Figure 4.2, the average field boost value for each of the 42255 documents in the collection can be seen. The average field boost value $\text{avgfw}(D)$ is defined as the boost value for each field $f_1 \ldots f_n \in F$ in document D as calculated by $\text{nfw}(f)$ in Equation 4.2, divided by the number of fields that are weighted ($|F|$). It is given by the following formula:

$$\text{avgfw}(D) \quad = \quad \frac{\sum_{i=1}^{n}\text{nfw}(f_i)}{|F|}. \tag{4.3}$$

The values in the graph were calculated at index time. The lowest boost value is 0.6931817, the highest is 2.4963796, and the average is 1.6932384. Please note that these values differ from the values actually used during retrieval, due to a limitation in Lucene (see Section 5.7 for more information).

4.2.2 Simple query expansion using NER and MeSH

The idea here is based on the concepts introduced in Section 3.1.3. We are not implementing this in a large scale, but wanted to see if we could improve matching of the MeSH fields if we used Named Entity Recognition using the MeSH dictionary chunker introduced in Section 4.2 to map query terms to MeSH terms. The implementation is simple; for each document we use NER to identify terms in the query that maps to MeSH descriptors. If any MeSH terms are found, we add them to the query when searching the MeSH field, and let the ranking model in use take care of the scoring and matching. An illustration of the process is found in Figure 4.3.

[5] Available from http://www.nlm.nih.gov/mesh/filelist.html.

[6] The sandbox contains LingPipe projects that are in development or are experimental. It is available from http://alias-i.com/lingpipe/web/sandbox.html.

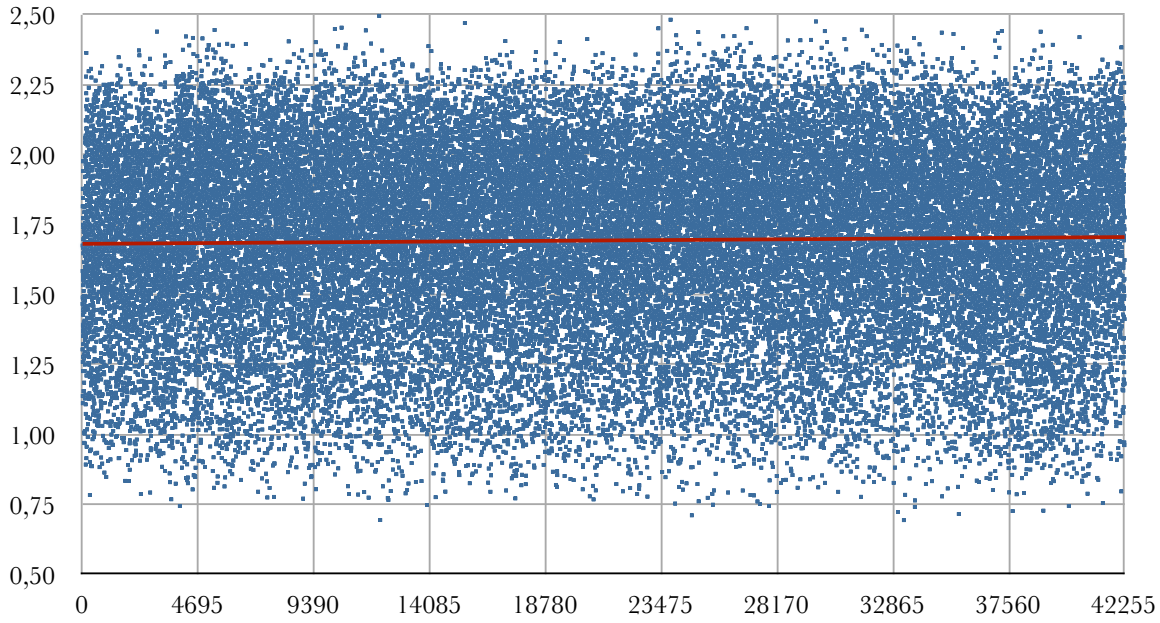

Figure 4.2: Distribution of the field boost values in our prototype as calculated during indexing, using the BM25F model extended with GeneTag. The red line is the average boost value (1.6932384).

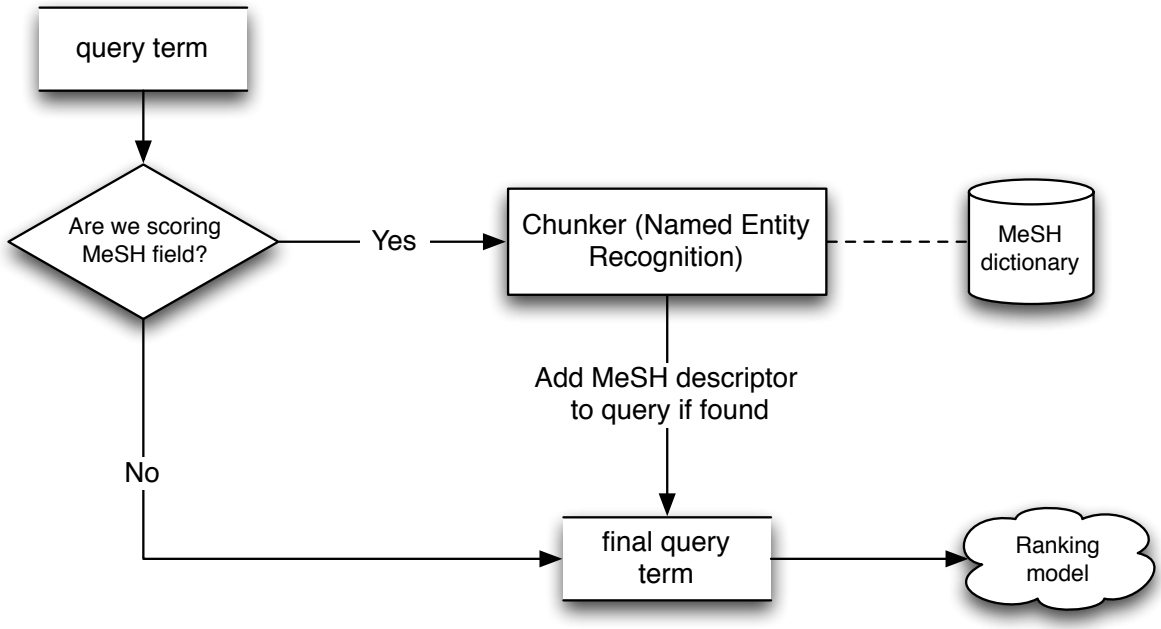

Figure 4.3: MeSH query expansion illustrated

An example using the MeSH dictionary chunker: if the chunker identifies the term `transgenesis` as relevant, the type for the chunk is `Gene Transfer Techniques`, which is the MeSH descriptor `transgenesis` maps to. `Gene Transfer Techniques` is then added to the query during the scoring process, but only when scoring the MeSH field.

4.3 Boosting specific document parts

While Johannsson only searched the abstract of the documents, we are taking advantage of the structured nature of MEDLINE citations. A MEDLINE citation contains many metadata fields, such as author(s), ISSN number, publication date, title, abstract and MeSH terms. Some of the fields are mandatory, while other are optional.

Ramampiaro (2010) and others have explored how you can weight document fields different to improve the ranking, for instance Ramampiaro's BioTracer search engine implemented this strategy in combination with Okapi BM25. However, Zaragoza et al. (2004); Robertson et al. (2004) shows that using Okapi BM25 across fields can lead to over-estimation of the importance of the terms. The idea is therefore to implement the weighting using the BM25F algorithm (outlined in Section 2.2.4.2) instead.

Chapter 5

Implementation

In this chapter we will elaborate on the technology we have used, and explain the implementation details of our prototype.

5.1 Technology

This section gives the reader an overview of the technologies and tools we have used when developing the prototype. Where possible, we have used existing libraries and open source tools to reduce development time.

5.1.1 Java

We have chosen to implement the prototype using Java 6[1], an object-oriented, open source, cross platform programming language developed by Sun Microsystems. Applications written in Java are compiled to byte code and run on a *Java Virtual Machine* (JVM), enabling the code to run on any computer architecture as long as there exists a JVM for the platform. Java is a very popular language, and a large number of both open source and commercial projects exists for the platform. The Java based products we have utilised either in the prototype or when developing it, are outlined in the following sections.

5.1.1.1 Eclipse

Eclipse[2] is an Integrated Development Environment (IDE) for use in software development. Written in Java itself, it was originally a developing environment only for Java, but there now exists plug-ins for a variety of other languages as well. Eclipse also has a solid base of other extensions and plug-ins, such as SubVersive (SVN support) and TeXlipse (LaTeXsupport).

5.1.1.2 Apache Lucene

Apache Lucene[3] is a high-performance text search engine library completely written in Java (but is available in other flavours as well). It provides the necessary libraries and API's to build a search engine accustomed to the developer's need.

Our prototype uses Lucene 3.0.1 for both indexing and retrieval.

[1] Available from http://java.sun.com/.
[2] Available from http://www.eclipse.org.
[3] Available from http://lucene.apache.org/.

5.1.1.3 LingPipe

LingPipe[4] is a text and linguistics toolkit written in Java, with a number of possibilities. A lot of the functionality in LingPipe is aimed at the field of biomedicine, and through the LingMed project available from the LingPipe sandbox (available from http://alias-i.com/lingpipe/web/sandbox.html), even more so. LingPipe provides commercial licences, but is free for educational purposes.

In our prototype we use the following functionality from LingPipe 4.0:

Parsing MEDLINE citations in XML format: LingMed includes a SAX XML parser made especially for this purpose. SAX (Simple API for XML) means that the parser reads the XML MEDLINE file[5] as a stream, and deals with elements as they appear in the stream by calling event handlers. This is in contrast to a DOM (Document Object Model) parser, which loads the entire XML file in memory and provides direct access to the elements in the document. The TREC 2004 Genomics Track contains about 15 gigabytes of citations in XML format, which obviously makes using a DOM parser a bad choice.

Named Entity Recognition: Explained in Section 4.2, we use NER libraries from LingPipe to identify biomedical terms in the text. In addition to NER models for biomedical entities, LingPipe have a model available for identifying news entities.

5.2 Implementation overview

In this section we provide the reader with an overview of our prototype and how it works. The implementation is inspired by the prototype made by Johannsson (2009), but is almost completely rewritten. Foremost, it only supports Lucene 3.0 or newer[6]. This version of Lucene introduced some large architectural changes to the API and lost a great deal of backwards compatibility[7]. Conversely, plug-ins and extensions to older versions of Lucene often breaks in combination with the new version, especially when it comes to deeper level modifications such as changing the ranking model. Johannsson also had to make some modifications directly to the Lucene source code. Our prototype is implemented solely by extending Lucene, not by altering it.

We have also implemented a BM25F ranking model as described in Section 2.2.4.2. There are few publicly available implementations of BM25 for Lucene, at least to our knowledge. Ramampiaro (2010); Johannsson (2009) uses one approach for BM25 weighting and Pérez-Iglesias et al. (2009) have published another approach that also supports BM25F. Neither of these approaches are compatible with Lucene 3.0, however. We describe the process of implementing a BM25F scorer model into Lucene 3.0 in Section 5.6.

The prototype created serves two main functions: indexing MEDLINE abstracts, and, given a query, retrieving MEDLINE abstracts that matches. The architecture defines one abstract class for indexers, and one for searchers. To implement a new ranking model, one only have to extend at least one, or preferably both, of these classes. Each model we have implemented uses its own index, but the architecture permits sharing a index between two or more ranking models if so desired.

The prototype can automatically run searches using predefined queries for one or more ranking models, and output the results in a format compatible with the *trec_eval* tool used in the TREC conferences to evaluate the participating IR systems. This is elaborated on in Chapter 6.

The following sections describes the different parts of the prototype in depth.

[4] Available from http://alias-i.com/lingpipe/.

[5] An example XML MEDLINE citation is found in Appendix B

[6] It *should* be compatible with Lucene 2.9.1 as well, but this is untested.

[7] See http://lucene.apache.org/java/3_0_1/changes/Changes.html for an overview of the changes.

5.3 Parsing MEDLINE citations

The parsing process is initialised in the class MyMedlineIndexer, which creates a MedlineParser (from the LingMed project) that parses the XML file containing the MEDLINE citations. MedlineParser requires a handler class, specified by the MedlineHandler interface. We create an instance of MyMedlineHandler (which implements this interface) which in turn calls the addDocument()-method of the model:

```
1  public class MyMedlineHandler implements MedlineHandler {
2
3      private MedlineIndexWriter indexWriter;
4
5      public MyMedlineHandler(MedlineIndexWriter indexWriter) {
6          this.indexWriter = indexWriter;
7      }
8
9      @Override
10     public void delete(String pmid) {
11         // Unsupported
12     }
13
14     @Override
15     public void handle(MedlineCitation citation) {
16         indexWriter.addDocument(citation.article().articleTitleText(), ↩
               →abstrct.textWithoutTruncationMarker(), citation.pmid(), ↩
               →citation.meshHeadings());
17     }
18 }
```

A model is a subclass of MedlineIndexWriter. The architecture is outlined in the class diagram in Figure 5.1.

5.4 Indexing

As the MEDLINE citations are parsed, they are being added to a index for later look up. We use Lucene (see Section 5.1.1.2) for this task, using a standard Lucene IndexWriter object. In the case of our BM25F implementation (BM25FIndexWriter), we have used the following code to initialise the index:

```
1  @Override
2  protected IndexWriter initializeIndexWriter() throws IOException {
3      (...)
4      IndexWriter writer = new IndexWriter(dir, AnalyzerFactory.↩
           →getBM25MedlineAnalyzer(), true, IndexWriter.MaxFieldLength.LIMITED↩
           →);
5      writer.setSimilarity(new BM25Similarity());
6      return writer;
7  }
```

This should be fairly straightforward, but two details are worth mentioning:

The Analyzer: The method AnalyzerFactory.getBM25MedlineAnalyzer() returns a regular Lucene StandardAnalyzer , but with custom stop words[8]. We have implemented it as a singleton factory to keep the number

[8] Instead of using the default stop word list that comes with Lucene, we use the list of stop words recommended by PubMed, available from http://www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=helppubmed&part=pubmedhelp&rendertype=table&id=pubmedhelp.T43. See Section 2.1 for a short primer on stop words.

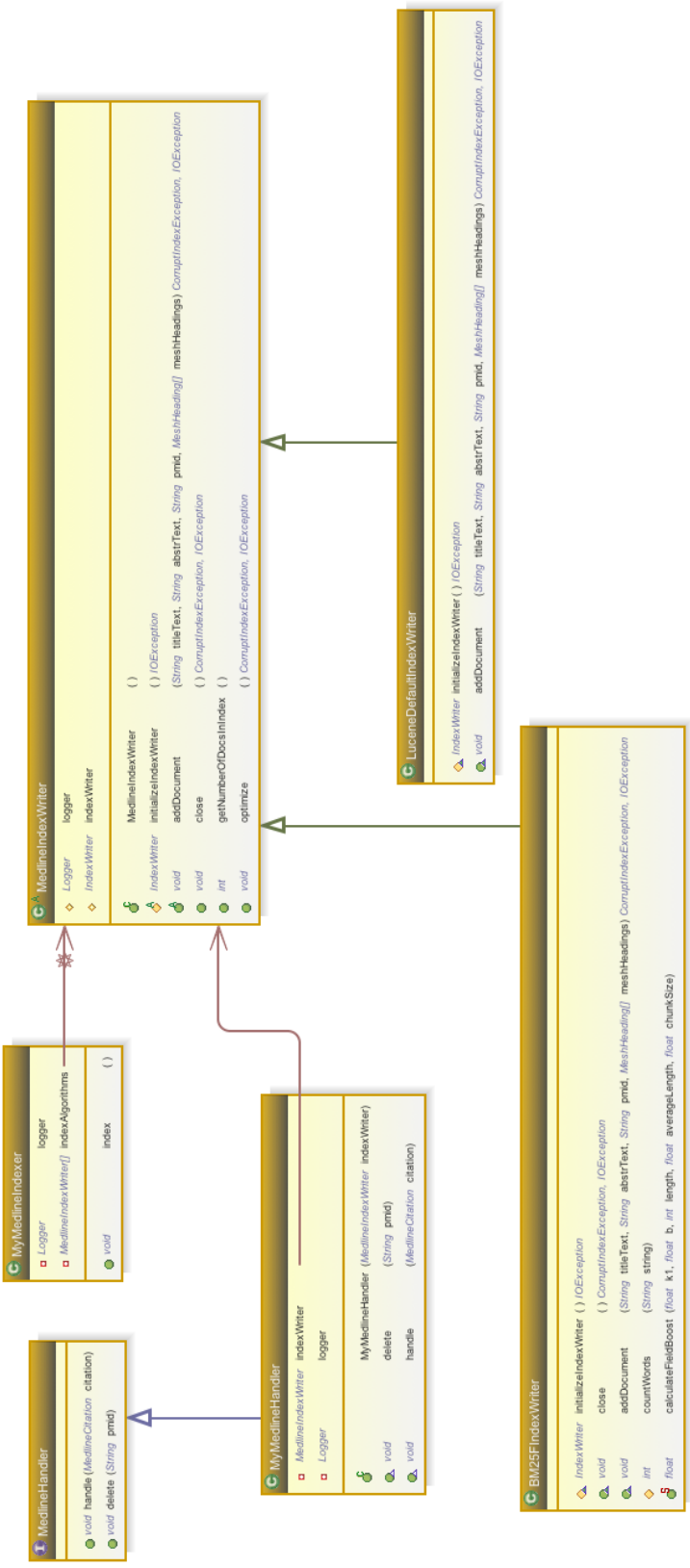

Figure 5.1: Class diagram of the classes used during indexing

of instances at one. There is also an AnalyzerFactory.getStandardMedlineAnalyzer(), but in the final prototype, the two methods are identical and returns the same StandardAnalyzer. The main reason for implementing it this way is to increase modifiability and extendability for the theoretical future additions of new ranking models.

The Similarity: The method writer.setSimilarity(new BM25Similarity()) creates a new BM25Similarity class, which extends the DefaultSimilarity class from Lucene and overrides some of its methods. This class affects the scoring of the documents, and might have to be overridden when using another ranking model than Lucene's default vector space model. At indexing time, the following methods from the Similarity class are used to create a normalisation value (The Apache Lucene project, 2009):

$$\text{norm}(t,d) \quad = \quad \text{doc.getBoost()} \cdot \text{lengthNorm(field)} \cdot \prod_{f' \in d} \text{f.getBoost()} \tag{5.1}$$

where $f' \in d$ are the fields f in document d that are named the same. No documents in our collection have duplicate field names.

Details of the BM25Similarity class can be found in Section 5.6.

See Figure 5.1 for an overview of the classes used during indexing.

5.4.1 Field boosting using NER weighting

Named Entity Recognition is a time-consuming process, and has to be done during indexing. We use three different different NER models in the prototype, as explained in Section 4.2.1. All of the models are stored as serialized Java objects that implements the Chunker interface from LingPipe, and are loaded into the application at run-time using the code (Chunker)AbstractExternalizable.readObject(serializedObject). The following code creates a document and a field, chunks the content of the field, returns a Set of Chunk objects, and uses the size of this set to calculate a normalised boost value for the single field, using the formula in Equation 4.2. It finally adds the boost value to the field, the field to the document, and the document to the index:

```
1  public void addDocument(String fieldText) {
2      Document doc = new Document();
3      Field field = new Field("exampleField", fieldText, Field.Store.YES, ↩
           → Field.Index.ANALYZED);
4      Set<Chunk> chunkSet = chunker.chunk(fieldText).chunkSet();
5      field.setBoost(calculateFieldBoost(kl, b, titleLength, ↩
           → averageTitleLength, chunkSet.size()));
6      doc.add(field);
7      indexWriter.addDocument(doc);
8  }
9
10 public static float calculateFieldBoost(float kl, float b, int length, float↩
       → averageLength, float chunkSize) {
11     return ((kl + 1) * (chunkSize + 1)) / (kl * (1 - b + (b * (length / ↩
           → averageLength))) + (chunkSize + 1));
12 }
```

calculateFieldBoost() is thus called one time for each document, and produces the values seen in Figure 4.2. We chose to use the native Lucene functionality for storing the field boost. We have outlined the drawbacks of this method in Section 5.7. The benefit is that no extensions to Lucene are needed, and thus not hurting performance.

5.4.2 Field boosting using multiple NER models

We have also tested if there is any improvement in performance when combining different NER models. This is implemented by first chunking the title field and the abstract field normally, but using both GENIA, GeneTag and MeSH to identify biomedical entities. After removing duplicates (entities that are found by two or more models), we use the size of the set as the boost value for the title and the abstract field in the same way as earlier. Additionally, if either the matched entity or the type that it maps to is found in the MeSH field of the citation, we increment the MeSH field boost by one. The idea is that because MeSH field only contains MeSH terms which are probably identified by all chunkers, we have to weight the field in another way. The implementation is as follows:

```
Chunker[] chunkers = new Chunker[] {
        chunkerGeneTag, chunkerGenia, chunkerMesh
};
String[] searchStrings = new String[] {
        titleText, abstrText
};
int localCounter = 0;
HashSet<String> meshSet = new HashSet<String>();

for (String searchString : searchStrings) {
    HashSet<String> querySet = new HashSet<String>();
    for (Chunker chunker : chunkers) {
        for (Chunk chunk : (chunker.chunk(searchString).chunkSet())) {
            String match = searchString.substring(chunk.start(), chunk.end()
                →);
            querySet.add(match);
            if (meshString.contains(match)) {
                meshSet.add(match.toLowerCase());
            }
            if (meshString.contains(chunk.type())) {
                meshSet.add(chunk.type().toLowerCase());
            }
        }
    }
    if (localCounter == 0) {
        titleField.setBoost(calculateFieldBoost(k1, b, titleLength, ↩
            →averageTitleLength, querySet.size()));
    } else if (localCounter == 1) {
        abstractField.setBoost(calculateFieldBoost(k1, b, abstractLength, ↩
            →averageAbstractLength, querySet
                .size()));
        meshField.setBoost(calculateFieldBoost(k1, b, meshLength, ↩
            →averageMeshLength, meshSet.size()));
    }
    localCounter++;
}
```

5.5 Retrieval

The main classes involved in the retrieval are very similar to the indexing classes, and can be seen in Figure 5.2. We have an abstract class `MedlineSearcher` that all implemented retrieval models have to extend. At minimum they have to implement the methods `initializeIndexSearcher()` and `parseQuery()`. The framework supports executing searches automatically using the `TrecEval` class. One can choose one

or more of the implemented ranking models to use during the search, which outputs the results of each ranking model in a text file compatible with the `trec_eval` tool. The three first lines of one of the result files generated by our prototype looks like this:

```
1   Q0  12730111    1   20.827084   jervi
1   Q0  12393173    2   17.148758   jervi
1   Q0  12377801    3   16.286715   jervi
```

where each column is defined as follows (Hersh, 2005):

- The first column is the topic number (1–50).
- The second column is the query number within that topic. This is currently unused and must always be Q0.
- The third column is the official PubMedID of the retrieved document.
- The fourth column is the rank at which the document is retrieved.
- The fifth column shows the score (integer or floating point) that generated the ranking. This score MUST be in descending (non-increasing) order. The `trec_eval` program ranks documents based on the scores, not the ranks in column 4. If a submitter wants the exact ranking submitted to be evaluated, then the SCORES must reflect that ranking.
- The sixth column is called the "run tag" and must be a unique identifier across all runs submitted to TREC.

5.6 Implementing a BM25F scoring model into Lucene

As briefly mentioned in Section 5.2, existing Lucene extensions that implements BM25F are sparse. The only publicly available implementation we could find was by Pérez-Iglesias et al. (2009), but it was not compatible with Lucene 3.0. A patch, however, made available from the Lucene issues tracker[9], claimed to support Lucene 3.0. We tested the patch, and found that while it sort of worked, Pérez-Iglesias et al. had implemented a lot of classes[10] that seemed to just mimic the official `BooleanScorer` included with Lucene, and our testing showed that the implementation seemed poor. Using the 50 boolean queries made by Johannsson (2009) (each corresponding to a topic in the TREC 2004 Genomics Track), only 34 of the 50 searches returned any results at all.

To fix the problem, we removed the extraneous `*BooleanScorer` classes and modified the code to use standard Lucene classes instead. To accomplish this, we use a "query rewriter" approach introduced by Johannsson (2009): each query (all of them boolean in nature) is parsed by Lucene's `MultiFieldQueryParser`, which (in the case of a boolean query) returns a `BooleanQuery` consisting either of more `BooleanQuery` objects or several `TermQuery` objects. We break down the query and replace each occurrence of `TermQuery` with our `BM25TermQuery` object, as outlined in the following listing:

```
1  @Override
2  public Query parseQuery(String query) throws ParseException {
3      if (parser == null) {
4          parser = new MultiFieldQueryParser(Version.LUCENE_30, fields, ↩
               →AnalyzerFactory.getBM25MedlineAnalyzer(), boosts);
```

[9] See http://issues.apache.org/jira/browse/LUCENE-2091.

[10] AbstractBooleanScorer, MatchAllBooleanScorer, MustBooleanScorer, NotBooleanScorer and ShouldBooleanScorer

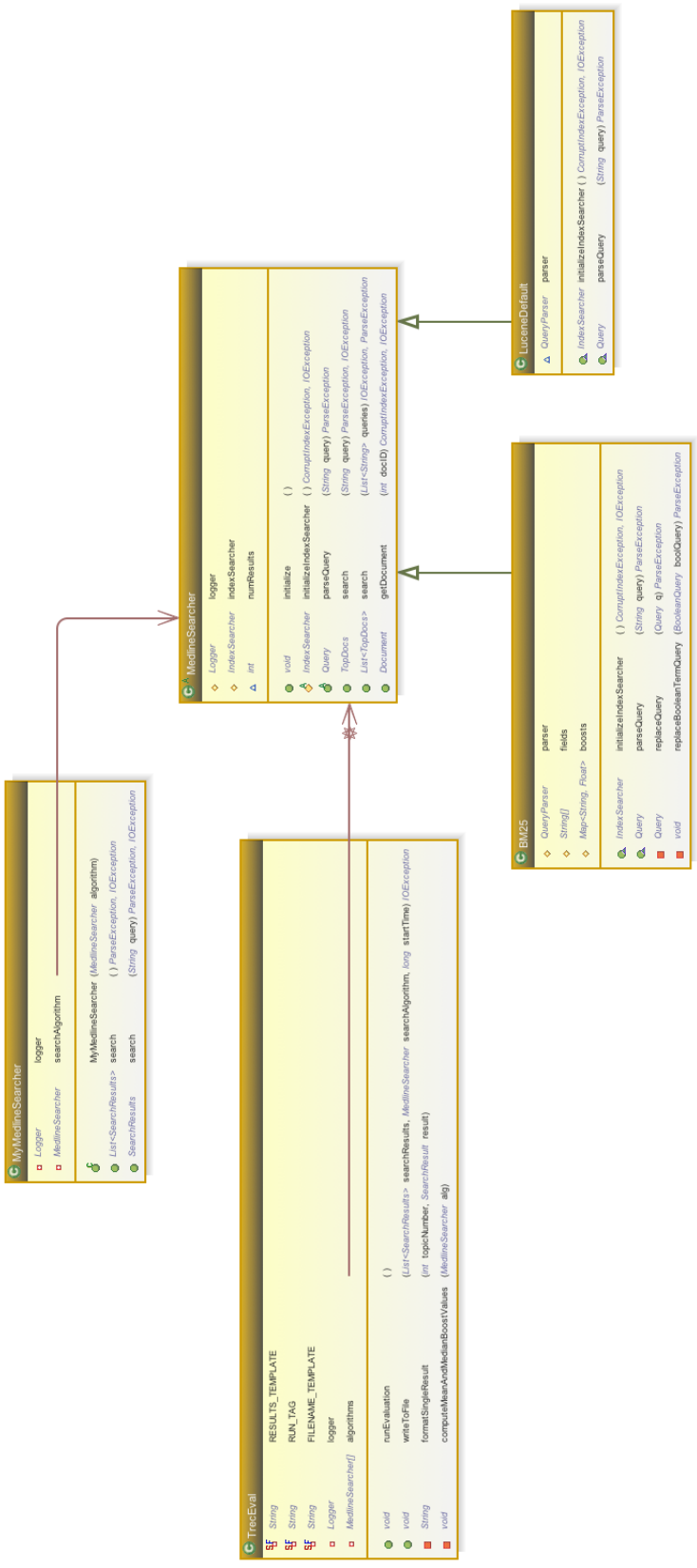

Figure 5.2: Class diagram of the classes used during retrieval

```
    }
    Query parsedQuery = parser.parse(query);
    return replaceQuery(parsedQuery);
}

private Query replaceQuery(Query q) throws ParseException {
    if (q instanceof TermQuery) {
        TermQuery tq = (TermQuery) q;
        BM25TermQuery bm25tq;
        if (fields == null) // BM25
            bm25tq = new BM25TermQuery(tq.getTerm());
        else { // BM25F
            float[] boostsArray = new float[boosts.size()];
            for (int i = 0; i < boostsArray.length; i++) {
                boostsArray[i] = boosts.get(fields[i]);
            }
            bm25tq = new BM25TermQuery(tq.getTerm(), fields, boostsArray,
                →BM25FParameters.getBParam());
        }
        bm25tq.setBoost(tq.getBoost());
        return bm25tq;
    } else if (q instanceof BooleanQuery) {
        replaceBooleanTermQuery((BooleanQuery) q);
        return q;
    } else {
        throw new ParseException("unsupported query type " + q.getClass() +
            →": " + q.toString());
    }
}

private void replaceBooleanTermQuery(BooleanQuery boolQuery) throws
    →ParseException {
    for (BooleanClause clause : boolQuery.getClauses()) {
        Query q = clause.getQuery();
        Query newQ = replaceQuery(q);
        clause.setQuery(newQ);
    }
}
```

We then start the search. In a typical Lucene search, a `Query` (in our prototype this is the `BooleanQuery` returned from the `parseQuery()` method seen above) is passed to a `Searcher` object that executes the search. Each `Query` object returns a `Weight` object used during ranking when the method `createWeight(Searcher searcher)` is called. This way, each `Query` object returns its own weighter/scorer. In the case of our `BM25TermQuery` objects, the scoring is performed by either `BM25Scorer` or `BM25FScorer` depending on whether or not we are performing a multi-field search, in which case the latter of the two scorers are used.

The implementation of the `BM25FScorer` is implemented as described by Pérez-Iglesias et al. (2009), and the code that cohere with Equation 2.17 follows:

```
@Override
public float score() throws IOException {
    float acum = 0f;

    for (int i = 0; i < len; i++) {
        if (this.termDocs[i].doc() == doc) {
```

```
7               float av_length = this.averageLengths[i];
8               float fieldNorm = Similarity.decodeNorm(norms[i][this.docID()]);
9               float length = 1 / (fieldNorm * fieldNorm);
10
11              float aux = this.bParam[i] * length / av_length;
12
13              aux += (1 - this.bParam[i]);
14              acum += (this.termBoost * this.boosts[i] * this.termDocs[i].freq↩
                    →()) / aux;
15          }
16      }
17
18      acum /= (this.K1 + acum);
19      acum *= this.idf;
20      return acum;
21  }
```

There are some limitations in the implementation. Firstly, it only supports boolean queries, and secondly, BM25F requires the inverse document frequency (idf) of the document, but Lucene always computes idf on a field level. Without rewriting Lucene, this is not possible. When doing multi field searching using BM25F, the prototype therefore uses the workaround of using the idf of the field with the longest average length, in our case the abstract field.

5.7 Limitations of field boosting using Lucene

Instead of implementing a new index containing the field boosts calculated by the named entity recognition (see Section 4.2.1 and 4.2.1), we chose to save the boost value in the standard Lucene index, using the method `field.setBoost(float)`. The field boost value is multiplied with the document boost and a length normalisation value (as seen in Equation 5.1) to a "norm", *before* being added to the index, so the actual boost value is lost. In addition, the value is encoded as a single `byte` to save memory at search time (because all norms have to be loaded in memory during the search). An effect of the choice of using a `byte` to save the norm value is that precision is lost, that is $\text{decode}(\text{encode}(x)) \neq x$. The lack of precision is apparent in Figure 5.3, especially if you compare this figure to Figure 4.2 which illustrates the original boost values during indexing. The Apache Lucene project (2009) notes, however, that:

> The rationale supporting such lossy compression of norm values is that given the difficulty (and inaccuracy) of users to express their true information need by a query, only big differences matter.

While this might be true in many IR settings, this is not the case for this prototype, as all the queries are carefully constructed according to the topics given in the TREC Genomics 2004 track.

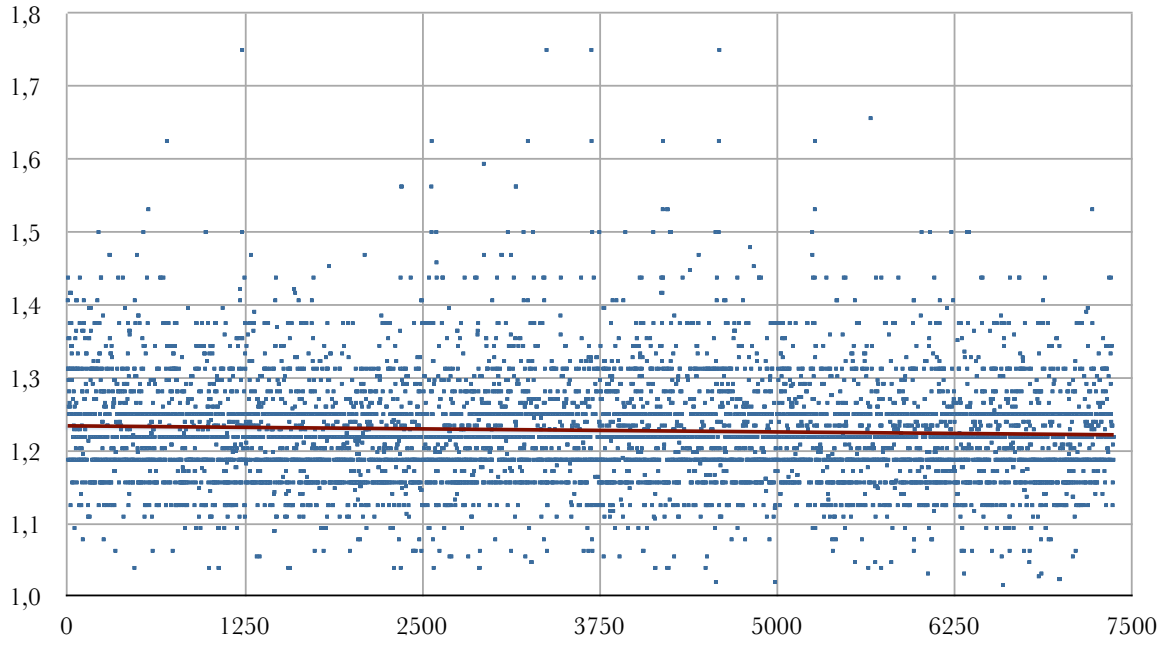

Figure 5.3: Distribution of the field boost values in our prototype during search, using the BM25F model extended with GeneTag. The red line is the average boost value.

Chapter 6

Evaluation

This chapter explains the evaluation method we have used for measuring the performance of the prototype, and presents our results.

6.1 The document collection

To be able to compare our results with Johannsson's results, we used the same test collection in our prototype: a subset based on the TREC 2004 Genomics track test collection, which again consists of a subset of the MEDLINE database. The citations used in the collection were published, last reviewed or created between 1994 and 2003, totaling 4,591,008 records. 42,255 of these records have been judged against the 50 topics or "information needs" published in the TREC. We only use these judged records in our prototype. The justification for doing this is that indexing the entire document collection would be to time consuming when using Named Entity Recognition, and the fact that Johannsson (2009) also used the same subset in his prototype, as previously mentioned. We used a filtering tool developed by Johannsson to filter out all unjudged documents from the test collection, leaving 42,255 citations. As discussed in section 2.3.2.6, this allows us to use performance measures that are not robust when used on incomplete judgement sets (such as MAP, defined in Section 2.3.2.5), bypassing the problem completely.

6.2 Queries

The topics published in the TREC 2004 Genomics tracks consists of 50 information needs and a set of relevance judgements for each topic. The relevance judgement consist of a list of PubMed ID's that are judged against each topic and a judgement of relevance (not relevant, possibly relevant or definitively relevant). In total, there are 48753 judged records, giving an average of 975 per topic (ranging from 476 to 1450).

An example topic can be seen in Appendix C.

Our prototype only supports boolean queries due to limitations in the BM25 and BM25F implementations. We thus have to convert the above information needs into structured queries. We have tested a few different approaches to creating these queries: using the queries created by Johannsson (but slightly modifying a few of the queries because of low performance) and automatically generating the queries based on the information needs. To do this, we tried to identify biomedical relevant entities in each topic (bot title, need and context fields were used) using Named Entity Recognition, and use each identified entity in the query.

The best result using default Lucene scoring (VSM) and automatically created queries had a MAP score of 0.1988, compared to 0.2371 using Johannsson's manually crafted boolean queries. For some topics, however, the automated method scored better than the manual method (see Figure **??**), so we extracted these queries and used them where applicable, and achieved a MAP score of 0.2591 (still using

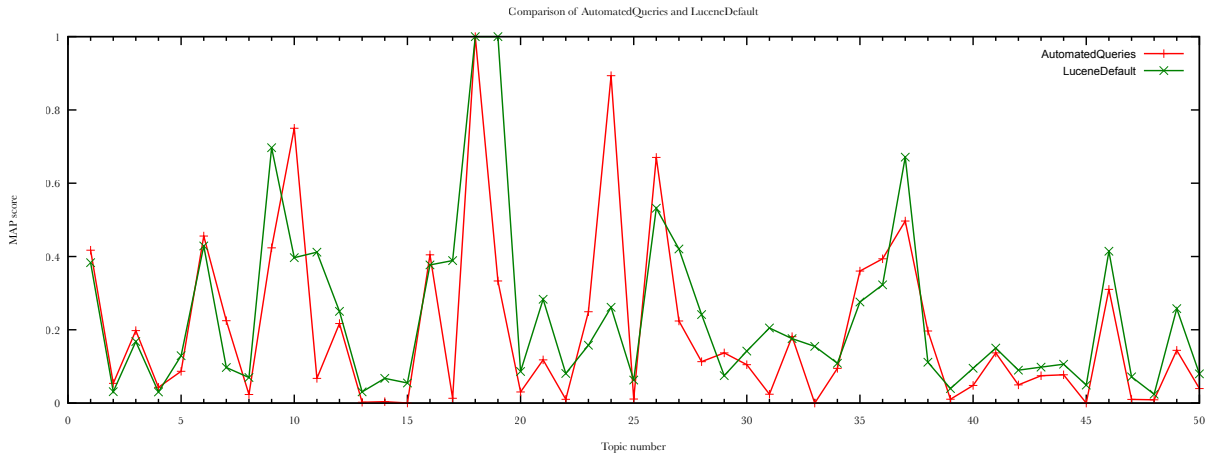

Figure 6.1: Comparison of manually (green) and automatically (red) generated queries, using Lucene's default scoring model (VSM).

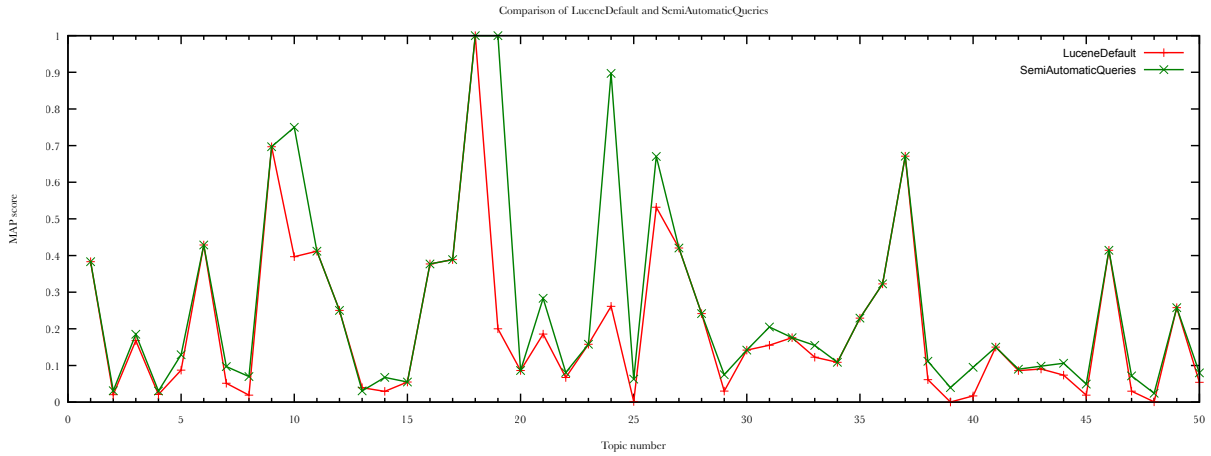

Figure 6.2: Comparison of the new queries (green) and the old queries used by Johannsson (2009) (red), using Lucene's default scoring model (VSM).

the VSM model). The final queries can be seen in Appendix A, and how they compare to the Johannsson's approach is illustrated in Figure 6.2.

6.3 Evaluation method

To evaluate the performance of our prototype, we use the tool `trec_eval` 9.0[1], which is the same tool used to evaluate TREC runs. By providing the tool with a result set and relevance judgements, it will output statistics about the performance, including mean average precision (MAP), R-precision and bpref, as discussed in Section 2.3.2.

Johannsson (2009) developed a web page interface for running the `trec_eval` tool and comparing results. To avoid "reinventing the wheel", we chose to use the same interface during our evaluation. The interface is developed using Perl for the back-end and HTML/Javascript for the front-end.

[1] Available from http://trec.nist.gov/trec_eval/.

6.4 Environment

The environment on which the prototype was developed and the tests run, is outlined in Table 6.1. Note that neither the hardware or the software should affect the results (at least in theory), and are included for reference only. The Java VM arguments are important though, because the MeSH NER chunker uses more than the standard allowed memory for Java, and will not run without these arguments.

CPU:	Intel Core 2 Duo 2.4 GHz
Memory:	2x2GB
Disk:	160GB SATA 5400RPM
Operating system:	Mac OS X 10.6.4
Java version:	Sun Java 6 (1.6.0_20) 64-bit
Java VM arguments:	-Xms128m -Xmx512m

Table 6.1: This table lists the hardware and software used for evaluating the prototype.

6.5 Results

In this section we present the results from the prototype. Table 6.2 and 6.3 shows the averaged results for the different models. We have included the results from Johannsson's Extended BM25 (GENIA) for comparison reasons. The results are illustrated in the charts in Figure 6.3 and 6.4. The model *Extended BM25F (GeneTag and GENIA)* is the approach described in Section 5.4.2, where we used all three NER models at once during the boosting. We have measured the score after returning 100 and 1000 documents, the latter being the default value used in TREC runs, but the former maybe a bit more representative as few or no users actually will sift through a 1000 results. In fact, as mentioned in Section 2.3.2.3, very few users actually looks at more than 10 results.

	MAP	bpref	R-precision	P(5)	P(100)
Johannsson's Extended BM25 (GENIA)	0.1992	0.3205	0.2564	0.5320	0.3480
Lucene Default (VSM)	0.2591	0.3745	0.3283	0.6000	0.3938
BM25F	0.2727	0.3708	0.3337	0.6280	0.4026
Extended BM25F (GeneTag)	0.2713	0.3698	0.3318	**0.6560**	0.4040
Extended BM25F (GENIA)	0.2716	0.3683	0.3322	**0.6560**	0.4002
Extended BM25F (MeSH)	0.2722	0.3696	0.3340	0.6480	0.4016
Extended BM25F (GeneTag and GENIA)	**0.2770**	**0.3808**	**0.3351**	0.6440	**0.4080**

Table 6.2: Evaluation measures using the 100 first results. The best results are in bold.

6.6 Simple MeSH Query Expansion

When using the simple MeSH query expansion as explained in 4.2.2, we found a very small increase in the MAP score, averaging about 0.75%. The increase is neglectable, but as it did not decrease the scores, we decided to leave it on for all measures. An comparison can be seen in 6.5.

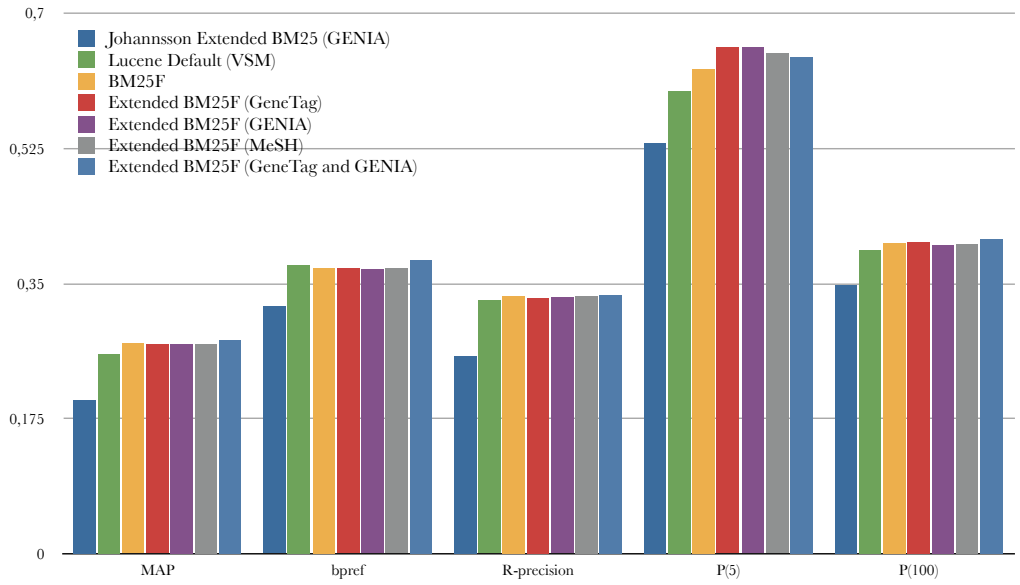

Figure 6.3: Illustration of the evaluation measures using the 100 first results.

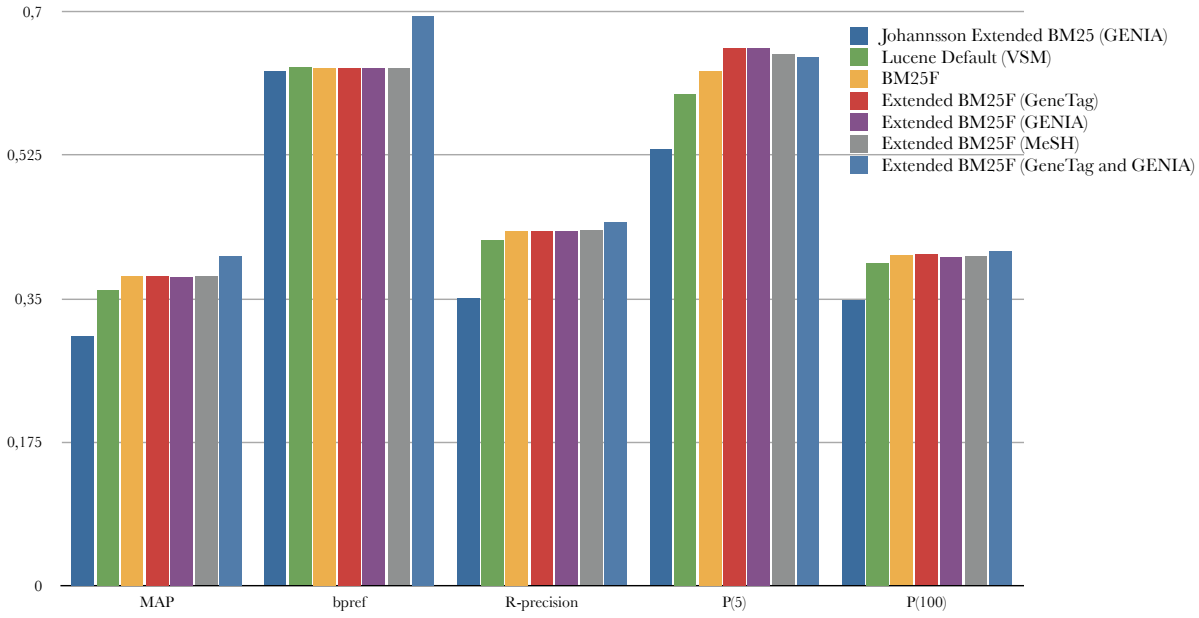

Figure 6.4: Illustration of the evaluation measures using the 1000 first results.

	MAP	bpref	R-precision	P(5)	P(100)
Johannsson Extended BM25 (GENIA)	0.3045	0.6275	0.3505	0.5320	0.3480
Lucene Default (VSM)	0.3606	0.6318	0.4216	0.6000	0.3938
BM25F	0.3779	0.6317	0.4326	0.6280	0.4026
Extended BM25F (GeneTag)	0.3777	0.6312	0.4322	**0.6560**	0.4040
Extended BM25F (GENIA)	0.3769	0.6309	0.4320	**0.6560**	0.4002
Extended BM25F (MeSH)	0.3772	0.6308	0.4336	0.6480	0.4016
Extended BM25F (GeneTag and GENIA)	**0.4023**	**0.6951**	**0.4434**	0.6440	**0.4080**

Table 6.3: Evaluation measures using the 1000 first results. The best results are in bold.

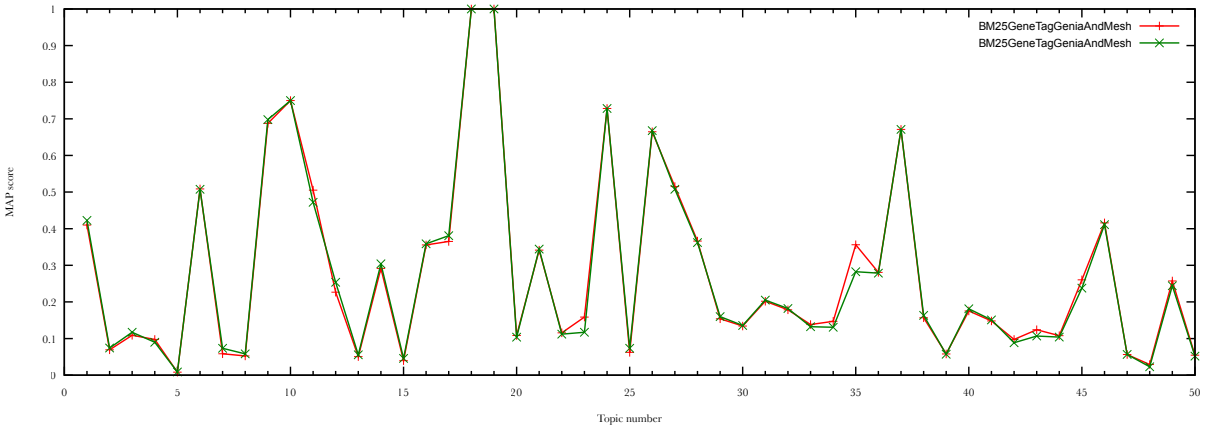

Figure 6.5: Comparison of MeSH Query Expansion turned on (red) or off (green).

6.7 Boosting the fields differently

As described in Section 4.3, we tried to boost the sections independently. While Ramampiaro (2010) found that weighting the title twice as much as the abstract field yielded good results, we have actually experienced the exact opposite. When weighting the abstract twice as much as the title and the MeSH fields, we see an 5.67% increase compared to when weighting the title twice as much as the abstract, using default Lucene weighting (VSM), as illustrated in Figure 6.6.

6.8 Discussion

As seen in the previous sections, we can see that all the BM25F based methods significantly outperforms standard Lucene weighting using the vector space model. When measuring the first 100 documents, we see a 6.9% increase in MAP between standard Lucene weighting and the Extended BM25F (GeneTag and GENIA) model, and when measuring the first 1000 results, we see an 11.5% increase. The increase of the bpref value is 10%.

All the documents in our relatively small document collection are judged and contains all three fields necessary to perform the proposed weighting. This is, however, a constructed and limited corpora. In practice, some of the MEDLINE citations available through PubMed are missing abstracts, and the size and the growth of the database makes NER weighting nearly impossible. Except when using the GENIA NER model, indexing the 42255 citations in our collection varied between 20 seconds and 2:30 minutes on our hardware. The GENIA model, however, used more than an hour to complete. The generality of

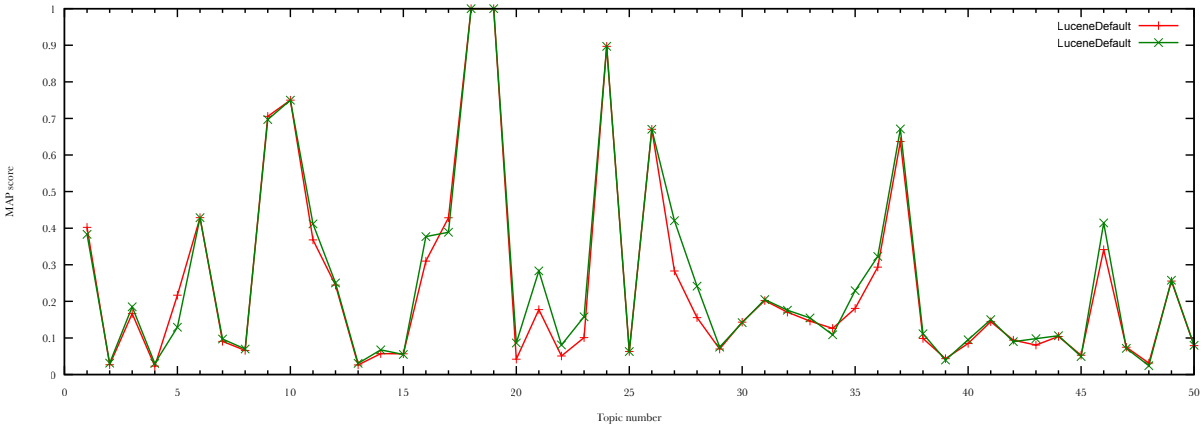

Figure 6.6: Comparison of boosting values. Green: boosting abstract field by 2, other fields by 1. Red: boosting title field by 2, other fields by 1.

this research can therefore be a point of discussion, but it should be applicable on smaller corpora as well as collections that have few changes, and thus are rarely in need of re-indexing.

We also observed improvements just by changing the weighting model from using the vector space model to BM25F, and this approach is neither time-consuming during indexing or in need of the abstract field of a MEDLINE citation. We conclude that using BM25F is a significant improvement over using the Okapi BM25 model or VSM, at least on structured documents.

Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis we have implemented a prototype that uses the BM25F weighting model in combination with named entity recognition (NER) to identify biomedical terms in a collection. We use this knowledge to boost fields with a lot of identified terms more than fields with fewer identified terms. The NER models we have used are implemented using three different approaches; GENIA (which uses a token shaping model), GeneTag (which uses a hidden Markov model) and MeSH (which uses a dictionary model). We found only small improvements in MAP when used by themselves, but when all three models were combined, we observed a significant boost in performance (up to 11.5%). Using BM25F for scoring also increased the performance of the prototype compared to using Okapi BM25 or VSM.

7.2 Further work

The improvements seen in the performance of our prototype is promising, and may be worth further work. A major limitation of the implemented BM25F model is that it only supports simple boolean queries. A field that could be researched further is extending the BM25F implementation to support phrase queries, including for instance wildcard support. Using other NER models could also prove beneficial.

Bibliography

Jesse Alpert and Nissan Hajaj. Official Google Blog: We knew the web was big..., July 2008. Available from http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html. [Cited April 29, 2010].

M. Bacchin and M. Melucci. Symbol-based query expansion experiments at TREC 2005 Genomics Track. I *The Fourteenth Text REtrieval Conference Proceedings (TREC 2005)*. Citeseer, 2005.

Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. I *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32, New York, NY, USA, 2004. ACM. ISBN 1-58113-881-4. doi: 10.1145/1008992.1009000.

Fabrice Camous, Stephen Blott, and Alan F. Smeaton. On Combining Text and MeSH Searches to Improve the Retrieval of MEDLINE documents. I *CORIA*, pages 271–282. Université de Lyon, 2006. ISBN 2-9520326-6-1.

Cyril Cleverdon. The Cranfield tests on index language devices. I *Aslib Proceedings*, volum 19, pages 173–192. MCB UP Ltd, 1967.

Aaron M. Cohen and William R. Hersh. A survey of current work in biomedical text mining. *Brief Bioinform*, 6(1):57–71, 2005. doi: 10.1093/bib/6.1.57. Available from http://bib.oxfordjournals.org/cgi/content/abstract/6/1/57.

Andreas Doms and Michael Schroeder. GoPubMed: exploring PubMed with the Gene Ontology. *Nucl. Acids Res.*, 33(2):W783–786, 2005. doi: 10.1093/nar/gki470. Available from http://nar.oxfordjournals.org/cgi/content/abstract/33/suppl_2/W783.

Gunther Eysenbach and Christian Kohler. How do consumers search for and appraise health information on the world wide web? Qualitative study using focus groups, usability tests, and in-depth interviews. *BMJ*, 324(7337):573–577, 2002. doi: 10.1136/bmj.324.7337.573. Available from http://www.bmj.com/cgi/content/abstract/324/7337/573.

William Hersh. TREC 2004 Genomics Track Final Protocol, February 2005. Available from http://ir.ohsu.edu/genomics/2004protocol.html. [Cited June 13, 2010].

Jorge R Herskovic, Len Y Tanaka, William Hersh, and Elmer V Bernstam. A Day in the Life of PubMed: Analysis of a Typical Day's Query Log. *Journal of the American Medical Informatics Association*, 14(2):212–220, 2007. doi: 10.1197/jamia.M2191. Available from http://jamia.bmj.com/content/14/2/212.abstract.

Dagur Valberg Johannsson. Biomedical Information Retrieval based on Document-Level Term Boosting. Master's thesis, Norwegian University of Science and Technology, Department of Computer and Information Science, June 2009.

Hyunki Kim and Su-Shing Chen. Ontology search and text mining of medline database. *Data Mining in Biomedicine*, pages 177–189, 2007. doi: 10.1007/978-0-387-69319-4_11. Available from http://dx.doi.org/10.1007/978-0-387-69319-4_11.

Michael Krauthammer and Goran Nenadic. Term identification in the biomedical literature. *Journal of Biomedical Informatics*, 37(6):512 – 526, 2004. ISSN 1532-0464. doi: 10.1016/j.jbi.2004.08.004. Available from http://www.sciencedirect.com/science/article/B6WHD-4DH2B7N-2/2/3802f0e5a1f604886d6cd075d6cf5683. Named Entity Recognition in Biomedicine.

Dominik Kuropka. Modelle zur Repräsentation natürlichsprachlicher Dokumente. *Ontologiebasiertes Information-Filtering und-Retrieval mit relationalen Datenbanken. Advances in Information Systems and Management Science, bd*, 10:264, 2004.

Zhiyong Lu, Won Kim, and W. John Wilbur. Evaluation of query expansion using MeSH in PubMed. *Information Retrieval*, 12(1):69–80, February 2009. doi: 10.1007/s10791-008-9074-8. Available from http://dx.doi.org/10.1007/s10791-008-9074-8.

Christopher D. Mannning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Alireza Mansouri, Lilly Suriani Affendey, and Ali Mamat. Named Entity Recognition Approaches. *IJCSNS International Journal of Computer Science and Network Security*, 8(2):339–344, February 2008.

Elaine Marsh and Dennis Perzanowski. MUC-7 EVALUATION OF IE TECHNOLOGY: Overview of Results. I *Proceedings of MUC-7*, 1998. Available from http://acl.ldc.upenn.edu/muc7/M98-0002.pdf.

Hans-Michael Müller, Eimear E Kenny, and Paul W Sternberg. Textpresso: An ontology-based information retrieval and extraction system for biological literature. *PLoS Biol*, 2(11):e309, 09 2004. doi: 10.1371/journal.pbio.0020309. Available from http://dx.doi.org/10.1371%2Fjournal.pbio.0020309.

Joaquín Pérez-Iglesias, José R. Pérez-Agüera, Víctor Fresno, and Yuval Z. Feinstein. Integrating the Probabilistic Models BM25/BM25F into Lucene. *CoRR*, abs/0911.5046, 2009. Available from http://arxiv.org/abs/0911.5046.

Heri Ramampiaro. Retrieving BioMedical Information with BioTracer: Challenges and Possibilities. I *Proceedings of the 1st International Conference on Information Technology in Bio- and Medical Informatics (ITBAM 2010)*, Bilbao, Spain, August 2010. Springer Verlag (To Appear).

S.E. Robertson and K.S. Jones. Simple, proven approaches to text retrieval. Technical report, University of Cambridge, December 1994.

Stephen Robertson and Hugo Zaragoza. Tutorial 2D – The Probabilistic Relevance Model: BM25 and beyond, 2007. Available from http://www.zaragozas.info/hugo/academic/pdf/tutorial_sigir07_2d.pdf. [Cited May 25, 2010].

Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. I *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, New York, NY, USA, 2004. ACM. ISBN 1-58113-874-1. doi: 10.1145/1031171.1031181.

Tetsuya Sakai and Noriko Kando. A Further Note on Alternatives to Bpref. 2008. doi: 10.1.1.92.1340. Available from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.92.1340.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18 (11):613–620, 1975. ISSN 0001-0782. doi: 10.1145/361219.361220.

Kwangcheol Shin, Sang-Yong Han, Alexander Gelbukh, and Jaehwa Park. Advanced relevance feedback query expansion strategy for information retrieval in medline. *Progress in Pattern Recognition, Image Analysis and Applications*, pages 481–491, 2004. doi: 10.1007/978-3-540-30463-0_53. Available from http://dx.doi.org/10.1007/978-3-540-30463-0_53.

Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999. ISSN 0163-5840. doi: 10.1145/331403.331405.

Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4): 35–43, 2001.

Padmini Srinivasan. Retrieval Feedback in MEDLINE. *Journal of the American Medical Informatics Association*, 3(2):157–167, 1996a. doi: 10.1136/jamia.1996.96236284. Available from http://jamia.bmj.com/content/3/2/157.abstract.

Padmini Srinivasan. Query expansion and MEDLINE. *Information Processing & Management*, 32(4): 431–443, 1996b. ISSN 0306-4573. doi: 10.1016/0306-4573(95)00076-3. Available from http://www.sciencedirect.com/science/article/B6VC8-3VW1P4J-3/2/8b4a4890a57ca42a23f45e8ac19e7091.

The Apache Lucene project. Similarity (Lucene 3.0.1 API), 2009. Available from http://lucene.apache.org/java/3_0_1/api/core/org/apache/lucene/search/Similarity.html. [Cited February 26, 2010].

Ellen M. Voorhees. The Philosophy of Information Retrieval Evaluation. I *Evaluation of cross-language information retrieval systems: Second Workshop of the Cross-Language Evaluation Forum, CLEF 2001, Darmstadt, Germany, September 3-4, 2001: revised papers*, pages 355–370. Springer Verlag, 2002.

Yu-Chieh Wu, Teng-Kai Fan, Yue-Shi Lee, and Show-Jane Yen. Extracting Named Entities Using Support Vector Machines. *Knowledge Discovery in Life Science Literature*, pages 91–103, 2006. doi: 10.1007/11683568_8. Available from http://dx.doi.org/10.1007/11683568_8.

Emine Yilmaz and Javed A. Aslam. Estimating average precision with incomplete and imperfect judgments. I *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 102–111, New York, NY, USA, 2006. ACM. ISBN 1-59593-433-2. doi: 10.1145/1183614.1183633.

H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and S. Robertson. Microsoft Cambridge at TREC-13: Web and HARD tracks. I *Proceedings of TREC 2004*. Citeseer, 2004.

Appendix A

Custom queries

This chapter lists the 50 queries used in the prototype, corresponding to each of the topics in the TREC 2004 Genomics Track. The queries are a mix of queries acquired from Johannsson (2009) and queries automatically generated from the information needs of the track.

```
1  iron AND (Ferroportin1 OR (Ferroportin AND 1) OR SLC40A1 OR FPN1 OR HFE4 OR IREG1 OR (Iron AND regulated AND gene AND 1) OR MTP1 OR SLC11A3)
2  (transgenic OR transgenesis OR (copy AND gene)) AND (mice OR mouse OR murine)
3  (mouse kidney) (gene expression) (kidney development) (kidney)
4  kidney AND ((gene OR genes) OR (expression AND (profile OR profiles))) AND (mice OR mouse OR murine)
5  (isolate OR isolating OR fractionation OR purify) AND (cell OR (nucleus OR nuclei) OR subcellular)
6  FancD2 OR (Fanconi AND anemia) OR (group D2) OR (type AND 4 AND fanconi AND pancytopenia)
7  ((oxidative OR oxidation) AND stress) AND DNA AND repair
8  (oxidative OR oxidation) OR (cancer OR cancers OR carcinoma OR cancerous) AND (disease OR diseases OR carcinogenesis) AND (gene OR pathway) AND (DNA AND repair)
9  (muty OR hmyh) AND -myoglobin
10 (NEIL1)
11 hairless mice carcinogenesis skin OR UV
12 (gene OR genes) AND smad4
13 (TGFB OR (transforming growth factor beta) OR (TGF AND □)) AND (homeostasis OR angiogenesis)
14 (TGFB OR (transforming growth factor beta)) AND ((head and neck squamous cell) OR HNSCC)
15 (ATPase OR ATPases) AND (apoptosis OR (cell death))
16 (AAA proteins) (lipids) (AAA protein family) (protein interactions)
17 (DO1 OR (p53 AND (antibody OR anti))) AND binding
18 (Gis4 OR YML006C)
19 (GAL1 OR SUC1) AND (repressors OR reprosessor OR activators OR activator) AND SNF1
20 (covalent OR attachment OR covalence OR substrate) AND (ubiquitin OR ubiquitously OR ubiquitylation OR ubiquitination)
21 (p63 OR TP63) OR (TP73 OR p73) ((cell cycle arrest) OR apoptosis) DNA
22 p53 AND DNA AND (respond OR responding OR response) AND (break OR damage OR ((single OR double) AND stranded))
23 Saccharomyces OR cerevisiae (protein OR proteins) (ubiquitin OR proteolytic OR pathway)
24 (PGRP) (mouse AND peptidoglycan AND recognition AND proteins)
25 (scleroderma OR (autoimmune AND disease)) AND ((genes OR gene OR genome) OR (scan OR scans) OR (microarray OR (micro AND array)))
26 (BFA1) (BUB2)
```

```
(autophagy OR (gene autophagic)) AND apoptosis
(autophagy OR (gene autophagic)) AND apoptosis AND (proteases OR morphological)
(gyrA OR (DNA gyrase)) AND ((phenotype OR phenotypes) OR (sequence OR sequences)) AND (mutation OR mutations OR alteration) AND ((E AND coli) OR escherichia)
Nkx OR Sax
(TOR OR mTOR OR (Target AND Of AND Rapamycin) OR FRAP1 OR (FK506 AND associated AND protein))
Xenograft AND (tumorogenesis OR cancer OR cancers OR carcinoma)
(histoplasmosis OR (histoplasma AND capsulatum) OR (blood borne pathogen)) AND (mice OR mouse OR murine)
Cryptococcus AND (gene OR genes OR genome)
(histoplasmosis OR (histoplasma AND capsulatum) OR (blood borne pathogen)) AND (mice OR mouse OR murine)
Cryptococcus AND (gene OR genes OR genome)
PAM OR (peptide AND amidating AND enzyme) OR (peptidylglycine AND amidating)
(stroke OR (cerebrovascular AND accident) OR CVA) AND ((genetic AND (loci OR locus)) OR E4 OR (factor AND V) OR (risk AND (factor OR factors)))
(hypertension OR HTN OR (high AND blood AND pressure)) AND ((risk OR danger) AND (genes or gene))
(antigen OR antigens) AND (epithelial OR epithelium) (lung OR pulmonary OR lungs)
(mutation OR mutations OR altered) AND ((Cystic AND Fibrosis) OR CF OR mucovoidosis OR muscoviscidosis)
((chromosome OR chromosomal) AND (translocations OR translocation)) OR ((chromosome OR chromosomal) AND (rearrangement OR rearrangements))
(sleeping AND beauty) OR (Kleine AND Levin AND Syndrome) OR KLS
((nerve AND growth AND factor) OR NGF) AND (protein OR proteins)
(MWH1 OR (mental health wellness)) OR (mental (disorder OR disorders) (gene OR genes OR genetic))
RSK2 OR (ribosomal protein kinase)
(BCL OR BCL2 OR (BCL AND 2)) AND ((antagonists OR antagonist) OR (inhibitors OR inhibitor))
(UNC OR (homologues OR homolog) OR BGS) AND ((gene OR genes) OR (c AND elegans) OR (Caenorhabditis AND elegans))
(glyphosate OR glycine) AND (tolerance OR tolerant OR immune)
(temperature OR cold) AND protein AND ((E AND coli) OR escherichia)
```

Appendix B

Example MEDLINE citation in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MedlineCitationSet PUBLIC "-//NLM//DTD Medline Citation, 1st January, 2010//EN" "http://www.nlm.nih.gov/databases/dtd/nlmmedlinecitationset_100101.dtd">
<MedlineCitationSet>
    <MedlineCitation Owner="NLM" Status="MEDLINE">
        <PMID>10540283</PMID>
        <DateCreated>
            <Year>1999</Year>
            <Month>12</Month>
            <Day>17</Day>
        </DateCreated>
        <DateCompleted>
            <Year>1999</Year>
            <Month>12</Month>
            <Day>17</Day>
        </DateCompleted>
        <DateRevised>
            <Year>2006</Year>
            <Month>11</Month>
            <Day>15</Day>
        </DateRevised>
        <Article PubModel="Print">
            <Journal>
                <ISSN IssnType="Print">0950-382X</ISSN>
                <JournalIssue CitedMedium="Print">
                    <Volume>34</Volume>
                    <Issue>1</Issue>
                    <PubDate>
                        <Year>1999</Year>
                        <Month>Oct</Month>
                    </PubDate>
                </JournalIssue>
                <Title>Molecular microbiology</Title>
                <ISOAbbreviation>Mol. Microbiol.</ISOAbbreviation>
            </Journal>
            <ArticleTitle>Transcription regulation of the nir gene cluster encoding nitrite reductase of Paracoccus denitrificans involves NNR and NirI, a novel type of membrane protein.</ArticleTitle>
            <Pagination>
                <MedlinePgn>24-36</MedlinePgn>
```

```
        </Pagination>
        <Abstract>
            <AbstractText>The nirIX gene cluster of Paracoccus denitrificans is
                →located between the nir and nor gene clusters encoding nitrite
                →and nitric oxide reductases respectively. The NirI sequence
                →corresponds to that of a membrane-bound protein with six
                →transmembrane helices, a large periplasmic domain and cysteine-
                →rich cytoplasmic domains that resemble the binding sites of [4Fe
                →-4S] clusters in many ferredoxin-like proteins. NirX is soluble
                →and apparently located in the periplasm, as judged by the
                →predicted signal sequence. NirI and NirX are homologues of NosR
                →and NosX, proteins involved in regulation of the expression of
                →the nos gene cluster encoding nitrous oxide reductase in
                →Pseudomonas stutzeri and Sinorhizobium meliloti. Analysis of a
                →NirI-deficient mutant strain revealed that NirI is involved in
                →transcription activation of the nir gene cluster in response to
                →oxygen limitation and the presence of N-oxides. The NirX-
                →deficient mutant transiently accumulated nitrite in the growth
                →medium, but it had a final growth yield similar to that of the
                →wild type. Transcription of the nirIX gene cluster itself was
                →controlled by NNR, a member of the family of FNR-like
                →transcriptional activators. An NNR binding sequence is located in
                → the middle of the intergenic region between the nirI and nirS
                →genes with its centre located at position -41.5 relative to the
                →transcription start sites of both genes. Attempts to complement
                →the NirI mutation via cloning of the nirIX gene cluster on a
                →broad-host-range vector were unsuccessful, the ability to express
                → nitrite reductase being restored only when the nirIX gene
                →cluster was reintegrated into the chromosome of the NirI-
                →deficient mutant via homologous recombination in such a way that
                →the wild-type nirI gene was present directly upstream of the nir
                →operon.</AbstractText>
        </Abstract>
        <Affiliation>Department of Molecular Cell Physiology, Faculty of Biology,
            →BioCentrum Amsterdam, Vrije Universiteit, De Boelelaan 1087, NL-1081
            →HV Amsterdam, The Netherlands.</Affiliation>
        <AuthorList CompleteYN="Y">
            <Author ValidYN="Y">
                <LastName>Saunders</LastName>
                <ForeName>N F</ForeName>
                <Initials>NF</Initials>
            </Author>
            <Author ValidYN="Y">
                <LastName>Houben</LastName>
                <ForeName>E N</ForeName>
                <Initials>EN</Initials>
            </Author>
            <Author ValidYN="Y">
                <LastName>Koefoed</LastName>
                <ForeName>S</ForeName>
                <Initials>S</Initials>
            </Author>
            <Author ValidYN="Y">
                <LastName>de Weert</LastName>
                <ForeName>S</ForeName>
```

```
                    <Initials>S</Initials>
                </Author>
                <Author ValidYN="Y">
                    <LastName>Reijnders</LastName>
                    <ForeName>W N</ForeName>
                    <Initials>WN</Initials>
                </Author>
                <Author ValidYN="Y">
                    <LastName>Westerhoff</LastName>
                    <ForeName>H V</ForeName>
                    <Initials>HV</Initials>
                </Author>
                <Author ValidYN="Y">
                    <LastName>De Boer</LastName>
                    <ForeName>A P</ForeName>
                    <Initials>AP</Initials>
                </Author>
                <Author ValidYN="Y">
                    <LastName>Van Spanning</LastName>
                    <ForeName>R J</ForeName>
                    <Initials>RJ</Initials>
                </Author>
            </AuthorList>
            <Language>eng</Language>
            <DataBankList CompleteYN="Y">
                <DataBank>
                    <DataBankName>GENBANK</DataBankName>
                    <AccessionNumberList>
                        <AccessionNumber>AF005358</AccessionNumber>
                        <AccessionNumber>U47133</AccessionNumber>
                        <AccessionNumber>U94899</AccessionNumber>
                    </AccessionNumberList>
                </DataBank>
                <DataBank>
                    <DataBankName>PDB</DataBankName>
                    <AccessionNumberList>
                        <AccessionNumber>P33943</AccessionNumber>
                    </AccessionNumberList>
                </DataBank>
            </DataBankList>
            <PublicationTypeList>
                <PublicationType>Journal Article</PublicationType>
                <PublicationType>Research Support, Non-U.S. Government</
                    → PublicationType>
            </PublicationTypeList>
        </Article>
        <MedlineJournalInfo>
            <Country>ENGLAND</Country>
            <MedlineTA>Mol Microbiol</MedlineTA>
            <NlmUniqueID>8712028</NlmUniqueID>
            <ISSNLinking>0950-382X</ISSNLinking>
        </MedlineJournalInfo>
        <ChemicalList>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
```

```
                <NameOfSubstance>Bacterial Proteins</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>DNA-Binding Proteins</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>Membrane Proteins</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>NNR protein, Paracoccus denitrificans</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>NirI protein, Paracoccus denitrificans</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>NirX protein, Paracoccus denitrificans</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>0</RegistryNumber>
                <NameOfSubstance>Transcription Factors</NameOfSubstance>
            </Chemical>
            <Chemical>
                <RegistryNumber>EC 1.7.-</RegistryNumber>
                <NameOfSubstance>Nitrite Reductases</NameOfSubstance>
            </Chemical>
        </ChemicalList>
        <CitationSubset>IM</CitationSubset>
        <MeshHeadingList>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Amino Acid Sequence</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="Y">Bacterial Proteins</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Base Sequence</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="Y">DNA-Binding Proteins</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Gene Expression Regulation, Bacterial</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Genetic Complementation Test</DescriptorName>
```

```
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Membrane Proteins</DescriptorName>
                <QualifierName MajorTopicYN="N">chemistry</QualifierName>
                <QualifierName MajorTopicYN="Y">genetics</QualifierName>
                <QualifierName MajorTopicYN="N">metabolism</QualifierName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Molecular Sequence Data</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Multigene Family</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Mutation</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Nitrite Reductases</DescriptorName>
                <QualifierName MajorTopicYN="Y">genetics</QualifierName>
                <QualifierName MajorTopicYN="N">metabolism</QualifierName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Paracoccus denitrificans</DescriptorName>
                <QualifierName MajorTopicYN="Y">genetics</QualifierName>
                <QualifierName MajorTopicYN="N">metabolism</QualifierName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Protein Structure, Secondary</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Sequence Homology, Amino Acid</DescriptorName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="N">Transcription Factors</DescriptorName>
                <QualifierName MajorTopicYN="Y">genetics</QualifierName>
                <QualifierName MajorTopicYN="N">metabolism</QualifierName>
            </MeshHeading>
            <MeshHeading>
                <DescriptorName MajorTopicYN="Y">Transcription, Genetic</DescriptorName>
            </MeshHeading>
        </MeshHeadingList>
    </MedlineCitation>
</MedlineCitationSet>
```

Appendix C

Example topic from the TREC 2004 Genomics Track

```
<?xml version="1.0" encoding="UTF-8"?>
<TOPICS>
    <TOPIC>
        <ID>8</ID>
        <TITLE>Correlation between DNA repair pathways and skin cancer</TITLE>
        <NEED>Genes and proteins (pathways) common to DNA repair, oxidative diseases, skin-carcinogenesis, and UV-carcinogenesis.</NEED>
        <CONTEXT>
            Are there genes and mechanisms that are utilized by more than one of these fields? A relevant article mentions a gene or pathway, DNA repair, and one or more oxidative or cancerous diseases.
        </CONTEXT>
    </TOPIC>
</TOPICS>
```