

Allowing users to weight search terms

Ronald Fagin¹

IBM Almaden Research Center
San Jose, CA 95120-6099, USA
fagin@almaden.ibm.com

Yoëlle S. Maarek

IBM Research Laboratory in Haifa
MATAM, Haifa 31905, Israel
yoelle@il.ibm.com

Abstract

Information retrieval systems typically weight the importance of search terms according to document and collection statistics (such as by using $tf \times idf$ scores, where less common terms are given higher weight). We consider here the scenario where a user can express her own subjective weighting of the importance of the terms that form the query on top of the system-generated weighting, and show how this should modify the relevance scores of documents. This has been allowed before, but only by ad hoc heuristics. We give the first principled method for taking into account the user's subjective weighting of the importance of query terms. Our method is based on an approach by Fagin and Wimmers, that gives a simple formula derived from any existing "unweighted" ranking function. A naive application of the formula would require issuing as many distinct queries as there are terms in the query (search terms), thus damaging the response time of the retrieval. We explain here how to "smoothly" integrate the formula in most information retrieval systems so as not to affect the retrieval performance in terms of response time.

Appeared in: **Proc. RIAO (Recherche d'Informations Assistée par Ordinateur = Computer-Assisted Information Retrieval) '2000**, pp. 682–700.

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 terms in the query, which we will refer to in the following as "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 IR engine. In this paper, we assume that the user can assign a weight to each search term (where the default would be that each search term is given equal weight). We now consider two scenarios for this approach.

¹Most of this research was done while the author was a Research Fellow at the IBM Haifa Research Laboratory.

Scenario 1: Modifying the IR engine. Most IR engines will not consider search terms 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. The most classical embodiment of this approach is the family of $tf \times idf$ scores [Har92, SM83] where tf stands for the “term frequency” of a term in a document, and idf for the “inverse document frequency”.

The possibility of allowing the user to assign weights to search terms exists already in some IR engines (although apparently not in any of the popular Web search services), but is typically implemented via ad hoc heuristics, such as arbitrarily increasing the contributing score of the preferred search terms. As a typical example, in the INQUERY system [BCCN95], the user may be given the option of modifying the weightings in the tf factor. Similarly, some expert users who understand the underlying retrieval model of some given IR 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, and thus artificially boost the relevance scores of documents that contain it. We propose here a principled method for taking user-assigned weights into account in the ranking process (using any kind of GUI artifact). Note that user-assigned weights might be drastically different from the relative importance assigned to each term (or indexing unit) by the IR engine according to collection statistics. In the first case, we refer to a manually assigned subjective weighting, while in the latter, we have an “objective” weighting derived from the intuition that the more frequent a term is in a collection, the less discriminating it is.

Scenario 2: Post-processing the results of an IR engine. We now consider another scenario in which the user might wish to weight the importance of search terms. Assume that the user gives a standard (unweighted) query to an IR engine, and the IR engine gives back a ranked list of results. The user might wish to use a post-processor to rerank the results. Such a situation might arise because the user is not satisfied with the results, since some documents that prioritize one perspective of the query get a higher rank (and steal “real estate” on the top of the ranked list) than others that reflect what the user really intended. To answer such needs, most search services give users the option to refine results. Apart from the traditional “relevance feedback” facility, some search services propose simpler refinement paradigms. Thus, on the Web, Excite² proposes a set of search terms to be added to the query by the user, and Infoseek³ allows users to search within the search results. In the same spirit as the latter, we propose here a less drastic approach that does not actually prune the search results, but instead simply reranks them according to user-assigned weights on the query terms. In other words, instead of doing a binary choice, as to whether search terms are included or not, we propose a more continuous approach, where search terms are each given a weight between 0 and 1.

Consider the following example, where a user is looking for information about tax treaties between the U.S. and France, and issues the query “tax treaties US France” to Infoseek.⁴ She will get a number of documents, dealing with tax treaties or simply U.S. treaties, ranked higher

²<http://www.excite.com>

³<http://www.infoseek.com>

⁴Note that we use Infoseek here for the sake of the example. For that particular query, we did in fact find much more authoritative answers via Yahoo or Google, for instance.

than the first relevant document, which comes as candidate number 21:



Figure 1: InfoSeek 21st candidate to “U.S.-France tax treaties”

She might improve her results significantly by using the “Search within results” feature (See one of the two buttons below the search field in Fig. 1) using the term “France”. She will get then the same candidate in second position.

However, in the above mentioned example, our user might still be interested in other related tax treaties (e.g., with Spain) and might want a finer control than a binary choice regarding the presence/absence of the term “France”. We propose here a mechanism for letting the user manually (and subjectively) assign a higher weight to the term “France”, so that the IR engine will 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 “France” in her query, our user would obtain a different ranking of the results. Some “new” documents, that were at lower ranks in the previous example, suddenly make their appearance in the top five.

In this paper, we propose a principled method for taking user-assigned weights into account in the ranking process, that can be applied in either scenario. This method is based on a formula, developed by Fagin and Wimmers [FW97] 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 weighting search terms. As another contribution, we show how the formula can be implemented efficiently in IR engines. 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. We show here that in most IR engines (typically those based on the vector space model), this multiple issuing of queries is not necessary, and the same results can be achieved via

Your search for [tax treaties: US France](#)
 - france resulted **INFOSEEK SEARCH** [Tips](#) | [Advanced search](#) | [GOguardian\[tm\] is off](#)

In:

[390 directory](#)

[topics](#) New search Search within results
[26,596/1kb](#)
[search results](#)

Infoseek web search results

RESULTS 1 - 10 of 26,596 total results, grouped by site

[Hide summaries](#) | [Sort by date](#) | [Ungroup results](#) | [Next 10](#)

Current Tax Developments - New Tax Treaty Signed with Japan - Archibald Andersen
 Archibald Andersen Current Tax Developments - New Tax Treaty Signed with Japan Date: 21 December 1995 France and Japan signed a new income tax treaty last March to replace ...
 62% **Date: 2 Dec 1998**, Size 5.7K,
<http://www.mondaq.com/docs/noframes/1001086.html>
[Find similar pages](#) | [Grouped results from www.mondaq.com](#)

Mondaq Ltd - Ernst & Young, France
 Date: 17 March 1995 . Taxation in France - Tax Treaties - HSD Ernst & Young. France has concluded numerous tax treaties that eliminate double taxation on the worldwide income of resident taxpayers. If no tax treaty has been concluded, mitigation ...
 56% **Date: 5 Dec 1997**, Size 8.4K,
<http://www.businessmonitor.co.uk/docs/table/tax13.html>
[Find similar pages](#) | [Grouped results from www.businessmonitor.co.uk](#)

Figure 2: InfoSeek results after “Search within results” on term “France”

a simple modification of the retrieval scheme that does not affect the retrieval response time.

Section 2 discusses Fagin and Wimmers' formula for incorporating weights into scoring rules, which is unique under certain natural assumptions, and gives our extension and simplification. Section 3 shows how, in practice, the weighted ranking formula can be integrated in typical IR engines while not drastically changing their architecture or affecting their efficiency in terms of response time. In Section 4, we describe the integration of the reranking feature into Fetuccino [BSHJ⁺99], and continue our discussion of our running example of U.S.-France tax treaties. Section 5 considers the effects, in the weighted case, of allowing search terms to be prefaced by the plus symbol (+) and the minus symbol (−) (in the unweighted case, the plus symbol typically means that the search term must appear in the document, and the minus symbol typically means that the search term must not appear in the document). Section 6 summarizes our contributions.

2 A Principled Formula for Incorporating Weights into Scoring Rules

In this section, we discuss a technique developed by Fagin and Wimmers [FW97] for incorporating user preferences in multimedia queries. We show how we both extend and simplify their methodology, in order to apply it to weighting search terms.

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 aggregation function. Typical aggregation functions include the min (the standard aggregation 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 an aggregation function. How should this function be “modified” so that it is possible to weight the importance of the arguments?

Let us consider more closely the situation where the user cares twice as much about the color as the shape. How should we combine the color score and the shape score to obtain an overall score? If the aggregation function is simply to take the average, then the answer is fairly clear. We would assign a weight $\theta_1 = 2/3$ to the color, and a weight $\theta_2 = 1/3$ to the shape. (The weights must sum to one, and the weight for color, namely θ_1 , should be twice the weight θ_2 for shape.) We then take the weighted average $\theta_1 x_1 + \theta_2 x_2$. But what if we are using a different aggregation function than the average for combining scores? For example, let us assume that we are using the min. Then as we now show, we cannot simply take the result to be $\theta_1 x_1 + \theta_2 x_2$. For, consider the case where $\theta_1 = \theta_2 = 1/2$, which corresponds to the situation where the user cares equally about the color and the shape. Then $\theta_1 x_1 + \theta_2 x_2$ gives us the wrong answer: it gives the average, not the min. How should the min function be “modified” so that it is possible to weight the importance of the arguments?

We face a similar situation in our search application. We are given the indexing units, or for short, search terms, extracted from a query, and we are given an IR 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 [FW97], the “input” to the method is a collection of aggregation functions, one for each set of attributes. For example, if the attributes of interest are color, shape, and texture, then there are seven aggregation functions, one for every nonempty subset of {color, shape, texture}. Then the aggregation 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 [FW97]: the methodology there is applied only to aggregation 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 (aggregation) functions, we realized that all that is needed is that there be an underlying *scoring 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 scoring 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 our modification of the framework of [FW97]. 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 on d (like in Section 3), we shall write $r_{x_1, \dots, x_m}(d)$ instead of $f(x_1, \dots, x_m)$.

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 . For each weighting $\Theta = (\theta_1, \dots, \theta_m)$, we obtain (using the methodology of [FW97]) 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 .

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 “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 $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$, under the assumption that $\theta_1 \geq \dots \geq \theta_m$:

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

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 $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$, which we call the *weighting formula*, is the formula we adopt for use in our application.

It is shown in [FW97] that the weighting formula is well-defined, even when some of the θ_i 's are equal. For example, if $\theta_2 = \theta_3$, then should the second term of the weighting formula involve $f(x_1, x_2)$ or $f(x_1, x_3)$? The point is that it does not matter, since in either case the result is multiplied by $(\theta_2 - \theta_3)$, which is 0.

Although the weighting formula may look somewhat arbitrary, it is shown in [FW97] that it is actually uniquely determined, under one additional assumption that we now discuss. Two weightings are called *comonotonic* if they agree on the order of importance of the arguments. Formally, assume that Θ, Θ' are weightings (over the same index set). Then Θ, Θ' are *comonotonic* if there do *not* exist i, j with $\theta_i < \theta_j$ and $\theta'_j < \theta'_i$ both holding. For example, $(.2, .7, .1)$ and $(.3, .5, .2)$ are comonotonic because in both cases, the second entry is biggest, the first entry is next-biggest, and the third entry is smallest. It is clear that comonotonicity is reflexive and symmetric. Comonotonicity is *not* transitive, since for example $(0, 1)$ and $(1, 0)$ are not comonotonic, while $(0.5, 0.5)$ is comonotonic to both $(0, 1)$ and $(1, 0)$. 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 comonotonic, and $\alpha \in [0, 1]$.

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 $\theta_1, \dots, \theta_m$) [FW97]. Furthermore, the choice of the weighting formula (1) for $f_{(\theta_1, \dots, \theta_m)}(x_1, \dots, x_m)$ is the unique one that satisfies **D1**, **D2**, and **D3'** [FW97].

We now discuss local linearity. Intuitively, local linearity says that the aggregation function acts like a balance. Local linearity demands that the weighting that is the midpoint of two comonotonic weightings should produce a value that is the midpoint of the two values produced by the given weightings. In fact, local linearity extends beyond the midpoint to any weighting that is a convex combination of two comonotonic weightings: if a weighting is a convex combination of two comonotonic weightings, then local linearity demands that the associated value should be the same convex combination of the values associated with the given weightings. In other words, local linearity demands that (2) must hold when Θ and Θ' are comonotonic, and so agree on 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).

We note that it is shown in [FW97] that an assumption that (2) holds for *every* choice of Θ and Θ' , even those that are not comonotonic, would be incompatible with the properties **D1** and **D2**, unless the unweighted function f is of a very special form, namely, $f(x_1, \dots, x_k) = \frac{f(x_1) + \dots + f(x_k)}{k}$ for every k .

The weighting formula 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 the weighting formula 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

In this section, we consider how to implement the weighting formula in an IR engine, in order to weight the importance of search terms. From now on, we shall use the notation $r_{x_1, \dots, x_m}(d)$ to represent the relevance score assigned by the IR engine to the document d when the search terms are x_1, \dots, x_m . In the notation of the previous section, this is $f(x_1, \dots, x_m)$. Similarly, we use $r_{x_1, \dots, x_m}^{\theta_1, \dots, \theta_m}(d)$ to represent the relevance score that the weighting formula gives in the weighted case, where the search terms x_1, \dots, x_m are assigned the weights $\theta_1, \dots, \theta_m$, respectively. Thus, when $\theta_1 \geq \dots \geq \theta_m$, we have that $r_{x_1, \dots, x_m}^{\theta_1, \dots, \theta_m}(d)$ is given by

$$(\theta_1 - \theta_2) \cdot r_{x_1}(d) + 2 \cdot (\theta_2 - \theta_3) \cdot r_{x_1, x_2}(d) + 3 \cdot (\theta_3 - \theta_4) \cdot r_{x_1, x_2, x_3}(d) + \dots + m \cdot \theta_m \cdot r_{x_1, \dots, x_m}(d). \quad (3)$$

We shall hereafter refer to (3) as the weighting formula.

3.1 General Case

Assume that we are given an arbitrary IR engine that, when given search terms x_1, \dots, x_m , assigns a relevance score to each document d . By making use of the weighting formula, this IR engine

can be adapted to assign a relevance score even in the weighted case, where the search terms x_1, \dots, x_m are assigned the weights $\theta_1, \dots, \theta_m$, respectively, and where $\theta_1 \geq \dots \geq \theta_m$. Thus, the IR engine need only compute $r_{x_1}(d), r_{x_1, x_2}(d), \dots, r_{x_1, \dots, x_m}(d)$, and then take the convex combination of these values given by the weighting formula (3). Thus, by means of the weighting formula, the IR engine can be given the additional capability of allowing the search terms to be weighted. On the face of it, this procedure of passing to the weighted case requires m calls to the IR engine: once for the query with search term x_1 ; once for the query with search terms x_1, x_2 ; and so on. In the next two subsections, we discuss situations where the weighted case can be done much more efficiently: essentially as efficiently as in the unweighted case.

3.2 A Simple, Concrete Example

In this section, we consider a simple example of an IR engine, and show how to modify it to deal with the weighted case. We assume that the ranking formula in the unweighted case is derived from $tf \times idf$ in the vector space model. To keep the example especially simple, we assume that there is no query normalization [Sal89] (although this assumption is not really needed).

In practice, given search terms x_1, \dots, x_m , such a search engine does not compute $r_{x_1, \dots, x_m}(d)$ for every document d in the collection being searched. Instead, by an inverted index file, the IR engine accesses only the postings of those search terms that appear in the query, and updates an accumulator that stores the score of every document that shares at least one search term with the query⁵.

In such an IR engine, the accumulator can be updated using the following simple algorithm:

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 .

⁵Some heuristics such as partial list searching [Sal89] could also be used in which even less documents are considered. However, the explanation stated here still holds in that case.

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

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

Let us consider the weighted case, where the search terms x_1, \dots, x_m are assigned the weights $\theta_1, \dots, \theta_m$, respectively, and where $\theta_1 \geq \dots \geq \theta_m$. 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 the weighting formula.

Since we are assuming in this example that there is no query normalization, we can easily verify that for each j (with $1 \leq j \leq m$), we have

$$r_{x_1, \dots, x_j}(d) = r_{x_1}(d) + \dots + r_{x_j}(d). \quad (4)$$

Therefore, it follows simply from (3) that

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

Let us denote the coefficient of $r_{x_i}(d)$ above by α_i . Thus, if θ_{m+1} is artificially defined to be 0, then

$$\alpha_i = \sum_{j=i}^m j(\theta_j - \theta_{j+1}).$$

Hence, the accumulator algorithm stated above (which deals with the unweighted case) can be modified so as to take into account weighted terms by simply changing the computation of the acc formula as follows:

$$acc[d_{ik}] = acc[d_{ik}] + \alpha_i \cdot g(\rho_i(d_{ik}), x_i).$$

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 the less highly weighted x_i is. On the face of it, this is a reasonable heuristic. The weighting formula determines exactly what the values of these multipliers α_i should be.

3.3 Sufficient Conditions for Efficiency

There are many situations, much more general than that described in the previous subsection, 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 of the IR engine. This applies to IR engines such that:

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) $r_{x_1, \dots, x_j}(d)$, for each document d .
2. The value $r_{x_1, \dots, x_j}(d)$ is positive precisely if document d is relevant to at least one of x_1, \dots, x_j , and otherwise $r_{x_1, \dots, x_j}(d) = 0$.
3. Given search terms x_1, \dots, x_j , we are interested only in determining some subset of the documents d such that $r_{x_1, \dots, x_j}(d)$ 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 three conditions intuitively say that given the search terms x_1, \dots, x_j , the relevant documents d that we wish to determine all have $r_{x_1, \dots, x_j}(d)$ 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 weighting formula (3) that the set of documents d where $r_{x_1, \dots, x_m}^{\theta_1, \dots, \theta_m}(d)$ is positive is a subset of D . Therefore, the documents d that are relevant in the weighted case (where in particular $r_{x_1, \dots, x_m}^{\theta_1, \dots, \theta_m}(d)$ 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.

3.4 Using a Post-processor

We now discuss the implementation of a post-processor, that takes the results of an IR engine and reranks them to take into account the user-assigned weights (this is Scenario 2 in the introduction). There are two different sub-scenarios.

Sub-scenario 1: Using information from the IR engine. The first sub-scenario requires a modification of the IR engine, so that the IR engine returns not only the top documents and their scores using the set of search terms, but also the scores of those documents using each search term separately. Just as before, there is very little loss in efficiency, since the same entries in the

inverted index file are touched as in the unweighted case. In this sub-scenario, we could imagine that the IR engine has a “Rerank” button, that the user can press to impose weights and thereby rerank the results.

Sub-scenario 2: Using a search parasite. In the second sub-scenario, there is a search “parasite”, that takes the small set of documents that are a result of the unweighted query, and reranks them using its own ranking algorithm. Thus, under this approach, the scores obtained from the original IR engine are ignored: the input to the search parasite is simply the set of documents obtained from the original search, along with the set of search terms and the user-assigned weighting.

By using our weighting formula, we can convert any search parasite into one that takes into account user-assigned weightings. Since we have available to us a search parasite, namely Fetuccino [BSHJ⁺99], we implemented this approach. We describe this implementation in more detail in the next section.

4 Integration in a Search Parasite for Dynamic Reranking

We integrated the “dynamic reranking” feature described above, in an experimental version of Fetuccino [BSHJ⁺99]. Fetuccino is a search parasite that validates and augments existing Web search services results (by local directed crawling à la WebGlimpse [MSG97]), and finally visualizes them in a Java applet. Its standard version (without the rerank feature) is available as a free service on the IBM Corporate Java page⁶, as well as a free stand-alone application on the IBM Alphaworks site⁷. The integration was done by simply modifying the similarity measure computation in the relevance ranking component, using the weighting formula, and by adding a customized view in the result visualization applet. In the example below, the user selected Excite as the primary IR engine, entered the query “tax treaty between US and France” (without assigning any weighting yet) and Fetuccino returned the results shown in Fig. 3.

Once the user clicks on the “Add-Ons/Rerank” function, she needs to assign weights to each indexing unit of the query (Fig. 4). In the case of Fetuccino, not only are single terms considered, but also “lexical affinities”, that is, binary word correlations (that were also used in the original ranking in Fig. 3), since the Fetuccino internal IR engine is provided by the Guru IR system [MS89], [Maa91], that supports this original indexing scheme. This demonstrates the flexibility of the weighting formula. It can be applied not only to single terms, but to any kind of indexing unit.

After clicking on “Rerank”, the default view is changed to a “reranked view” (Fig. 5), where the scoring of each document has been changed to the new score computed dynamically according to the weighting formula (of course, we normalize the weights so that they add to 1). For the

⁶Free service available at <http://www.ibm.com/java/fetuccino>.

⁷Stand-alone application downloadable from <http://www.alphaworks.ibm.com>, choose Mapuccino/Fetuccino technology download.

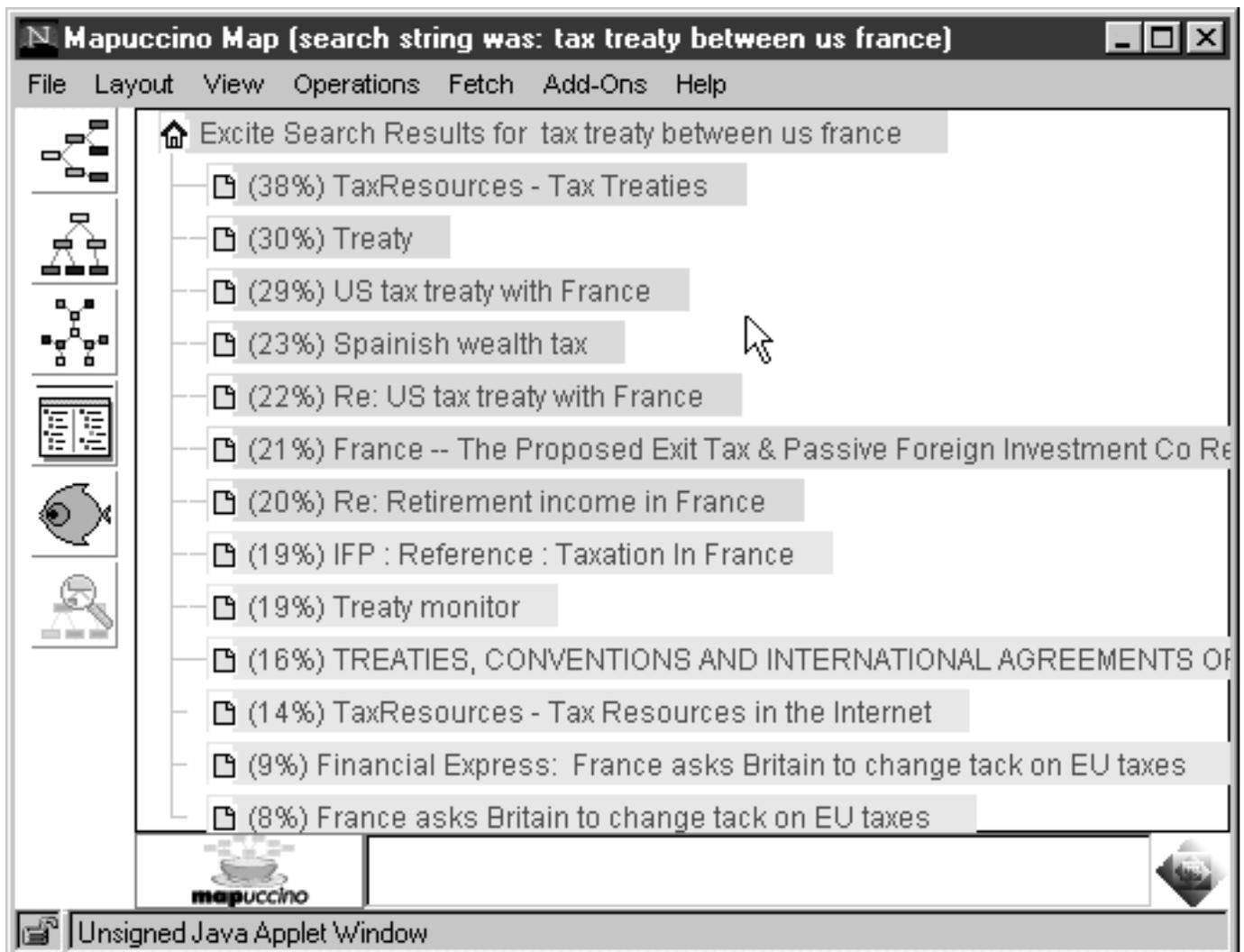


Figure 3: Fetuccino results for unweighted query

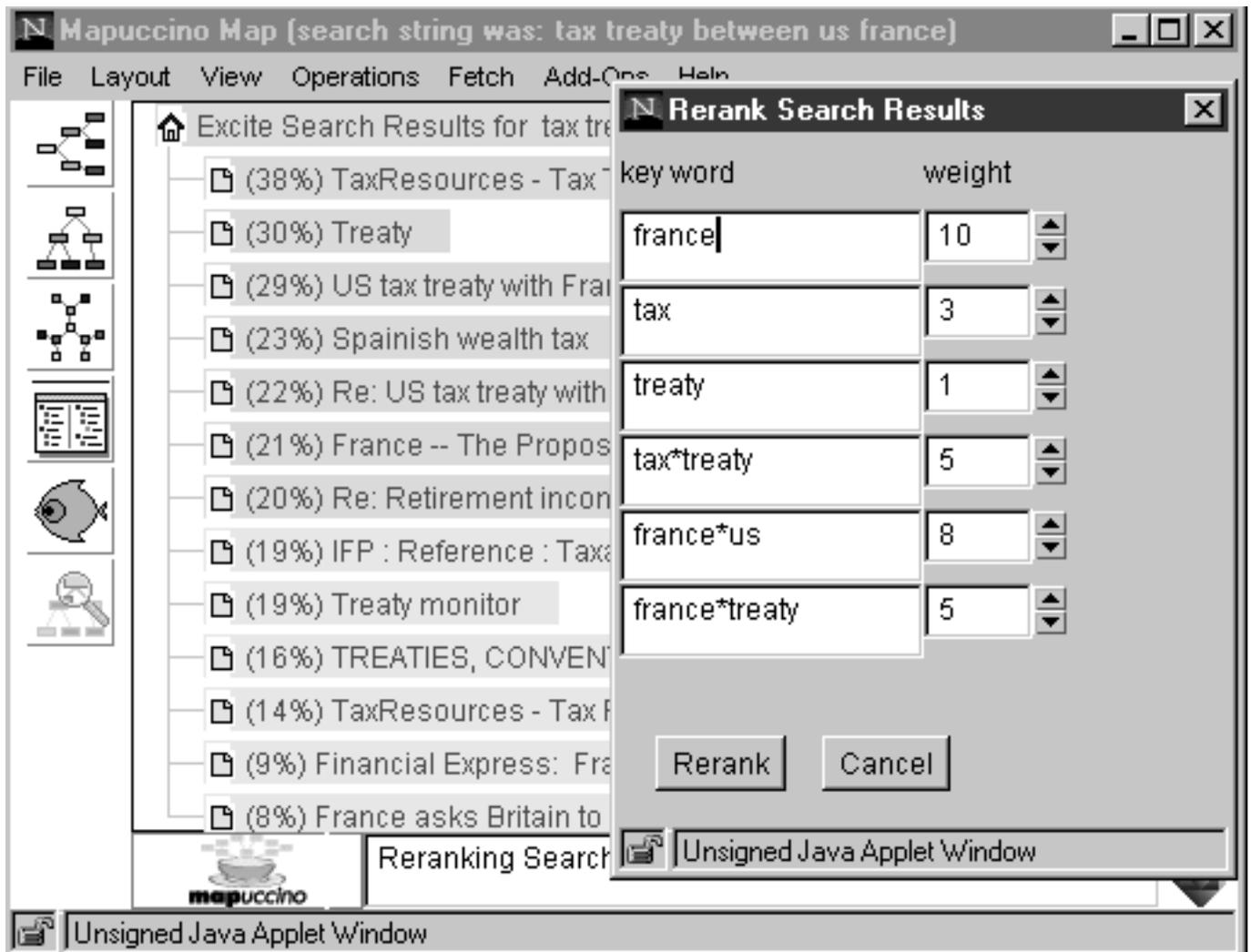


Figure 4: Assigning weights to query terms

sake of the comparison, this view does not reorder the documents, but rather leaves them in the descending order shown in Fig. 3. The changes of the relevance score of each result is reflected in three ways: (a) the score itself, which prefaces the document title, (b) the shade of blue, which reflects the score (the darker, the more relevant), and (c) the rank in the list according to the score appended to the title. Thus, the top candidate had its score changed from 38% to 7%, and its rank moved to 7, while the 3rd candidate was boosted to 1st place. Note that, interestingly enough, the previous 4th candidate (“Spanish wealth tax”) is rightly boosted to 2nd place, because in spite of its title, its contents mainly refers to the tax treaty with France. The previous second candidate that was highly ranked is “demoted”, since it does not refer either to France or to tax treaties, but to general U.S. treaties.



Figure 5: Fetuccino results to a weighted query

The examples provided constitute in no way an evaluation of the effectiveness of the weighting

formula, but rather an example of its usage in an existing system. Indeed, we do not propose here yet another ranking mechanism, but rather, assuming that such a ranking mechanism is given, we give a principled formula for integrating user-assigned weights into this mechanism, which is the unique possible such formula under certain assumptions. Therefore, evaluation based on recall and precision is out of the scope of this paper. However, evaluating such results would definitely be interesting, in the context of the TREC forum for instance, to decide not on the correctness of the weighting formula itself, but rather whether weighting search terms at all is beneficial to the search process.

5 Allowing + and -

Nearly every major Web search service allows a search term to be prefaced by the plus symbol (+), which means that the search term must appear in the document, or by the minus symbol (-), which means that the search term must not appear in the document. See for example the help pages of AltaVista⁸, Excite⁹, Lycos¹⁰ Infoseek¹¹, etc. Thus, the plus and minus symbols can be viewed as filtering functions, that cause the results of a search to be filtered by search terms that must be present or absent from the answer [BYRN99]. In some search services, such as Google¹², the plus symbol has no effect, since each search term is already required to be present in documents that match the query.

What should the meaning of plus and minus symbols be when weightings are allowed? For example, let $+x$ be a search term, prefaced by a plus symbol, that is given a very low user-assigned weight. There seems to be a conflict, since the plus symbol attaches great importance to the term, whereas the low weight attaches low importance to the term.

Our methodology can be applied directly to the case where plus and minus symbols are allowed. All we need to assume is that in the unweighted case, a score is still assigned to every document, even when some of the search terms may be prefaced by a plus or a minus symbol. Of course, in the unweighted case, because of the semantics of the plus and minus symbols, the score for a document should be zero if it does not contain a search term that is prefaced by a plus symbol, or if it does contain a search term that is prefaced by a minus symbol.

Just as before, our weighting formula gives the unique solution that satisfies our desiderata in Section 2. For example, assume that the search terms are $+x_1, +x_2, -x_3$, with weights $\theta_1, \theta_2, \theta_3$, respectively, where $\theta_1 \geq \theta_2 \geq \theta_3$ and $\theta_1 + \theta_2 + \theta_3 = 1$. Then the weighting formula tells us that $r_{+x_1, +x_2, -x_3}^{\theta_1, \theta_2, \theta_3}(d)$ should be taken to be

$$(\theta_1 - \theta_2) \cdot r_{+x_1}(d) + 2 \cdot (\theta_2 - \theta_3) \cdot r_{+x_1, +x_2}(d) + 3 \cdot \theta_3 \cdot r_{+x_1, +x_2, -x_3}(d). \quad (5)$$

⁸http://doc.altavista.com/help/search/search_help.shtml

⁹<http://www.excite.com/Info/searching2.html>

¹⁰<http://www.lycos.com/help/search-help.html#exclude>

¹¹http://infoseek.go.com/Help/help.html?key=HELP_T00008_SRCHQUIK

¹²<http://www.google.com>

Note that if θ_1 is strictly greater than θ_2 , then by applying (5) we might get a high score for a document d that does not contain the search term x_2 (because $r_{+x_1}(d)$ might be large). This is in spite of the fact that x_2 is prefaced by a plus symbol. This is probably as it should be, since the fact that $+x_2$ does not have the highest weight somewhat “overrides” the fact that x_2 is prefaced by a plus symbol. Similarly, by applying (5) we might get a high score for a document d that contains the search term x_3 , in spite of the fact that x_3 is prefaced by a minus symbol. Intuitively, this corresponds to the fact that we are not giving a large weight to $-x_3$.

It is instructive to see why our desiderata in Section 2 imply that the semantics of the plus symbol must change in the weighted case, in the sense that a plus symbol does not force a term to appear. That is, there may be a document with a positive score, and in fact even a very high score, even though there is some search term $+x$ where x does not appear in the document (this search term has a low weight).

Let us assume that the search terms are x_1 and $+x_2$. Let d be a document where x_2 does not appear (x_2 plays the role of x in our discussion above), but whose score is very high when x_1 is the only search term (the latter statement means that the unweighted score $r_{x_1}(d)$ is very high). By our continuity condition **D3** of Section 2, it follows that as θ converges to 0, we have that $r_{x_1,+x_2}^{1-\theta,\theta}(d)$ must converge to $r_{x_1,+x_2}^{1,0}(d)$, which by condition **D2** must equal $r_{x_1}^1(d)$, which by condition **D1** (with $m = 1$) must equal $r_{x_1}(d)$. In particular, there is some positive ϵ such that $r_{x_1,+x_2}^{1-\epsilon,\epsilon}(d)$ is very high (as close as we want to the very high score $r_{x_1}(d)$). Thus, $r_{x_1,+x_2}^{1-\epsilon,\epsilon}(d)$ is very high, even though the term x_2 does not appear in document d . So indeed, the semantics of the plus symbol must change in the weighted case, in the sense that there may be a document with a very high score, even though there is some search term $+x$ where x does not appear in the document.

A problematic case occurs when some search term that is prefaced by a minus symbol gets strictly the highest weight. For example, assume that the search terms are $-x_1, x_2$, with weights θ_1, θ_2 , respectively, where $\theta_1 > \theta_2$ and $\theta_1 + \theta_2 = 1$. Then the weighting formula tells us that $r_{-x_1,x_2}^{\theta_1,\theta_2}(d)$ is taken to be

$$(\theta_1 - \theta_2) \cdot r_{-x_1}(d) + 2 \cdot \theta_2 \cdot r_{-x_1,x_2}(d). \quad (6)$$

But in the evaluation of $r_{-x_1}(d)$ in (6), we are in a situation where there is only one search term, and it is prefaced by a minus symbol. This is a peculiar search, in which, intuitively, all we care about is that the term x_1 not appear. This is analogous to “unsafe queries” [Zan86, RBS87] in a database context, which, intuitively, can produce through the use of negation an infinite set of answers.¹³ It is amusing to try a search against major Web search services with only one search term, and with that search term prefaced by a minus symbol, and to see the results that are produced. AltaVista seems to follow the literal meaning of a minus symbol: for example, the query “-dog” seems to return every Web page that does not contain the word “dog”. Somewhat mysteriously, if we give AltaVista a query such as “-wildeoikldiejf”, where wildeoikldiejf is a nonsense word that presumably does not appear in any Web page¹⁴, then the number of documents returned varies by one percent or so, depending on which nonsense word

¹³An example is a query that asks for the name of every person not listed in the database.

¹⁴except for those containing this document!

appears after the minus symbol. (Note, incidentally, that this number of Web pages returned seems to give us a quick estimate of the number of Web pages indexed by AltaVista.) However, when there is only one search term, and that search term is prefaced by a minus symbol, most major Web search services simply say that there is no match: apparently, they treat such a search as illegal.

6 Summary of contributions

IR engines typically weight the importance of search terms (such as by using $tf \times idf$ scores, where less common terms are given higher weight). We consider here the scenario where a user can express her own subjective weighting of the importance of the search terms on top of the system-generated weighting, and show how this should modify the relevance scores of documents. This has been allowed before, but only by ad hoc heuristics. We give the first principled method for taking into account the user's subjective weighting of the importance of search terms. Our method is based on a formula that has the key advantages of being very simple and of being uniquely determined under certain constraints that seem intuitively desirable.

We now consider what our contributions have been. First, we have introduced to the information retrieval community the formula in [FW97], 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 [FW97] can be applied whenever there is an underlying scoring 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, we have simplified the notation of [FW97].¹⁵ We show how to implement the weighting formula efficiently in our application; this is important because a naive application of the weighting formula would require issuing as many distinct queries as there are search terms. Finally, we exemplify the usage of the weighting formula in the Web, by showing how it was integrated in the Fetuccino search mapping system.

7 Acknowledgments

We thank David Carmel, Byron Dom, David Notkin and Frank Smadja for their valuable comments and feedback on previous versions of this paper. Furthermore, we are grateful to Yael Dvir and Miki Herscovici for integrating the weighting formula in Fetuccino in record time. Finally, we thank David Carmel for discussions about the role of $+$ and $-$ that led to Section 5.

¹⁵Subsequently to our work, Fagin and Wimmers have incorporated our extensions and simplifications into a revised version [FW99] of their paper.

References

- [BCCN95] J. Broglio, J. P. Callan, W. B. Croft, and D. W. Nachbar. Document retrieval and routing using the INQUERY system. In *Proceedings of Third Text Retrieval Conference (TREC-3)*, pages 29–38, 1995.
- [BSHJ⁺99] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, V. Soroka, and S. Ur. Adding support for dynamic and focused search with Fetuccino. In *Proceedings of the WWW8 Conference*, Toronto, CA, May 1999.
- [BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.
- [FW97] 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.
- [FW99] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. (This is the journal version of “Incorporating user preferences in multimedia queries”.) *Theoretical Computer Science*, to appear, 1999.
- [Har92] 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.
- [Maa91] Y.S. Maarek. Software library construction from an IR perspective. *SIGIR Forum*, 25(2):8–18, Fall 1991.
- [MS89] 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.
- [MSG97] U. Manber, M. Smith, and B. Gopal. Webglimpse: Combining browsing and searching. In *Proceedings of the Usenix Technical Conference*, Los Angeles, CA, 1997.
- [RBS87] R. Ramakrishnan, F. Bancilhon, and A. Silberschatz. Safety of recursive Horn clauses with infinite relations. In *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, pages 328–339, 1987.
- [Sal89] G. Salton. *Automatic text processing, the transformation, analysis and retrieval of information by computer*. Addison-Wesley, Reading, MA, 1989.
- [SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. Computer Series. McGraw-Hill, New York, 1983.
- [Zan86] C. Zaniolo. Safety and compilation of nonrecursive Horn clauses. In *Proceedings of First International Conference on Expert Database Systems*, pages 167–178, 1986.