Exploiting Morphological Query Structure Using Genetic Optimisation *

Jose R. Pérez-Agüera, Hugo Zaragoza, Lourdes Araujo

Dpto de Ingeniería del Software e Inteligencia Artificial, UCM. jose.aguera@fdi.ucm.es
Yahoo! Research Barcelona. hugoz@yahoo-inc.com
Dpto. de Lenguajes y Sistemas Informáticos, UNED. lurdes@lsi.uned.es

Abstract. In this paper we deal with two issues. First, we discuss the negative effects of term correlation in query expansion algorithms, and we propose a novel and simple method (query clauses) to represent expanded queries which may alleviate some of these negative effects. Second, we discuss a method to optimise local query expansion methods using genetic algorithms, and we apply this method to improve stemming. We evaluate this method with the novel query representation method and show very significant improvements for the problem of optimising stemming.

1 Introduction

There is an underlying common background in many of the works done in query reformulation, namely the appropriate selection of a subset of search terms among a list of candidate terms. The number of possible subsets grows exponentially with the size of the candidate set. Furthermore, we cannot evaluate a priori the quality of a subset with respect to another one: this depends on the (unknown) relevance of the documents in the collection. For these reasons, standard optimisation techniques cannot be applied to this problem. Instead, we must resort to heuristic optimisation algorithms, such as genetic algorithms, that sample the space of possible subsets and predicts their quality in some unsupervised manner.

Before considering any query reformulation process is very important to take into account that modern Information Retrieval ranking functions apply the *term independence* assumption. This assumption takes on many forms, but loosely it implies that the *effect of each query term on document relevance can be evaluated independently of the other query terms*. This has the effect of rendering all queries *flat*, whithout structure.

However, there are many cases in which queries have some known *linguistic structure*, such as degree of synonymy between terms, term cooccurrence or correlation information with respect to the query or to specific query terms, etc. This is typical of queries constructed by a query-expansion method, of stemming

* Supported by projects TIN2007-67581-C02-01 and TIN2007-68083-C02-01

or normalizing terms, of taking into account multi-terms or phrases, etc. Surprisingly, almost all ranking functions (and experiments) ignore this structural information: after expansion, selection and re-weighting of terms, a flat query (a set of weighted terms) is given to the ranking function which *assumes terms are independent* and scores documents accordingly.

Specifically, we want to investigate two issues: the selection of adequate terms for query reformulation and term independence assumption, to propose a new method that solve the classical problems associated to these issues. The morphological query structure of the query has been chosen to show how our approach is capable to improve state-of-art approaches like Porter's stemmer.

In Section 2 we propose a novel way to represent expanded queries that encodes information about the term correlation using clauses like set of related terms. The proposed representations greatly increase the expressivity power of queries, but at the expense of introducing parameters (weights) which may be hard to set. Section 3 shows one experiment where our clauses representation model is adapted to the problem of morphological query expansion. In section 4 we apply it to the problem of optimising the expansion of a term with respect to its stem. We show that we can significantly improve the performance of Porter stemming by adapting the expansion to every query. Section 5 draw the main conclusions and describe the future lines of work.

2 Ranking independent clauses of dependent terms

One of the reasons of the high performance of modern ad-hoc retrieval systems is their use of document term frequency. It is well known[12] that i) probability of relevance of a document increases as the term frequency of a query term increases, and ii) this increase is non-linear. For this reason most modern ranking functions use an increasing saturating function to weight document terms that are in the query. An example of this is the term saturating function used as part as BM25[12] :

$$w(d,t) := \frac{tf(d,t)}{tf(d,t) + K1} \tag{1}$$

where $tf(d,t)$ is the term frequency of term t in document d, and $K1$ is a constant. Similar nonlinear term frequency functions are found in most IR ranking models such modern variants of the vector space model, the language model, divergence from randomness models, etc. Besides, all these ranking functions assume that *the relevance information of different query terms is independent* and therefore the relevance information gained by seeing query terms can be computed separately and added linearly (or log-linearly), for example, in BM25:

$$score(d) := \sum_{t \in q} w(d,t) \cdot idf(t) \tag{2}$$

This independence assumption is usually reasonable for short queries (i.e. *"Italian restaurant in Cambridge"*), since users use each term to represent a different aspect of the query. However, such assumption breaks down for queries

that are sufficiently complex to contain terms with sufficiently close meaning. Consider for example the query *"Italian restaurant cafeteria bistro Barcelona"*. Having seen the term *restaurant* twice in a document, which term is more informative: *Barcelona* or *cafeteria*? Loosely speaking, if a group of terms carries the same meaning, the amount of relevance information gained by their presence should diminish as we see other terms in this group, very much like in equation (1) does for term frequency, and unlike (2).

This situation arises very often in modern IR tasks and systems, in particular in the following areas:

- morphological expansion (e.g. stemming, spelling, abbreviations, capitalization),
- extracting multi-terms from the query
- query term expansion (e.g. user feedback, co-occurrence based expansion),
- lexical semantic expansion (e.g. using WordNet),
- using taxonomies and ontologies to improve search,
- user modeling, personalization,
- query disambiguation (where terms are added to clarify the correct semantic context),
- finding similar documents (where the query is an entire document),
- document classification (where the query is a set of documents),
- structured queries (such as TREC structured topics).

We propose to consider two *levels* of representation: *terms* and *term clauses.* Clauses are *sets of weighted terms* that are intended to represent a particular aspect of the query. The weights w represent their relative importance within the clause (in particular, the strength of the dependence with relevance). Thus, a query can be thought of as a bag of bags of (weighted) terms:

$$c := \left\{ (t_0, w_0), (t_1, w_1), ..., \left(t_{|c|}, w_{|c|}\right) \right\}$$
$$q := \left\{ c_1, c_2, ..., c_{|q|} \right\}$$

Boolean retrieval models and the Inquery[3] retrieval model have used query representations even more general than this. Here we restrict ourselves to this representation with two levels to give clear semantics to each level: term and clause. We are going to consider terms *within* a clause as if they were greatly dependent with respect to relevance; in fact we will consider them as if they were virtually the same term. Second, we consider terms *across* clauses as being independent with respect relevance, as is usually done across terms.

Conceptually, what we propose is a projection from the space of terms to the space of clauses. Formally, we represent a document as the vector $d = (tf_1, ...tf_i, ..., tf_V)$ where V if the size of the vocabulary. We represent a query having n clauses as a nxV matrix of weights: $C = (c_{ij})$ where $c_{ij}$is the weight of jth term in ith clause. The projected document is then $d|_C := d \times C^{\mathsf{T}}$.

Consider this example. Imagine that we are given a corpus with four terms A-D:

Doc	A	B	C	D
d_1	2	1	0	1
d_2	0	1	1	1
d_3	1	2	0	2

Now consider the query with two clauses:

$$q := \{ \{ (A, 1.0) , (B, 0.7) \} , \{ (C, 1.0) \} \}$$

This query can also be represented by the matrix:

$$C := \begin{bmatrix} 1 & .7 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

.

In the projection $d|_C := d \times C^{\mathsf{T}}$ the original query terms are removed from the collection and replaced with new pseudo-terms representing the clauses; other terms are removed because all terms are included in the clauses. The term frequency of the clause pseudo-term will be equal to the weighted sum of term frequencies of the terms in the clause. In our example:

Transformed Doc	Clause1	Clause2
$d_1\|_q$	2.7	0
$d_2\|_q$	0.7	1
$d_3\|_q$	2.4	0

This projection will be different for every clause, so it cannot be done as a pre-processing step: it must be done online. In practice one can carry out this transformation very efficiently since the projection is performed only on the few terms contained in the query, and document vectors are very sparse. Most important, the information needed for this projection is contained in the postings of the query terms. This means that one can compute the needed term statistics on the fly after postings are returned from the inverted index. This will incur the cost of a few extra flops per document score, but without any extra disk or memory access.

The length of the document is not modified by the projection, nor the average document length. The *clause term frequencies* (ctf) and *clause collection frequencies* (ccf) can be computed as:

$$ctf(d, c) := \sum_{(t,w) \in c} w \cdot tf(d, t)$$

$$ccf(d, c) := \sum_{(t,w) \in c} w \cdot \sum_{d} tf(d, t)$$

$$p_{\mathsf{ML}}(c|d) := \frac{ctf(d, c)}{ctf(d, c) + \sum_{t \notin c} tf(d, t)}$$

$$p_{\mathsf{ML}}(c|Col) := \frac{ccf(d, c)}{ccf(d, c) + \sum_{d, t \notin c} tf(d, t)}$$

The most problematic statistic is the *inverse clause frequency* (icf), since this is not clearly defined in the weighted case. One possible choice is the number of postings containing *at least one term in the clause*; we refer this as $icf_{\mathsf{OR}}(c)$, and we note that it can be computed directly from the size of the clause result set (documents with non-zero ctf). However, this number may be unfairly large for clauses with lowly weighted common terms. Furthermore, in some settings this number may not even be available (for example if we only score the query term AND set or if we drop from the computation documents unlikely to be highly scored). Another possibility is to use the *expected idf* for a clause term in a document:

$$icf_{\mathsf{E}}(d,c) = \frac{1}{ctf(d,c)} \sum_{(t,w)\in c} w \cdot tf(d,t) \cdot idf(t) \tag{3}$$

In our empirical evaluation we found this is better than using the *min* or the *max clause idf*, and better than using the *mean idf*.

With these statistics at hand we can compute the relevance score of a document with respect to a query with clauses for a number of retrieval systems[2]; we display several in Table 1.

Table 1. Implementing query clauses in several standard ranking models.

MODEL	WEIGHTING
BM25	$\frac{ctf}{ctf+K} \cdot icf$
VSM	$\frac{ctf \cdot icf}{\|\|d_{\|q}\|\|}$
DFR (PL2)	$\frac{1}{ctf+1}\left(ctf \cdot \log_2 \frac{ctf}{\lambda} + (\lambda - ctf) \cdot \log_2 e + 0.5 \cdot \log_2(2\pi \cdot ctf)\right)$
LM (KL)	$p_{\mathsf{smoothed}}(c\|q) \log\left(p_{\mathsf{smoothed}}(c\|d)\right)$

3 Query Clause Experiments

We have performed experiments to demonstrate the dangers of the term independence assumption for queries with strongly correlated terms, and to test the proposed *query-clauses* ranking idea applied to the stemming problem. Evaluation has been carried out on the Spanish EFE94 corpus which is part of the CLEF collection [10] (approximately 215K documents of 330 average word length and 352K unique index terms) and the 2001 Spanish topic set, with 49 topics of which we only used the title (of 3.3 average word length). All runs employ the standard (equation 1) and the query clause version of BM25 Table 1.

3.1 Stemming

One can view stemming as a form of global query expansion: we expand a term in the query with every term in the dictionary sharing the same stem. However, doing this directly on the query greatly hurts performance (Table 2, rows 1 and 2) . One may think that this loss of performance is due to the noise introduced by the stemming algorithm, but this is not the case: if we replace terms by their stemmed version in the collection and in the query, performance will most often increase and rarely decrease (Table 2 row 3). This is another case of performance being degraded by adding information to the query. In our opinion, this is due to the strong violation of the term independence hypothesis produced by adding so many strongly correlated terms to the query.

A natural way to expand a query by stemming is to construct a set of sets of terms, or a set of *clauses*, where each clause represents all the forms of a stem, possibly weighted (since we may want to weight more strongly the original term typed by the user). The resulting query is a set of sets of clauses which can be ranked with our proposed method. Its performance, using as *idf* the clause's (icf_{OR}) is exactly equivalent to stemming the collection since in both cases term frequencies of stems are collapsed (this is also seen empirically in Table 2, last row)

Table 2. Stemming Performance

Method:	Avg.Prec	Prec10
No Stemming	.37	.47
Stem Expansion (Standard)	.20	.28
Stemming	**.43**	**.52**
Stem Expansion (Clauses)	**.43**	**.52**

3.2 Our Genetic Algorithm

Because we need to perform the selection of a particular set of terms among a huge amount of possible combinations of candidate query terms, the computational complexity of exhaustive search methods is non-viable and we have resorted to a heuristic method such as a genetic algorithm.

Genetic algorithms [7] have been shown to be practical optimization methods in very different areas [9]. Evolutionary algorithms mimic the principles of natural evolution: heredity and survival of the most fit individuals.

A genetic algorithm maintains a population of potential solutions, and is provided with some selection process based on fitness of individuals. The population is renewed by replacing individuals with those obtained by applying "genetic" operators to selected individuals. The usual "genetic" operators are *crossover* and *mutation*. Crossover obtains new individuals by mixing, in some problem dependent way, two individuals, called parents. Mutation gives a new individual

by performing some kind of change on an individual. The production of new generations continues until resources are exhausted or until some individual in the population is fit enough. Figure 1 shows the structure of a genetic algorithm. The algorithm works with a collection of individuals $\{x_i, \cdots, x_n\}$, called population. Each individual represents a potential solution to the problem considered, implemented as some data structure, which depends on the problem. The evaluation of each solution gives a measure of its *fitness*. At a new generation step, a new population is formed by selecting the more fit individuals. Some members of the new population suffer transformations as a consequence of applying *genetic operator* to form new solutions. After a number of generations, the program is expected to converge, and it is hoped that then, the best individual represents a solution close to the optimum.

```
evolution program
begin
    generation = 0
    P = initialize_population
    F = evaluation(P)
    while not required_fitness(F) and
          not termination_condition do
    begin
        generation = generation + 1
        I = individuals_selection(P, F) %for genetic operators
        P = new_generation(P, I)
        F = evaluation(P)
    end
end
```

Fig. 1. Structure of a genetic algorithm

Chromosomes of our GA are fix-length binary strings where each position corresponds to a candidate query term. A position with value one indicates that the corresponding term is present in the query. Because some preliminary experiments we have performed have shown that, in most cases, the elimination of the original query terms degrades the retrieval performance, we force to maintain them among the selected terms of every individual. The set of candidate terms is composed of the original query terms, along with related terms provided by the applied thesaurus. Each term of the original query is grouped with the expanded terms related to it, and this set (*term_set*) [11] is submitted as an individual query. The weights assigned to the documents retrieved with each *term_set* are used to sort the total set of retrieved documents.

The applied selection mechanism has been roulette wheel. In roulette wheel selection, the chances of an individual to be chosen for reproduction are proportional to its fitness. We apply the one-point crossover operator and random mutation [6, 7, 9]. In one-point crossover a single crossover point is chosen on

both parents strings. The parts of the parent strings divided by the crossover point are swapped to generate two children, containing a part of each parent. The random mutation operator simply flips the value of a randomly chosen bit (0 goes to 1 and 1 goes to 0). We also apply elitism, the technique of retaining in the population the best individuals found so far. The fitness function used is some measure of the degree of similarity between a document belonging to the system and the submitted query. We will discuss this further in the different experiments.

4 Experimental results

The system has been implemented in Java, using the JGAP library[1], that provides a generic implementation of a genetic algorithm, on a Pentium 4 processor.

We have carried out experiments to evaluate the fitness functions considered. We have investigated the best range of the GA parameters. Finally, we provide global measures for the whole set of queries that we have considered.

4.1 Selecting the Fitness Function

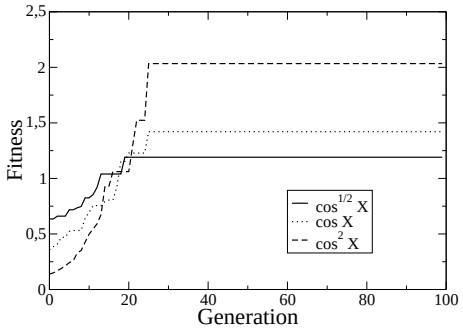

Fig. 2. Fitness functions comparison for the *best_query*, the one for which the greatest precision improvement is achieved.

In the begining, we would like to use Average precision as the fitness function. However, this is not known at query time. Instead, it has been suggested in previous work to use the document scores as the fitness[8]. While this may not be intuitive, it turns out that variations of these scores after expansion are correlated with relevance[1]. One intuitive explanation would be that adding an unrelated term to a query will not bring in new document with high scores, since it is unlikely that it will retrieve new documents; on the other hand adding a

[1] http://jgap.sourceforge.net/

term that is strongly related to the query will bring new documents that also contain the rest of the terms of the query and therefore it will obtain high scores.

We have considered three alternative fitness functions, $\sqrt{\cos\theta}$, $\cos\theta$ and $\cos^2\theta$. To select the fitness function to be used in the remaining experiments, we have studied the fitness evolution for different queries of our test set. Figure 2 compares the fitness evolution for the query which reaches the greatest improvement (*best_query*). The three functions converge to different numerical values that correspond to the same precision value (.68). We can observe that the square-root cosine function if the first one to converge. A similar behavior is observed in other queries. Accordingly, the square-root cosine has been the fitness function used in the remaining experiments.

4.2 Tuning the GA Parameters

The next step has been tuning the parameters of the GA. Figure 3 shows the fitness evolution using different crossover (a) and mutation (b) rates for the best query. Results show that we can reach a quickly convergence with values of the crossover rate around 25%. Mutation rates values around 1% are enough to produce a quick convergence.

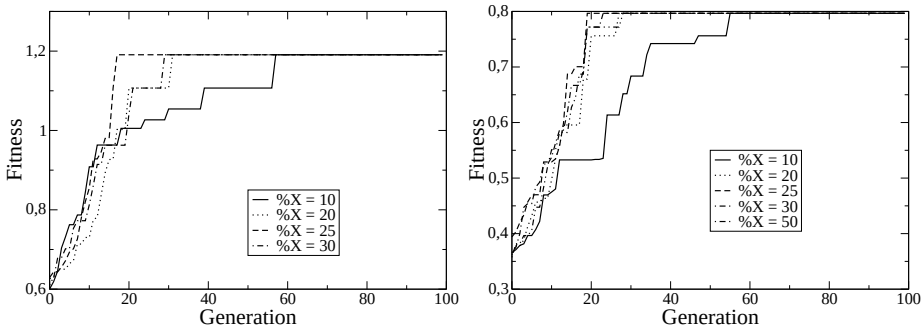

Fig. 3. Studying the best crossover (a) and mutation (b) rates for the best_query.

Figure 4 show the fitness evolution for the best (a) and the worst (b) queries, with different population sizes. The plots indicate that small population sizes, such as one of 100 individuals, are enough to reach convergence very quickly.

4.3 Overall Performance

Table 3 (Stem Expansion (Clauses)) shows the results obtained using stemming as query expansion but building a clause for every term. As expected, the results are exactly those obtained in traditionally stemming by collapsing terms to their stems.

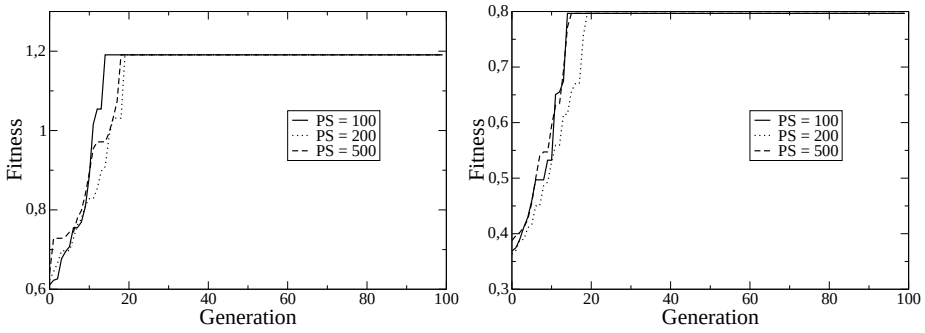

Fig. 4. Studying the population size for the best_query (a) and the worst one (b).

In order to show the improvement of our approach, we have compared the genetic algorithm performance with the results of the original user query (*Baseline*) and with the results obtained expanding with the stems provided by the Porter stemming (*Porter Stemming*). We can observe in Table 3 (Genetic Expansion (Clauses)) that the combination of clauses and genetic algorithm achieved an improvement of the performance, greater than the one achieved with other stemming methods traditionally used in the stemming process, such as Porter.

Table 3. Stemming Performance

Method:	AvgPrec	Prec10	Rel.Δ
No Stemming	.37	.47	-13.9%
Stemming (Baseline)	**.43**	**.52**	0*
Stem Expansion	.20	.28	-53.5%
Stem Expansion (Clauses)	**.43**	**.52**	0
Genetic Expansion	.39	.49	-9.4%
Genetic Expansion (Clauses)	**.45**	**.53**	+4.4%

5 Related works

In most query expansion literature terms are selected (globally from the entire corpus or locally from the top retrieved documents), weighted with respect to their potential relevance and then passed on to a standard retrieval system, which is considered a black box. Here we are concerned only with this black box and not with the expansion process; for this reason we will not review the query expansion literature (an up to date overview can be found in [5]). Some work on user and pseudo-feedback has tackled the issue of term re-weighting, from early Rochio algorithms to more modern probabilistic approaches of relevance feedback. While these works discuss the ranking function, to our knowledge

they all assume Query Term Independence and concentrate on the re-weighting formula. Again, we are not concerned here on the re-weighting of terms (this is left unspecified in our work), and therefore we do not review this literature further (see for example [4]).

A few papers have dealt with the issue of term *correlation* and its effect on retrieval. In [14] the problem of correlation is discussed in depth. They remark that *term correlation* is only an abstract concept and can be understood in a number of ways. They measure term correlation in terms of *term co-occurrence.* Furthermore they propose to represent documents not in the space of terms but in the space of *minterms* which are sets of highly correlated terms. This has the effect of *decorrelating* the terms in the query with respect to hypothetical *concepts* (formally defined as minterms). Instead of computing all term correlations, [13] proposes to mine association rules to compute the most significant term correlations and the rotates the term vectors to reflect the extracted correlations; this yields a more selective term de-correlation. [11] also proposes mining association rules to find term sets of correlated terms. However, the ranking function adjustment proposed is based on the same idea of this paper: collapsing term frequencies within a clause. In fact, if we disregard relative weights, we use the VSM model, and we construct query clauses using association rules in [11], the ranking function here is exactly the same as in [11]. However our work differs from the previously cited papers in that it is not tied to an extraction method or a ranking model, it does not specify the form of the term correlations and furthermore it assumes that term correlations will be *query-dependant.*

6 Conclusions and future work

In this paper we try to show the importance of term dependence issues, how they show up unexpectedly in simple experiments and how they can have a strong adverse effect in performance. Furthermore we propose a method to represent and take into account a simple form of dependence between terms.

On the other hand, we have shown how the clauses can be combined with an evolutionary algorithm to help to reformulate a user query to improve the results of the corresponding search. Our method does not require any user supervision. Specifically, we have obtained the candidate terms to reformulate the query from a *morphological thesaurus*, with provides, after applying stemming, the different forms (plural and grammatical declinations) that a word can adopt. The evolutionary algorithm is in charge of selecting the appropriate combination of terms for the new claused query. To do this, the algorithm uses as fitness function a measure of the proximity between the query terms selected in the considered individual and the top ranked documents retrieved with these terms.

We have investigated different proximity measures as fitness functions without user supervision, such as cosine, square cosine, and square-root cosine. We have also studied the GA parameters, and see that small values such as a population size of 100 individuals, a crossover rate of 25% and a mutation rate of 1%, are enough to reach convergence. Measures on the whole test set of queries

have revealed a clear improvement of the performance, both over the baseline, and over other stemming expansion methods.

A study of the queries resulting after the reformulation has shown that in many cases the GA is able to add terms which improve the system performance, and in some cases in which the query expansion worsen the results, the GA is able to recover the original query.

For the future, we plan to investigate the use of other sources of candidate terms to generate the claused queries applying different query expansion approaches like co-occurrence measures or methods based in Information Theory.

References

1. L. Araujo and J. R. Pérez-Agüera. Improving query expansion with stemming terms: A new genetic algorithm approach. In *EvoCOP*, 2008.
2. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
3. J. P. Callan, W. B. Croft, and S. M. Harding. The inquery retrieval system. In *DEXA*, pages 78–83, 1992.
4. C. Carpineto, R. de Mori, G. Romano, and B. Bigi. An information-theoretic approach to automatic query expansion. *ACM Trans. Inf. Syst.*, 19(1):1–27, 2001.
5. Y. Chang, I. Ounis, and M. Kim. Query reformulation using automatically generated query concepts from a document space. *Inf. Process. Manage.*, 42(2):453–468, 2006.
6. D. E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.
7. J. J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
8. C. Lopez-Pujalte, V. P. G. Bote, and F. de Moya Anegón. A test of genetic algorithms in relevance feedback. *Inf. Process. Manage.*, 38(6):793–805, 2002.
9. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution programs*. Springer-Verlag, 2nd edition edition, 1994.
10. C. Peters and M. Braschler. European research letter: Cross-language system evaluation: The clef campaigns. *JASIST*, 52(12):1067–1072, 2001.
11. B. Pôssas, N. Ziviani, J. Wagner Meira, and B. Ribeiro-Neto. Set-based vector model: An efficient approach for correlation-based ranking. *ACM Trans. Inf. Syst.*, 23(4):397–429, 2005.
12. S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
13. I. R. Silva, J. N. Souza, and K. S. Santos. Dependence among terms in vector space model. In *IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium (IDEAS'04)*, pages 97–102, Washington, DC, USA, 2004. IEEE Computer Society.
14. S. K. M. Wong, W. Ziarko, V. V. Raghavan, and P. C. N. Wong. On modeling of information retrieval concepts in vector space. *ACM Trans. Database Syst.*, 12(2):299–321, 1987.