Expanding Queries Using Multiple Resources

The AID Group at TREC 2006: Genomics Track

Edgar Meij[1] **Machiel Jansen**[2,*] **Maarten de Rijke**[1]

[1] ISLA, University of Amsterdam
emeij, mdr@science.uva.nl
[2] Free University Amsterdam
mgjansen@few.vu.nl

Abstract: We describe our participation in the TREC 2006 Genomics track, in which our main focus was on query expansion. We hypothesized that applying query expansion techniques would help us both to identify and retrieve synonymous terms, and to cope with ambiguity. To this end, we developed several collection-specific as well as web-based strategies. We also performed post-submission experiments, in which we compare various retrieval engines, such as Lucene, Indri, and Lemur, using a simple baseline topicset. When indexing entire paragraphs as pseudo-documents, we find that Lemur is able to achieve the highest document-, passage-, and aspect-level scores, using the KL-divergence method and its default settings. Additionally, we index the collection at a lower level of granularity, by creating pseudo-documents comprising of individual sentences. When we search these instead of paragraphs in Lucene, the passage-level scores improve considerably. Finally we note that stemming improves overall scores by at least 10%.

1 Introduction

In this paper we describe our participation in the TREC 2006 Genomics track. One of our working hypotheses is that finding synonymous terms while, at the same time, coping with ambiguous terms is essential. We propose several methods, collection-specific as well as web-based, to identify synonymous terms. To counter any possible query drift, we impose additional constraints on the expansion terms we find.

Additionally, we include the results of post-submission experiments, in which we have evaluated different passage lengths, stemming, and different retrieval engines.

The remainder of this paper is organized as follows. In Section 2 we describe our collection preprocessing and passage identification. Next, in Section 3 we introduce the retrieval models which we evaluated during this year's edition of TREC Genomics. We then elaborate on our proposed query expansion techniques in Section 4, and follow with a description of our submitted runs in Section 5. The results can be found in Section 6, and we summarize our findings in a concluding section.

2 Experimental Setup

In this section we elaborate on the particular tools and methods used for indexing and retrieving. We describe the way we handle passage indentification as well as the way we preprocess the collection.

2.1 Collection Preprocessing

The 2006 Genomics document collection consists of 162,259 full-text biomedical articles, which were preprocessed as follows:

1. replace HTML entities with their ISO-Latin1 counterparts,
2. remove HTML tags,
3. remove top-level tables; these only serve navigational purposes,
4. remove citations within text,
5. lowercase terms, and
6. remove stopwords.

The used stopword list includes common task-specific terms such as “figure,” “table,” “view,” “larger,” “version,” “interactions,” “role,” and “contribute.”

[*] Currently at Collexis, e-mail: jansen@collexis.com

2.2 Passage Identification

This year's Genomics track introduced a novel task. Unlike previous years, participants were requested to return relevant passages instead of entire documents and the systems are judged based on 3 levels of granularity: returned documents, passages, and aspects. We have chosen to mainly focus on documents and passages for our participation.

We experimented with various ways of identifying passages, and for our submissions we decided to consider every sentence as being a passage—which we identify using Lingpipe's sentence extractor [1]. Every sentence gets indexed as a separate (pseudo-)document and we include positional information and the originating PubMed ID in the index. The intuition behind this approach is to yield relatively high precision, since query terms should appear within the same sentence.

The maximum length of any retrieved passage was delimited by paragraph tags in the original HTML files. As a comparison, we also indexed every paragraph as a separate pseudo-document.

3 Retrieval Models

This section describes the models employed by the evaluated retrieval engines. All of our initially submitted runs were created using Lucene [13] with a multinomial language modeling extension that we developed in-house [8]. As post-submission experiments we include the results of using the default Lucene retrieval model, as well as Lemur and Indri [12].

3.1 Lucene – Language Modeling

We estimate a language model for each pseudo-document and for any given query we rank the documents with respect to the likelihood that the document language model generated the query:

$$P(d|q) \propto P(d) \cdot \prod_{t \in q} P(t|d), \tag{1}$$

where d is a document and t is a term in query q. In the implemented scoring formula the probabilities are reduced to rank-equivalent logs of probabilities. To account for data sparseness, we interpolate the likelihood $P(t|d)$ using Jelinek-Mercer smoothing [5, 18, 19]. This can be viewed as estimating the probability

$$P(d|q) = P(d) \cdot \prod_{t \in q} \left((1-\lambda) \cdot P(t|D) + \lambda \cdot P(t|d)\right), \tag{2}$$

where D is the collection. We need to estimate three probabilities: the prior probability of the document, $P(d)$; the probability of observing a term in a document, $P(t|d)$; and the probability of observing the term in the collection, $P(t|D)$. We assume the query terms to be independent, and use a linear interpolation of a document model and a collection model to estimate the probability of a query term.

The probabilities are estimated using maximum likelihood estimates:

$$P(t|d) = \frac{tf(t,d)}{|d|}, \tag{3}$$

$$P(t|D) = \frac{df(t,D)}{\sum_{t' \in D} df(t',D)}, \tag{4}$$

$$P(d) = \frac{|d|}{\sum_{d' \in D} |d'|}, \tag{5}$$

where $tf(t,d)$ is the termfrequency of term t in document d; $df(t)$ is the count of documents in which term t occurs, and $|d|$ denotes the length of a document d [4].

The risk with parameter estimation using maximum likelihood estimates in Equation 2, is the underestimation of unseen or rare terms and overestimation of frequently occurring ones. Especially when dealing with very short documents (such as sentences) this bias becomes clearly visible. To compensate, we smooth our language model in Equation 2 using collection frequencies rather than document frequencies [18]. The parameter λ is the smoothing parameter, which can be optimized using training data. Since this is the first year in which the current collection is being used, such training data is unavailable. We therefore use the standard value of 0.15, as described by Hiemstra and Kraaij [6].

3.2 Lemur

We also include Lemur's KL-divergence language model based retrieval method in our evaluations. Within that method, documents are ranked according to the negative of the divergence of the query language model from the document language model. We select interpolated Jelinek-Mercer smoothing and set document λ to 0.15.

3.3 Indri

The default retrieval method in Indri is based on the combination of inference networks with language modeling, using Dirichlet smoothing [9]. Thus, documents are ranked according to:

$$P(t|d) = \frac{tf(t,d) + \mu P(t|D)}{|d| + \mu}, \tag{6}$$

where μ is a free parameter to determine the amount of smoothing. We only use the #combine operator and set μ to its default value of 1500.

3.4 Lucene

We also include results of Lucene's off-the-shelf retrieval model, i.e., for a collection D, document d and query q:

$$sim(q,d) = \sum_{t \in q} \frac{tf_{t,q} \cdot idf_t}{norm_q} \cdot \frac{tf_{t,d} \cdot idf_t}{norm_d} \cdot coord_{q,d} \cdot weight_t,$$

where

$$\begin{aligned} tf_{t,X} &= \sqrt{\mathrm{freq}(t,X)}, \\ idf_t &= 1 + \log \frac{|D|}{\mathrm{freq}(t,D)}, \\ norm_q &= \sqrt{\sum_{t \in q} tf_{t,q} \cdot idf_t{}^2}, \\ norm_d &= \sqrt{|d|}, \\ coord_{q,d} &= \frac{|q \cap d|}{|q|}. \end{aligned}$$

4 Query Expansion

The fact that there are frequently occuring spelling variations and synonyms for any given biomedical concept, degrades the performance of regular adhoc IR techniques. To overcome this problem, we propose different forms of collection-specific and online query expansion methods, based on the hypothesis that proper handling of synonymous terms is essential in biomedical text retrieval. The methods we propose include using acronyms and their corresponding long forms from the collection, the matching of related long forms, and the online lookup of unknown query terms and gene names.

As is common with using query expansion in general, one is likely to improve recall at the cost of precision [10, 17]. Some of the added synonyms, acronyms, or long forms for a particular query term might be identical to other biomedical concepts (e.g., diseases or methods, where the query term is a gene name). Including all possible expansions in the query will therefore result in higher recall but also in more noise. Our intuition is that the high-precision approach to passage identification, by indexing individual sentences as pseudo-documents as described in Section 2.2, would compensate for any query drift.

4.1 Query Preprocessing

We use the Genia parser [3, 11] to syntactically parse the topics and extract all noun phrases (NPs) and headwords, thus identifying all relevant *aspects* from the query [2]. All topics follow a certain topic template, so each contains one or more biological concepts and processes and some explicit relationship between them. We identify the biological *subject(s)* and *object(s)* of the query and discard the relationship term(s).

The resulting query aspects are kept as phrases for subsequent query expansion, since phrases are reported to improve retrieval results when compared to single-word indexing [14, 15]. We believe this is also the case in biomedical IR. Finally, we apply the breakpoint algorithm, as introduced by Huang et al. [7], to the query aspects. In the following sections we elaborate on our query expansion strategies.

4.2 Corpus-Specific Acronym Identification

We mine acronyms and their corresponding forms directly from the documents in the collection, using the algorithm described by Schwartz and Hearst [16]. We adapted their approach in order to also collect frequency information. All found acronyms, long forms and frequencies were stored in a database, with an acronym being defined as a term with a maximum length of 6 characters and containing at least one uppercase character. For every query aspect, we check whether it is an acronym, and proceed with different approaches, depending on whether the term is indeed an acronym.

4.3 Acronyms

If the term is indeed an acronym, we look up all possible long forms in the database and add all results with a frequency of more than 1 to the query.

In addition, we also look up alternative acronyms for a given acronym. These are identified as follows: A list was made for all long forms of every acronym. The most frequent long form with a *different* acronym is identified and the acronym is selected for addition to the query. For example, the most commonly used long form for the term PrnP is "prion protein gene." The most commonly used acronym for this long form is not PrnP, but prp. We hesitated to put the alternative longer form in the query as well, but chose not to do so.

4.4 Long Forms

If the NP is not an acronym, we check whether it *has* one or more acronyms. Again, all resulting acronyms with a frequency greater than 1 are added to the query.

Acronyms related to a given long form are also searched for in the database. For all long forms we check if they occur as a substring in other long forms. If so, these *related* long forms, together with their most frequent acronym, were returned. As an example "Alzheimer's disease" produces "fad" and "Familiar Alzheimer's disease" as related acronym and long form respectively.

Finally, for long forms which don't have a long form in the database, we turn to Google. These long forms were submitted to the search engine, prefixed with the *define* operator, and acronyms that occurred more than once in the snippets

Results.

Run	Model	Stemmed	Pseudo-document	MAP		
				Document	Passage	Aspect
`baseline`	Indri	yes	paragraph	0.29	0.017	0.20
`baseline`	Indri	no	paragraph	0.26	0.012	0.16
`baseline`	Lemur	yes	paragraph	**0.36**	0.031	**0.22**
`baseline`	Lemur	no	paragraph	0.31	0.022	0.18
`baseline`	Lucene	no	paragraph	0.26	0.027	0.11
`baseline`	Lucene	no	sentence	0.25	**0.048**	0.058
`baseline`	Lucene-LM	no	paragraph	0.30	0.025	0.17
`baseline`	Lucene-LM	no	sentence	0.30	0.047	0.093
`UAmsBaseLine`	Lucene-LM	no	paragraph	0.27	0.017	0.097
`UAmsBaseLine`	Lucene-LM	no	sentence	0.25	0.033	0.060
`UAmsExp`	Lucene-LM	no	paragraph	0.26	0.020	0.087
`UAmsExp`	Lucene-LM	no	sentence	0.24	0.042	0.069

Table 1: Results of the evaluated runs we evaluated (best scores in boldface.)

returned were used as query expansion terms, together with their long forms. This helped us, for example, to find the acronym BSE for "mad cow disease" and PCD for "apoptosis."

4.5 Breakpoints

We also applied the earlier mentioned breakpoint algorithm [7] to the found expansion terms. To limit needlessly long queries, we filter the generated morphological variants based on the terms in the index—only if a particular variant occurs in the collection does it get added to the query. For example: "Prn-P" "Prn P" "Pr P" "Pr-P" are breakpoint alternatives for "PrnP" but none are selected for addition because none of these occur in the collection.

4.6 Gene Name Expansion

For Gene names we propose a different algorithm. A gene name is defined as a long form or an NP that ends in "gene[s]". It turns out that all identified acronyms that don't have an entry in the database are indeed gene names. We look these up on BioInformatics.org[1] and again mine the output for (synonymous) acronyms. All those that start with a bracket or a digit are discarded; the resulting ones are added to the query.

5 Runs

To get some idea of the relative performance of various retrieval models, we evaluate a baseline run using Lucene, Lemur, Indri, and Lucene with a language modeling extension. Due to time constraints however, we could not evaluate the query expansion methods from Section 4 with every retrieval model. These were only evaluated using Lucene with the language modeling extension, as described in Section 3.1. Also, we were unable to index sentences as pseudo-documents with Indri/Lemur. We did apply Porter's stemming algorithm with Indri/Lemur.

Using the above mentioned models and indexes, we evaluate the following runs:

`baseline` A baseline run, using only the extracted noun phrases from the original topics.

`UAmsBaseLine` The same as `baseline`, with breakpoint variant generation, collection-specific acronym, and long-form expansion.

`UAmsExp` Same as `UAmsBaseLine`, with additional query expansion: substring matching, gene name expansion, alternative acronyms, and Google define.

6 Results

Table 1 displays the results of all evaluated runs (best scores per metric in boldface). Looking at the baseline runs, we first note that stemming improves document retrieval effectiveness by more than 10%. The KL-divergence method, as implemented in Lemur, is able to outperform any other retrieval method on the document-level, using mostly default settings. Lucene, when used out-of-the-box, has the lowest performance scores when taking only document-level assessments into account. It is however able to achieve the highest passage-level scores, closely followed by Lucene with the language modeling extension.

It is too early to dismiss our intuition that the high-precision approach to passage identification will improve average precision scores. With the current results, it seems that indexing sentences instead of entire paragraphs as pseudo-documents boosts passage-level scores, while at the same time keeping document-level scores at a reasonable level. Expanding the queries using the proposed algorithms has

[1] BioInformatics http://bioinformatics.org/textknowledge/synonym.php

mixed effects on retrieval effectiveness. They do certainly not succeed in outperforming the already strong baseline.

When zooming in on the individual document scores, there are three distinct peaks for topics 160, 163, and 181. These topics contain “mad cow disease,” “Alzheimer’s disease,” and “colon cancer,” respectively. As indicated earlier in Section 4.4, the use of a generic web search engine helps us to find the proper acronym for “mad cow disease,” and substring matching helps us to find the more common long form for “Alzheimer’s disease,” namely “familial Alzheimer’s disease,” together with its acronym FAD.

We did not pursue any specific goal regarding aspect retrieval. However, the aspect scores indicate that stemming helps, whereas indexing documents at the sentence level certainly does not.

7 Conclusions

We have described our participation in the TREC 2006 Genomics track. Our aim this year was to apply query expansion techniques in order to find synonymous terms. First off, we used a POS tagger to identify noun phrases (or *aspects*) in the topics. Next, the identified query aspects were expanded using various query expansion strategies, based on collection-specific as well as online algorithms. The application of these methods however, degraded the strong baseline. We do still believe that proper handling of synonymous terms is essential and might further improve the retrieval scores and we plan to experiment with weighing various expansion terms to smooth the effect of adding new terms to the original query.

We used sentence boundaries within the original source documents as our passage boundaries, under the assumption that this would yield results with relatively high precision scores. While this certainly seems to boost passage-level effectiveness, there is room for improvement, e.g., by merging high-scoring consecutive sentences. Future experiments will also include building and evaluating a sentence-level index in Indri/Lemur. Finally, tuning various parameters might further improve the highest scores.

Acknowledgements

We would like to thank Willem van Hage and Sophia Katrenko for their contributions and many insightful discussions. This work was carried out in the context of the Virtual Laboratory for e-Science project (http://www.vl-e.nl). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ). Maarten de Rijke was supported by the Netherlands Organization for Scientific Research (NWO) under project number 220-80-001.

8 References

[1] Alias–i. Lingpipe - a suite of java libraries for the linguistic analysis of human language., 2005. http://www.alias-i.com/lingpipe/.

[2] C. Buckley. Why current IR engines fail. In *SIGIR ’04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 584–585, New York, NY, USA, 2004. ACM Press.

[3] A. Clegg and A. Shepherd. Evaluating and integrating treebank parsers on a biomedical corpus. In *Proceedings of the Association for Computational Linguistics Workshop on Software 2005*, 2005.

[4] W. B. Croft and J. Lafferty. *Language Modeling for Information Retrieval*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[5] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, 2001.

[6] D. Hiemstra and W. Kraaij. Twenty-One at TREC7: Ad-hoc and Cross-Language Track. In *TREC*, pages 174–185, 1998.

[7] X. Huang, Z. Ming, and L. Si. York University at TREC 2005: Genomics track. In *Proceedings of the 14th Text Retrieval Conference*, 2005.

[8] ILPS. The ILPS extension of the Lucene search engine, 2005. http://ilps.science.uva.nl/Resources/.

[9] Indri. Indri retrieval model, 2007. http://ciir.cs.umass.edu/˜metzler/indriretmodel.html.

[10] J. Kekäläinen and K. Järvelin. The impact of query structure and query expansion on retrieval performance. In W. B. Croft, A. Moffat, C. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st Annual International ACM SIGIRConference on Research and Development inInformation Retrieval*, pages 130–137, Melbourne, Australia, Aug. 1998. ACM Press, New York.

[11] J. D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. Genia corpus–semantically annotated corpus for bio-textmining. In *Bioinformatics*, volume 19 Suppl 1, pages 180–182, CREST, Japan Science and Technology Corporation, Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan., 2003. URL http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/.

[12] Lemur. The Lemur toolkit, 2007. http://www.lemurproject.org.

[13] Lucene. The Lucene search engine, 2005. http://jakarta.apache.org/lucene/.

[14] G. Mishne and M. de Rijke. Boosting web retrieval through query operations. In D. E. Losada and J. M. Fernández-Luna, editors, *ECIR*, volume 3408 of *Lecture Notes in Computer Science*, pages 502–516. Springer, 2005.

[15] J. Pickens and W. B. Croft. An exploratory analysis of phrases in text retrieval. In *Proceedings of RIAO (Recherche d’Information assiste par Ordinateur)*, pages 1179–1195, 2000.

[16] A. Schwartz and M. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical texts. In *Proceedings of the Pacific Symposium on Biocomputing (PSB 2003)*, 2003. URL citeseer.ist.psu.edu/schwartz03simple.html.

[17] E. M. Voorhees. Query expansion using lexical-semantic rela-

tions. In W. B. Croft and C. J. van Rijsbergen, editors, *SIGIR*, pages 61–69. ACM/Springer, 1994.

[18] H. Zaragoza, D. Hiemstra, and M. Tipping. Bayesian extension to the language model for ad hoc information retrieval. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 4–9, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-646-3.

[19] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-331-6.