

Allowing users to weight search terms in information retrieval

Ronald Fagin*
IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099
email: fagin@almaden.ibm.com

Yoëlle S. Maarek
IBM Haifa Research Laboratory
MATAM, Haifa 31905, Israel
email: yoelle@haifa.vnet.ibm.com

Abstract: We give a principled method for allowing users to assign subjective weights to the importance of search terms, that is, the terms forming a query, in information retrieval systems. For example, our method makes it possible for a user to say that she cares twice as much about the first search term as the second search term, and to obtain a ranked list of results that reflects this preference. Our method is based upon a simple formula derived from any existing “unweighted” ranking function. A naive application of the weighted formula would require issuing as many distinct queries as there are search terms, thus damaging the response time of the retrieval. We explain here how to “smoothly” integrate the formula in most retrieval engines, so as not to affect the retrieval performance in terms of response time.

*Most of this research was done while the author was a Research Fellow at the IBM Haifa Research Laboratory.

1 Introduction and Motivation

Users issuing a query (either free-text or Boolean) in an information retrieval (IR) system often feel the need to specify to the system which search terms are more crucial to them. Thus, for the same query, two distinct users might have different perspectives, and might wish to give that information to the retrieval engine.

Consider the following example, where a user is interested in Internet security, and wants to search newswires for recent press releases related to that domain. She accesses a relevant search engine and issues the following query: “*Internet security products covering proxy servers, firewalls and encryption mechanisms*”.

Most modern IR systems will not consider search terms in this query equally, but instead evaluate their relative importance by considering their distribution in the full document collection. The intuition behind this approach is that the more frequent a term is in a collection, the less discriminating it is.¹

However, in the above mentioned example, our user might be more interested in *encryption mechanisms*, while another user might be more interested in *proxy servers*. As far as the collection of documents is concerned, these two concepts are probably similarly important (unlike more common words like *products*). Therefore, the retrieval system will consider them equally, and return the following ranked list of results (where for the sake of the simplicity, we show here only the top five candidates):

1. **Title:** Firewall 2.0 for Windows NT
Hits: *server , proxi , firewal*
2. **Title:** Internet Dynamics Unveils Encryption TCP/IP Software 12/01/97
Hits: *encrypt , firewal*
3. **Title:** Lucent Managed Firewall Intro'd 12/05/97
Hits: *server , firewal*
4. **Title:** Integralis Releases MIMESweeper for CheckPoint FireWall-1
Hits: *firewal*
5. **Title:** Raptor Systems Announces Major Enhancements to Eagle Enterprise Firewall
Hits: *server , encrypt , proxi , firewal*

The results list shows for every document a “hit list” that consists of the search terms (or rather their stems) that were actually considered as a match in the considered document. According to the hit lists above, we can see that some top documents deal more with *firewalls* or *proxy servers* than with *encryption*. However, considering the user’s interests, she would be more

¹The most classical embodiment of this approach is the family of $tf \times idf$ scores [2, 6] where tf stands for the frequency of a term in a document, and idf for the “inverse document frequency”.

satisfied with a different ranking of candidates that would give preference to those candidates that deal more directly with *encryption*. Other candidates, while being still relevant, should get a lower ranking.

We propose here a mechanism for letting the user manually (and subjectively) assign a higher weight to the term *encryption* (via any kind of GUI artifact), and making the retrieval engine produce a ranking that reflects the user's preferences. Thus, in the example cited above, once she has assigned a higher weight to the term *encryption* in her query, our user would obtain a different ranked list of results as shown below (only the top five are listed here as well). Some "new" documents that were at lower ranks in the previous example, and thus not shown, suddenly make their appearance in the top five.

1. **Title:** Internet Dynamics Unveils Encryption TCP/IP Software 12/01/97
Hits: *encrypt , firewall*
2. **Title:** Raptor Systems Announces Major Enhancements to Eagle Enterprise Firewall
Hits: *server , encrypt , proxy , firewall*
3. **Title:** New Oak Ships Industry's First Directory-Enabled Networking Product
Hits: *server , encrypt , firewall*
4. **Title:** Secure Computing Anchors Its Family of Security Products for the Windows NT Env
Hits: *server , encrypt , proxy , firewall*
5. **Title:** Firewall 2.0 for Windows NT
Hits: *server , proxy , firewall*

This approach is not equivalent to post-filtering the results, or using a Boolean AND constraint to force the term *encryption* to appear; indeed, the fifth document here is still relevant to the query, although it does not deal directly with encryption. Intuitively, it is only the ordering of the results that is affected by the weights.

This feature exists already in some IR systems, but is typically implemented via ad hoc heuristics such as arbitrarily increasing the contributing score of the preferred search terms. Some expert users who understand the underlying retrieval model of some given search engine can often "trick" the retrieval process by simply repeating the preferred terms in the query as if they were appearing more so as to increase their *tf* factor (see footnote 1) for instance and thus artificially boost documents that contain it.

We propose here a principled method for taking user-assigned weights into account in the ranking process. This method is based on a formula, developed by Fagin and Wimmers [1] for evaluating queries in a multimedia database system. (Fagin and Wimmers were interested in a situation where, say, the user searches for objects that are red and round, and where, say, the user cares twice as much about the color as the shape.) As we shall discuss, we both extend

and simplify Fagin and Wimmers' approach, in order to apply it to the IR domain. As another contribution, we show how the formula can be implemented efficiently in retrieval components. A naive "direct" implementation of the formula would require issuing as many distinct queries as there are search terms, which would significantly affect the efficiency of the retrieval engine. We show here that in most retrieval systems (typically those based on the vector space model), this multiple issuing of queries is not necessary, and the same results can be achieved via a simple modification of the retrieval scheme that does not affect the retrieval response time.

2 A Principled Formula for Incorporating Weights into Rules

In this section, we discuss a technique developed by Fagin and Wimmers [1] for incorporating user preferences in multimedia queries. We show how we both extend and simplify their methodology, in order to apply it to IR systems.

In a multimedia database system, queries may be fuzzy: thus, the answer to a query such as $(Color='red')$ may not be 0 (false) or 1 (true), but instead a number between 0 and 1. A conjunction, such as $(Color='red') \wedge (Shape='round')$, is evaluated by first evaluating the individual conjuncts and then combining the answers by some scoring function. Typical scoring functions include the min (the standard scoring function for the conjunction in fuzzy logic) and the average. Fagin and Wimmers were concerned with the issue of permitting the user to weight the importance of atomic subformulas (so that, for example, the user can say that she cares twice as much about the color as the shape). Thus, they dealt with the following question. Assume that we are given a scoring function (such as the min function). How should this function be "modified" so that it is possible to weight the importance of the arguments?

We face a similar situation in our IR application. We are given the indexing units, or for short, search terms, extracted from a query, and we are given a search engine that assigns a relevance score telling how well these search terms match a given document. We want to know how the search engine should be modified so that it is possible to weight the importance of the search terms.

In [1], the "input" to the method is a collection of scoring functions, one for each set of attributes. For example, if the attributes of interest are color, shape, and texture, then there are seven scoring functions, one for every nonempty subset of {color, shape, texture}. Then the scoring function that deals with, say, the subset {color, shape} tells how to combine the color score and the shape score to obtain a combined score.

One of our contributions in this paper is the realization that there is an unnecessary restriction in [1]: the methodology there is applied only to scoring functions (which combine a tuple of numbers, such as the color score and the shape score, into a single number). We wish to allow the arguments to be non-numerical (namely, search terms). Further, instead of dealing with the complexity of a set of (scoring) functions, we realized that all that is needed is that there be an underlying *rule*, which is an assignment of a value to every tuple, of varying sizes.

Fagin and Wimmers give an explicit formula for incorporating weights. Their formula is surprisingly simple, in that it involves far fewer terms than one might have guessed. It has three further desirable properties. The first desirable property is that when all of the weights are equal, then the result is obtained by simply using the underlying rule. Intuitively, this says that when all of the weights are equal, then this is the same as considering the unweighted case. The second desirable property is that if a particular argument has zero weight, then that argument can be dropped without affecting the value of the result. The third desirable property is that the value of the result is a continuous function of the weights. They prove that if these three desirable properties hold, then under one additional assumption (a type of local linearity), their formula gives the unique possible answer.

We now describe the framework of [1] and how we simplify it to fit our application. Let d be a fixed document. Let f be a function whose domain is the set of all tuples (of all sizes) over some common domain, and with range the set of real numbers. In our application, if x_1, \dots, x_m are search terms, then $f(x_1, \dots, x_m)$ is interpreted as the relevance of document d with respect to these search terms. Typically, search results consist of a list of documents ranked by decreasing f value (the list being possibly truncated according to the settings of the considered retrieval system). In situations where we want to make explicit the dependence of f on d (like in Section 3), we may write $f^{(d)}$ instead of simply f .

One reason that we are able to simplify the framework of [1] (so that in particular, the notation is much easier) is that in our domain, the function f is symmetric, in the sense that in many retrieval models, the order of the indexing units² does not matter. We could actually have taken the arguments of f to be sets, rather than tuples, but this does not lead to any simplification in the notation.

Assume that $\theta_1, \dots, \theta_m$ are all nonnegative and sum to one. Then we refer to $\Theta = (\theta_1, \dots, \theta_m)$ as a *weighting*. Intuitively θ_i is the weight of search term x_i . If $\theta_1 \geq \dots \geq \theta_m$, then we refer to the weighting $\Theta = (\theta_1, \dots, \theta_m)$ as *ordered*. For each ordered weighting $\Theta = (\theta_1, \dots, \theta_m)$, we obtain (using the methodology of [1]) a function f_Θ whose domain consists of tuples of length m (the length of Θ). Intuitively, if $X = (x_1, \dots, x_m)$ is a tuple of search terms, then $f_\Theta(X)$ is the relevance of document d with respect to the search terms x_1, \dots, x_m , when θ_i is the weight of search term x_i . For example, $f_{(\frac{2}{3}, \frac{1}{3})}(dog, flea)$ is the relevance of document d when we assign weight $\frac{2}{3}$ to the search term “dog” and weight $\frac{1}{3}$ to the search term “flea”. This reflects the situation where the search term “dog” is considered twice as important as the search term “flea”. We note that in [1], the function f_Θ is defined even when the weighting Θ is not ordered. It is necessary in [1] to consider also weightings that are not ordered, since f is not necessarily a symmetric function there.

Fagin and Wimmers give the following desiderata for the functions f_Θ :

D1. $f_{(\frac{1}{m}, \dots, \frac{1}{m})}(x_1, \dots, x_m) = f(x_1, \dots, x_m)$. That is, if all of the weights in Θ are equal, then the

²Note that we assume here that the reader has some understanding of information retrieval theory, and is familiar with concepts such as indexing, profiles, the vector space model, etc. More novice readers are referred to [6] for introductory material.

“weighted” function f_Θ coincides with the “unweighted” function f .

D2. $f_{(\theta_1, \dots, \theta_{m-1}, 0)}(x_1, \dots, x_m) = f_{(\theta_1, \dots, \theta_{m-1})}(x_1, \dots, x_{m-1})$. That is, if a particular argument has zero weight, then that argument can be dropped without affecting the value of the result.

D3. $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ is a continuous function of $\theta_1, \dots, \theta_m$.

Fagin and Wimmers give the following choice for $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ (modified to fit our terminology):

$$(\theta_1 - \theta_2) \cdot f(x_1) + 2 \cdot (\theta_2 - \theta_3) \cdot f(x_1, x_2) + 3 \cdot (\theta_3 - \theta_4) \cdot f(x_1, x_2, x_3) + \dots + m \cdot \theta_m \cdot f(x_1, \dots, x_m). \quad (1)$$

It is straightforward to verify that **D1**, **D2**, and **D3** are satisfied when we take $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ to equal (1). This formula (1) for $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ is the formula we adopt for use in our application.

Although the formula (1) may look somewhat arbitrary, it is shown in [1] that it is actually uniquely determined, under one additional assumption that we now discuss. We say that our collection of weighted functions f_Θ is *locally linear* if

$$f_{\alpha \cdot \Theta + (1-\alpha) \cdot \Theta'}(X) = \alpha \cdot f_\Theta(X) + (1 - \alpha) \cdot f_{\Theta'}(X), \quad (2)$$

whenever Θ, Θ', X are of the same length, and $\alpha \in [0, 1]$. (The definition of local linearity in [1] says essentially that (2) must hold whenever Θ and Θ' are ordered. For us, this is always the case, since we define f_Θ only when Θ is ordered.) Note that $\alpha \cdot \Theta + (1 - \alpha) \cdot \Theta'$ is ordered whenever Θ and Θ' are ordered.

Define the condition **D3'** as follows:

D3'. The collection of weighted functions f_Θ is locally linear.

Condition **D3'** implies condition **D3** above (that $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ is a continuous function of $\theta_1, \dots, \theta_m$) [1]. Furthermore, the choice of (1) for $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ is the unique one that satisfies **D1**, **D2**, and **D3'** [1].

We now discuss local linearity. Intuitively, local linearity says that the scoring functions act like a balance. Local linearity demands that the weighting that is the midpoint³ of two ordered weightings should produce a value that is the midpoint of the two values produced by the given weightings, or in other terms that (2) must hold when Θ and Θ' are ordered, and so agree on

³In fact, local linearity extends beyond the midpoint to any weighting that is a convex combination of two ordered weightings: if a weighting is a convex combination of two ordered weightings, then local linearity demands that the associated value should be the same convex combination of the values associated with the given weightings. In this paper, our life is made simpler by the fact that we consider weighted functions f_Θ only for weightings Θ that are ordered. In [1], where the function f_Θ is defined even when Θ is not ordered, it is shown that an assumption that (2) holds for *every* choice of Θ and Θ' would be incompatible with the properties **D1** and **D2**, unless $f(x_1, \dots, x_k) = \frac{f(x_1) + \dots + f(x_k)}{k}$ for every k .

which search term is the most important, which is the second most important, etc. Local linearity says that in this case, we do a linear interpolation, which is a very natural assumption. Another argument in favor of local linearity is that it leads to such a nice formula, namely, (1).

The formula (1) for $f_{\Theta}(X)$ is a convex combination of the values $f(x_1)$, $f(x_1, x_2)$, $f(x_1, x_2, x_3)$, \dots , $f(x_1, \dots, x_m)$, since the coefficients $(\theta_1 - \theta_2)$, $2 \cdot (\theta_2 - \theta_3)$, $3 \cdot (\theta_3 - \theta_4)$, \dots , $m \cdot \theta_m$ are nonnegative and sum to 1. A surprising feature of (1) is that it depends only on the m terms $f(x_1)$, $f(x_1, x_2)$, $f(x_1, x_2, x_3)$, \dots , $f(x_1, \dots, x_m)$, and not on any of the other possible terms, such as $f(x_2)$, $f(x_1, x_3)$, and so on. *A priori*, we might have believed that $f_{\Theta}(X)$ would depend on all of the $2^m - 1$ such terms.

3 Implementation of the formula

The formula (1) can always be used to obtain a document/query similarity score in the weighted case from the scores in the unweighted case. In this section, we show that in a number of cases the score can be obtained essentially as efficiently in the weighted case as in the unweighted case. We begin by showing how the formula (1) can be computed in the classical case of the vector space model, via a simple modification within the retrieval engine, and essentially without affecting its overall performance.

Consider a given query in free-text format, and its associated profile, which we denote by (x_1, \dots, x_m) , where each x_i represents a search term. Most modern retrieval systems define a ranking formula for each document in the considered collection that expresses how relevant that very document is to the query, via a certain measure of similarity between the document and the query. In the previous section, we denoted this ranking formula for document d by $f^{(d)}(x_1, \dots, x_m)$ (or simply $f(x_1, \dots, x_m)$, when the document d was being held fixed). We now discuss how in the vector space model we can actually compute the weighted ranking formula $f_{\Theta}^{(d)}(x_1, \dots, x_m)$.

Let us first discuss the implementation in the unweighted case of a typical ranking formula derived from $tf \times idf$ in the vector space model. In practice, systems implementing this model do not compute the $f^{(d)}$ value for every document in the collection. Instead, by an inverted index file, such systems access only the postings of those search terms that appear in the query, and update an accumulator that stores the score of every document that shares at least one search term with the query⁴. We do not take into account here a possible query normalization, as it is often considered that it is much less important than document normalization because of the small length and length differences of typical queries (see [5], p 306).

In such traditional systems, the accumulator can be updated using the following simple algorithm:

⁴Some heuristics such as partial list searching [5] could also be used in which even less documents are considered. However, the explanation stated here still holds in that case.

Let $acc[d]$ be the entry for document d in the score accumulator
Init all accumulator entries to 0
For each x_i in the query profile
 Retrieve its associated postings list $\{(d_{i1}, \rho_i(d_{i1})), (d_{i2}, \rho_i(d_{i2})), \dots, (d_{in_i}, \rho_i(d_{in_i}))\}$
 For each document d_{ik} (with $k = 1, \dots, n_i$) in the postings list
 Do $acc[d_{ik}] = acc[d_{ik}] + g(\rho_i(d_{ik}), x_i)$
Sort the accumulator by decreasing order
Return the list of documents sorted according to score accumulator value

Here d_{i1}, \dots, d_{in_i} are the documents associated in the inverted index file with the search term x_i . Also, $\rho_i(d_{ik})$ represents some numerical information pertinent to the occurrence of search term x_i in document d_{ik} ; in the case we are considering, this is derived from the occurrence count tf . Finally, $g(\rho_i(d_{ik}), x_i)$ gives a value that depends on the document-specific information $\rho_i(d_{ik})$ and on collection-specific information about the search term x_i , such as idf . In our case of interest, $g(\rho_i(d_{ik}), x_i)$ gives a score derived from the tf and idf values for the search term x_i .

It can be shown easily that for additive ranking measures such as the cosine measure of similarity as classically used in the vector space model, applying the above algorithm gives the appropriate value. In other words, at the end of the accumulative process, we have

$$acc[d] = \sum_{i=1}^m g(\rho_i(d), x_i) = f^{(d)}(x_1, \dots, x_m).$$

Now let us consider the weighted case, where the search terms x_1, x_2, \dots, x_m are associated with a weighting $\Theta = (\theta_1, \dots, \theta_m)$ as defined in Section 2 (for the sake of clarity, we order the search terms here by decreasing θ_i value). We now show how to modify this simple accumulator algorithm to use in the weighted case, so that the $acc[d]$ value at the end of the process is equal to the value given by formula (1).

In the absence of query normalization (which we are now assuming), one can easily verify that for each j (with $1 \leq j \leq m$), we have

$$f^{(d)}(x_1, \dots, x_j) = f^{(d)}(x_1) + \dots + f^{(d)}(x_j). \quad (3)$$

Therefore, from (1), we obtain

$$\begin{aligned}
 f_{(\theta_1, \dots, \theta_m)}^{(d)}(x_1, \dots, x_m) &= ((\theta_1 - \theta_2) + 2(\theta_2 - \theta_3) + \dots + (m-1)(\theta_{m-1} - \theta_m) + m\theta_m) \cdot f^{(d)}(x_1) \\
 &+ (2(\theta_2 - \theta_3) + \dots + (m-1)(\theta_{m-1} - \theta_m) + m\theta_m) \cdot f^{(d)}(x_2) \\
 &+ \dots \\
 &+ m\theta_m \cdot f^{(d)}(x_m)
 \end{aligned}$$

Let us denote the coefficient of $f^{(d)}(x_i)$ above by α_i , for $1 \leq i \leq m$, so that $f_{(\theta_1, \dots, \theta_m)}^{(d)}(x_1, \dots, x_m) = \sum_{i=1}^m \alpha_i \cdot f^{(d)}(x_i)$. Hence, the accumulator algorithm stated above (which deals with the unweighted case) can be modified so as to take into account weighted terms as follows:

Let $acc[d]$ be the entry for document d in the score accumulator
Init all accumulator entries to 0
For each x_i in the query profile
 Retrieve its associated postings list $\{(d_{i1}, \rho_i(d_{i1})), (d_{i2}, \rho_i(d_{i2})), \dots, (d_{in_i}, \rho_i(d_{in_i}))\}$
 For each document d_{ik} (with $k = 1, \dots, n$) in the postings list
 Do $acc[d_{ik}] = acc[d_{ik}] + \alpha_i \cdot g(\rho_i(d_{ik}), x_i)$
Sort the accumulator by decreasing order
Return the list of documents sorted according to score accumulator value

Note that the only change from the unweighted accumulator algorithm is to replace $g(\rho_i(d_{ik}), x_i)$ by $\alpha_i \cdot g(\rho_i(d_{ik}), x_i)$. Now $1 = \alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_m \geq 0$ (this follows from the fact, noted in Section 2, that the coefficients $(\theta_1 - \theta_2), 2 \cdot (\theta_2 - \theta_3), 3 \cdot (\theta_3 - \theta_4), \dots, m \cdot \theta_m$ of (1) are nonnegative and sum to 1). Therefore, the net effect of the change in the weighted accumulator algorithm from the unweighted accumulator algorithm is to multiply the term associated with x_i (namely, $g(\rho_i(d_{ik}), x_i)$) by a multiplier α_i that takes on the value 1 for the highest-weighted x_i (namely, x_1), and whose value decreases (or remains the same) the less highly weighted x_i is (that is, the bigger i is). On the face of it, this is a reasonable heuristic. The formula (1) determines exactly what the values of these multipliers α_i should be.

There are other models (such as simple probabilistic models) where an analog of the unweighted accumulator algorithm is valid, and where (3) holds, so that we can obtain the analog of the weighted accumulator algorithm given above. But it is important to note that there are many other information retrieval systems where these assumptions do not hold, and yet where we can convert an algorithm for the unweighted case into an algorithm for the weighted case, without drastically changing the architecture or affecting the performance. This applies to systems where

1. There is an inverted index file, which for each search term x_i gives those documents d that are relevant to x_i , possibly along with other information. (This information could be about, say, the number and/or location of occurrences of x_i in document d .) From the information associated with x_1 , the information associated with x_2 , \dots , and the information associated with x_j , it is possible to compute (quickly) $f^{(d)}(x_1, \dots, x_j)$, for each document d .
2. The value $f^{(d)}(x_1, \dots, x_j)$ is positive precisely if document d is relevant to at least one of x_1, \dots, x_j , and otherwise $f^{(d)}(x_1, \dots, x_j) = 0$.
3. Given search terms x_1, \dots, x_j , we are interested only in determining some subset of the documents d such that $f^{(d)}(x_1, \dots, x_j)$ is positive.

Intuitively, the first condition shows what the inverted index file is for: to determine certain documents d that are relevant to the search terms x_i . Taken together, these conditions intuitively say that given the search terms x_1, \dots, x_j , the relevant documents d that we wish to determine all have $f^{(d)}(x_1, \dots, x_j)$ positive, and these documents d can be obtained by using information in the inverted index file about x_1, \dots, x_j .

Let D be the set of all documents d that are relevant to at least one of x_1, \dots, x_m . We then see from the formula (1) that $f_{\Theta}^{(d)}(x_1, \dots, x_m)$ is positive only for a subset of the documents d in D . Therefore, the documents d that are relevant in the weighted case (where in particular $f_{\Theta}^{(d)}(x_1, \dots, x_m)$ is positive) can be obtained by using only information in the inverted index file about x_1, \dots, x_m . Hence, we can use the inverted index file to determine the relevant documents in the weighted case, just as we could in the unweighted case. Further, we look at the inverted index file only to find values for the search terms x_1, \dots, x_m , just as in the unweighted case.

4 Some experiments

We have conducted some experiments in order to evaluate the impact on retrieval effectiveness of applying the formula (1). We implemented the weighted formula algorithm (as described in Section 3) in the accumulator module of a regular retrieval component and experimented with several ranking models (from the standard vector space model to a more experimental probabilistic model), all using a regular inverted index file. Unfortunately, we did not have access to test collections coming with weighted test queries. Therefore, we decided to conduct a simple experiment on “old-fashioned” test collections such as the ones from the Virginia-Disc (namely MED, CACM, CRAN, etc.), to get an indication on how retrieval effectiveness, in terms of recall and precision, was affected by the weighted formula.

Considering our lack of a valid test collection, we decided to conduct the following experiment. We took the queries coming with the Virginia-Disc test collections, and somewhat arbitrarily assigned a weighting Θ to the search terms extracted from the query (as discussed below, we assigned the highest weight to the search term we subjectively considered the most relevant). We generated a set of such weighted queries, and ran them against the test collections with a weighted retrieval engine, that is, modified with the accumulator model described in the previous section. Since the MED, CACM, etc. collections provide relevance information in a “0-1” form (that is, a document is either irrelevant or relevant), we took the set of “relevant documents” to be the same in the weighted and unweighted cases.

With this fairly arbitrary “weighted” test collection model, we compared the recall/precision of the experimental system Guru [3, 4] (using a variety of models from the standard $tf \times idf$ model to a simple probabilistic model with an inverted index file) in weighted and unweighted cases. More precisely, we compared the results obtained for a set of possible weightings on test queries against the results obtained under the same model, with unweighted test queries.

We tried several choices for the weighting Θ , and verified that whatever the weighting is, as long as a non-null weight⁵ is given to each of the search terms, the recall/precision was always higher than in the unweighted case. We also found experimentally that we achieved the best effectiveness if we reserved 0.6 for the search term considered (subjectively) the most relevant, and distributed the remaining 0.4 equally between the remaining search terms. More precisely,

⁵A null weight would be equivalent to *not* taking the term into account, and would therefore significantly modify the test query.

as expected, at a low level of recall, higher precision was achieved, with the improvement at the 0.1 recall level being up to 10%. In other words, relevant candidates were moved higher in the ranked list of results.

However, since our weighted test collection was arbitrarily built, and since we did not choose the weights according to any objective testing procedure (but using instead our subjective interpretation), these experiments should be seen only as a positive indication. More objective evaluation should be conducted in a context of a formal framework such as TREC.

5 Summary of contributions

We have given a principled method for taking into account weights subjectively assigned by the user to search terms, and for ranking search results accordingly. Instead of using an ad hoc heuristic as is typically done, we provide a formula that is derived from the regular ranking formula (or relevance function) used for unweighted queries. This formula presents the key advantages of being very simple and being unique under certain constraints that seem intuitively desirable.

We now consider what our contributions have been. First, we have introduced to the IR community the formula in [1], which was designed originally to deal with queries in a multimedia database system. In doing so, we made a “conceptual leap”: we realized that the formula in [1] can be applied whenever there is an underlying rule (an assignment of a value to every tuple, of varying sizes) that we want to convert into a weighted version (where we can weight the importance of each argument). By contrast, Fagin and Wimmers considered only methods for combining various numerical scores into an overall numerical score. Furthermore, our application enables us to simplify the notation of [1] significantly. We show how to implement the weighted formula efficiently in our application; this is important because a naive application of the weighted formula would require issuing as many distinct queries as there are search terms. Finally, we conducted experiments about the effects of weighting search terms; these experiments were encouraging.

6 Acknowledgments

We are grateful to Victoria Skoblikov for conducting the recall/precision experiments. We are also thankful to Byron Dom, David Notkin and Frank Smadja for their valuable comments and suggestions in the writing of this paper.

References

- [1] R. Fagin and E. L. Wimmers. Incorporating user preferences in multimedia queries. In *Proceedings of the 1997 International Conference on Database Theory*, pages 247–261, 1997.
- [2] D. Harman. Ranking algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval, Data Structure and Algorithms*, pages 241–263. Prentice Hall, 1992.
- [3] Y.S. Maarek. Software library construction from an IR perspective. *SIGIR Forum*, 25(2):8–18, Fall 1991.
- [4] Y.S. Maarek and F.A. Smadja. Full text indexing based on lexical relations. an application: Software libraries. In N.J. Belkin and C.J. van Rijsbergen, editors, *Proceedings of SIGIR'89*, pages 198–206, Cambridge, MA, June 1989. ACM Press.
- [5] G. Salton. *Automatic text processing, the transformation, analysis and retrieval of information by computer*. Addison-Wesley, Reading, MA, 1989.
- [6] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. Computer Series. McGraw-Hill, New York, 1983.