MARS: MULTIPLICATIVE ADAPTIVE REFINEMENT WEB SEARCH

Xiannong Meng
Department of Computer Science
Bucknell University
Lewisburg, PA 17837, USA
Email: xmeng@bucknell.edu

Zhixiang Chen
Department of Computer Science
University of Texas -- Pan American
Edinburg, TX 78539, USA
Email: chen@cs.panam.edu

ABSTRACT

This chapter reports the project MARS (Multiplicative Adaptive Refinement Search), which applies a new multiplicative adaptive algorithm for user preference retrieval to Web search. The new algorithm uses a multiplicative query expansion strategy to adaptively improve and reformulate the query vector to learn users’ information preference. The algorithm has provable better performance than the popular Rocchio's similarity-based relevance feedback algorithm in learning a user preference that is determined by a linear classifier with a small number of non-zero coefficients over the real-valued vector space. A meta-search engine based on the aforementioned algorithm is built, and analysis of its search performance is presented.

Keywords: Web Applications, Data Mining, Information Search and Retrieval, Relevance Feedback
INTRODUCTION

Vector space models and relevance feedback have long been used in information retrieval (Salton, 1989; Baeza-Yates & Ribeiro-Neto, 1999). In the $n$-dimensional vector space model, a collection of $n$ index terms or keywords is chosen, and any document $d$ is represented by an $n$-dimensional vector $d = (d_1, \ldots, d_n)$, where $d_i$ represents the relevance value of the $i$-th index term in the document. Let $D$ be a collection of documents, $R$ be the set of all real values, and $R^+$ be the set of all positive real values. It has been shown in (Bollmann & Wong, 1987) that if a user preference relation $\prec$ is a weak order satisfying some additional conditions then it can be represented by a linear classifier. That is, there is a query vector $q = (q_1, \ldots, q_n) \in R^n$ such that

$$\forall d, d' \in D, d \prec d' \Leftrightarrow q \cdot d < q \cdot d'.$$  \hspace{1cm} (1)

Here, “$\cdot$” denotes the inner product of vectors. In general, a linear classifier over the vector space $[0,1]^n$ is a pair of $(q, \theta)$ which classifies any document $d$ as relevant if $q \cdot d > \theta$, or irrelevant otherwise, where the query vector $q \in R^n$, the classification threshold $\theta \in R^+$, and $[0,1]$ denote the set of all real values between 0 and 1. Recall that $q \cdot d$ is usually used as the relevance rank (or score) of the document $d$ with respect to user preference.

Let $D_r$ be the set of all relevant documents in $D$ with respect to a user's information needs (or search query). Assume that a user preference relation has a simple structure with only two levels, one level consisting of all relevant documents and the other consisting of all irrelevant documents, and within the same level no preference is given between any two documents. Then,
finding a user preference relation satisfying the expression (1) is equivalent to the problem of finding a linear classifier \((q, \theta)\) over \([0,1]^n\) with the property

\[
\forall d \in D, d' \in D_r \iff q \cdot d > \theta,
\]

(2)

where \(q \in R^n\) is the query (or weight) vector.

The goal of relevance feedback in information retrieval is to identify a user preference relation \(<\) with respect to his/her information needs from documents judged by that user. Since user preference relations vary from users and may have various unknown representations, it is not easy for an information system to learn such relations. The existing popular relevance feedback algorithms basically use linear additive query expansion methods to learn a user preference relation as follows:

- Start with an initial query vector \(q_0\).
- At any step \(k \geq 0\), improve the \(k\)-th query vector \(q_k\) to

\[
q_{k+1} = q_k + \alpha_1 d_1 + \ldots + \alpha_s d_s
\]

(3)

where \(d_1, \ldots, d_s\) are the documents judged by the user at this step, and the updating factors \(\alpha_i \in R\) for \(i = 1, \ldots, s\).

One particular and well-known example of relevance feedback is Rocchio's similarity-based relevance feedback (Rocchio, 1971). Depending on how updating factors are used in
improving the $k$-th query vector as in expression (3), a variety of relevance feedback algorithms have been designed (Salton, 1989). A similarity-based relevance feedback algorithm is essentially an adaptive supervised learning algorithm from examples (Salton & Buckley, 1990; Chen & Zhu, 2000; Chen & Zhu 2002). The goal of the algorithm is to learn some unknown classifier (such as the linear classifier in expression (1)) that is determined by a user's information needs to classify documents as relevant or irrelevant. The learning is performed by means of modifying or updating the query vector that serves as the hypothetical representation of the collection of all relevant documents. The technique for updating the query vector is linear addition of the vectors of documents judged by the user. This type of linear additive query updating technique is similar to what used by the Perceptron algorithm (Rosenblatt, 1958). The linear additive query updating technique has a disadvantage: itsconverging rate to the unknown target classifier is slow (Chen & Zhu, 2000; Chen & Zhu, 2002; Kivinen et al., 1997). In the real world of Web search, a huge number of terms (usually, keywords) are used to index Web documents. To make the things even worse, no users will have the patience to try, say, more than 10 iterations of relevance feedback in order to gain some significant search precision increase. This implies that the traditional linear additive query updating method may be too slow to be applicable to Web search, and this motivates the authors to design new and faster query updating methods for user preference retrieval.

THE MULTIPLICATIVE ADAPTIVE QUERY EXPANSION ALGORITHM

In this section, a multiplicative query updating technique is designed to identify a user preference relation satisfying expression (1) (Chen, 2001). The authors believe that linear
additive query updating yields some *mild* improvement on the hypothetical query vector towards the target user preference. One wants a query updating technique that can yield *dramatic* improvements so that the hypothetical query vector can be moved towards the target in a much faster pace. The idea is that when an index term is judged by the user, its corresponding value in the hypothetical query vector should be boosted by a multiplicative factor that is dependent on the value of the term itself. If a document is judged as relevant, its terms are promoted by a factor. If a document is judged as irrelevant, its terms are demoted by a factor. The algorithm is described in Figure 1.

**Algorithm MA**($q_0$, $f$, $\theta$):

(i) Inputs:
- $q_0$: the non-negative initial query vector
- $f(x)$: $[0,1] \rightarrow \mathbb{R}^+$, the updating function
- $\theta \geq 0$, the classification threshold

(ii) Set $k = 0$.

(iii) Classify and rank documents with the linear classifier $(q_k, \theta)$.

(iv) While (the user judged the relevance of a document $d$)

```
for (i = 1, ..., n)
{
    /* $q_k = (q_{1,k}, ..., q_{n,k}), d = (d_1, ..., d_n)$ */
    if (d_i \neq 0)
    {
        /* adjustment */
        if (q_i,k \neq 0) set $q_{i,k+1} = q_{i,k}$ else set $q_{i,k+1} = 1$
        if (d is relevant) /* promotion */
            set $q_{i,k+1} = (1 + f(d_i)) q_{i,k+1}$
        else /* demotion */
            set $q_{i,k+1} = q_{i,k+1} / (1 + f(d_i))$
    } /* end of if */
    else /* d_i == 0 */
        set $q_{i,k+1} = q_{i,k}$
} /* end of for */
} /* end of while */
```

(v) If the user has not judged any document in the $k$-th step, then stop. Otherwise, let $k = k + 1$ and go to step (iv).

**Figure 1. Algorithm MA** (Multiplicative Adaptive Query Expansion Algorithm)
In this chapter, only non-decreasing updating functions $f(x) : [0,1] \rightarrow \mathbb{R}^+$ are considered, because one wants the multiplicative updating for an index term to be proportional to the value of the term. The following two examples of algorithm MA are of particular interest.

**Algorithm LMA:** *In this algorithm, the updating function in algorithm MA is set to be $f(x) = \alpha x$, a linear function with a positive coefficient $\alpha > 1$.***

**Algorithm ENL:** *In this algorithm the updating function in algorithm MA is set to be $f(x) = \alpha^x$, an exponential function with $\alpha > 1$.***

The design of algorithm MA is enlightened by algorithm Winnow (Littlestone, 1988), a well-known algorithm equipped with a multiplicative weight updating technique. However, algorithm MA generalizes algorithm Winnow in the following aspects: (1) various updating functions may be used in MA, while only constant updating functions are used in Winnow; (2) multiplicative updating for a weight is dependent on the value of the corresponding indexing terms, which is more realistic and applicable to real-valued vector space, while Winnow considers all the terms equally; and (3) finally, a number of documents which may or may not be counterexamples to the algorithm’s current classification are allowed; while Winnow is an adaptive learning algorithm from equivalence queries, requiring the user to provide a counterexample to its current hypothesis. The equivalence query model is hardly realistic, because a user in reality has no knowledge about the information system or about the
representation of his/her preference. What the user may do, and is able to do, is that the user can judge some documents as what the user needs or not among those provided by the system.

Algorithm Winnow (Littlestone, 1988) and algorithm TW2 (Chen et al. 2002) can be derived from algorithm MA as follows.

**Algorithm Winnow:** Algorithm MA becomes algorithm Winnow when the following restrictions are applied:

- The vector space is set to the binary vector space \(\{0,1\}^n\).
- The initial query vector is set to \(q_0 = (1,\ldots,1)\).
- The updating function is chosen as \(f(x) = \alpha\), a positive constant function.
- At step (iv), equivalence query is adopted. That is, the user is asked to judge at most one document that is a counterexample to the current classification of the algorithm.

**Algorithm TW2:** Algorithm MA becomes algorithm TW2 when the following restrictions are applied:

- The vector space is set to the binary vector space \(\{0,1\}^n\).
- The initial query vector is set to \(q_0 = (0,\ldots,0)\).
- The updating function is chosen as \(f(x) = \alpha\), a positive constant function.

The performance of algorithm MA is now analyzed when it is used to identify a user preference satisfying expression (2), a linear classifier \((q, 0)\). Here the case when the threshold \(\theta = 0\) is considered. The algorithm is said to make a classification error at step \(k\) when the user
judged a document as a counterexample to the algorithm’s current hypothesis. The total number of classification errors that algorithm MA will make can be estimated based on the worst-case analysis. Also, at most one counterexample is provided to the algorithm at each step. From now on to the end of the section, it is assumed that \( \mathbf{q} \) is a non-negative query vector with \( m \) non-zero components and \( \theta > 0 \). Define

\[ \beta = \min \{q_i \mid q_i > 0, \ 1 \leq i \leq n\}. \]

**Definition:** Documents in the collection \( D \) are indexed with respect to a threshold \( \delta \), \( 0 < \delta \leq 1 \), if for any document \( \mathbf{d} = (d_1, \ldots, d_n) \in D \), either \( d_i = 0 \) or \( \delta \leq d_i, \ 1 \leq i \leq n \).

In other words, when a document is indexed with respect to a threshold \( \delta \), any index term with a value below the threshold \( \delta \) is considered not significant, and hence is set to zero. Recall that in the vector space model a document and its vector have the equivalent meaning, so one may not distinguish the two concepts.

**Lemma:** Assume that documents are indexed with respect to a threshold \( \delta \). Let \( u \) denote the total number of promotions algorithm MA needs to find the linear classifier \((\mathbf{q}, 0)\). Let \( m \) denote the number of non-zero components in \( \mathbf{q} \). Then,

\[ u \leq \frac{m \log \frac{\theta}{\beta \delta}}{\log(1 + f(\delta))} \]

**Proof:** Without loss of generality, it is further assumed that the \( m \) non-zero components of \( \mathbf{q} \) are \( q_i, \ldots, q_m \). When a promotion occurs at step \( k \), a relevant document \( \mathbf{d} \) is given to the
algorithm as a counterexample to its classification. Because the document is indexed with respect to threshold $\delta$, there is some $i$ with $1 \leq i \leq m$ such that $\delta \leq d_i$. This means that the $i$-th component $q_{i,k}$ of the query vector $q_k$ will be promoted to

$$q_{i,k+1} = (1 + f(d_i)) q_{i,k} > (1 + f(\delta)) q_{i,k}$$

(4)

because $f$ is non-decreasing. Since $q_{i,k}$ will never be demoted, it follows from expression (4) that $q_{i,k}$ can be promoted at most

$$\frac{\log \frac{\theta}{\beta \delta}}{\log(1 + f(\delta))}$$

(5)
times. Since each promotion yields a promotion for at least one $q_{i,k}$ for $1 \leq i \leq m$, the total number of promotions $u$ is at most $m$ times the value given in expression (5).

\[ \Box \]

**Theorem:** Assume that documents are indexed with respect to a threshold $\delta$. Let $T$ denote the total number of classification errors that algorithm MA makes in order to find the linear classifier $(q, 0)$ over the real-valued vector space $[0,1]^n$. Let $m$ denote the number of non-zero components in $q$. Then,

$$T \leq \left[ \frac{(1 + f(1))(n-m) + \sigma}{f(\delta)\theta} \right] \frac{1}{(1 + f(\delta))(1 + \delta)} + \frac{m \log \frac{\theta}{\beta \delta}}{\log(1 + f(\delta))} + 1$$

Where $\sigma$ is the sum of the initial weights. (Hence, if $\theta = \frac{n}{k}$ is chosen, $T = O(k \log n)$.)

**Proof.** Without loss of generality, assume that the $m$ non-zero components of $q$ are $q_1, ..., q_m$. The sum of the weights is estimated as $\sum_{i=1}^{n} q_{i,k}$. Let $u$ and $v$ be the number of promotion steps and the number of demotion steps occurred during the learning process, respectively. Let $t_k$
denote the number of zero components in \( q_k \) at step \( k \). Note that once a component of \( q_k \) is promoted to a non-zero value, it will never become zero again. For a promotion at step \( k \) with respect to a relevant document \( d \) judged by the user, for \( i = 1, \ldots, n \), the following relation can be established

\[
q_{i,k+1} = \begin{cases} 
q_{i,k}, & \text{if } d_i = 0, \\
(1 + f(d_i)), & \text{if } d_i \neq 0 \text{ and } q_{i,k} = 0, \\
(1 + f(d_i))q_{i,k}, & \text{if } d_i \neq 0 \text{ and } q_{i,k} \neq 0.
\end{cases}
\]

Since a promotion only occurs when

\[
q_k \cdot d = \sum_{i=1}^{n} d_i q_{i,k} = \sum_{d_i \neq 0 \text{ and } q_{i,k} \neq 0} q_{i,k} < \theta,
\]

the following derivation can be carried out.

\[
\sum_{i=1}^{n} q_{i,k+1} = \sum_{d_i \neq 0 \text{ and } q_{i,k} = 0} q_{i,k+1} + \sum_{d_i \neq 0 \text{ and } q_{i,k} \neq 0} q_{i,k+1} + \sum_{d_i = 0} q_{i,k+1} \\
= \sum_{d_i \neq 0 \text{ and } q_{i,k} = 0} (1 + f(d_i)) + \sum_{d_i \neq 0 \text{ and } q_{i,k} \neq 0} (1 + f(d_i))q_{i,k} + \sum_{d_i = 0} q_{i,k} \\
\leq (1 + f(1))q_k + \frac{1 + f(1)}{\delta} \sum_{d_i \neq 0 \text{ and } q_{i,k} = 0} \delta q_{i,k} + \sum_{i=1}^{n} q_{i,k} \\
\leq (1 + f(1))q_k + \frac{1 + f(1)}{\delta} \sum_{d_i \neq 0 \text{ and } q_{i,k} = 0} d_i q_{i,k} + \sum_{i=1}^{n} q_{i,k} \\
\leq (1 + f(1))q_k + \frac{1 + f(1)}{\delta} \theta + \sum_{i=1}^{n} q_{i,k}.
\]

For a demotion at step \( k \) with respect to an irrelevant document \( d \) judged by the user, for \( i = 1, \ldots, n \), it is true that

\[
q_{i,k+1} = q_{i,k} - \left(1 - \frac{1}{1 + f(d_i)}\right)q_{i,k} \leq q_{i,k} - (1 - \frac{1}{1 + f(\delta)})q_{i,k}
\]

Since a demotion occurs only when \( \sum_{i=1}^{n} d_i q_{i,k} > \theta \), it can be seen that
\[
\sum_{i=1}^{n} q_{i,k+1} \leq \sum_{i=1}^{n} q_{i,k} - \left(1 - \frac{1}{1 + f(\delta)}\right) \sum_{i=1}^{n} q_{i,k}
\]
\[
\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{1 + f(\delta)} \sum_{i=1}^{n} d_i q_{i,k}
\]
\[
\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1 + f(\delta))(1 + \delta)} \sum_{i=1}^{n} d_i q_{i,k}
\]
\[
\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1 + f(\delta))(1 + \delta)} \theta.
\]

Let the sum of the initial weights to be \(\sigma\). Hence, after \(u\) promotions and \(v\) demotions,
\[
\sum_{i=1}^{n} q_{i,k+1} \leq (1 + f(1)) \sum_{i=1}^{u} t_i + \sum_{i=1}^{n} q_{i,0} + \frac{(1 + f(1))\theta}{\delta} u - \frac{f(\delta)\theta}{(1 + f(\delta))(1 + \delta)} v
\]
\[
\leq (1 + f(1))(n - m) + \sigma + \frac{(1 + f(1))\theta}{\delta} u - \frac{f(\delta)\theta}{(1 + f(\delta))(1 + \delta)} v
\]

Note that at any step the weights are never negative. It follows from the above relation that
\[
v \leq \frac{[(1 + f(1))(n - m) + \sigma] f(\delta)}{f(\delta)\theta} + \frac{(1 + f(1))(1 + f(\delta)) u}{f(\delta)\theta} + u.
\]

It follows from the Lemma, expressions (6), (7) and (8) that the total number of promotions and demotions, i.e. the total number of classification errors \(T\), is bounded by
\[
T \leq u + v \leq \frac{[(1 + f(1))(n - m) + \sigma] f(\delta)}{f(\delta)\theta} + \frac{(1 + f(1))(1 + f(\delta)) u}{f(\delta)\theta} + u
\]
\[
\leq \frac{[(1 + f(1))(n - m) + \sigma] f(\delta)(1 + \delta)}{f(\delta)\theta} + \frac{(1 + f(1))(1 + f(\delta)) (1 + \delta)}{f(\delta)\theta} + 1 \frac{m\log \frac{\theta}{\beta\delta}}{\log(1 + f(\delta))}
\]

This completes our proof. \(\Box\)

**THE META-SEARCH ENGINE MARS**

This section reports the experimental meta-search engine MARS (Multiplicative Adaptive Refinement Search) that has been built using the algorithm MA to actually test the
effectiveness and efficiency of the algorithm. MARS can be accessed from the URL specified at the end of the chapter. Figure 2 shows a general architecture of the meta-search engine MARS.

![Figure 2. Architecture of MARS](image)

User queries to MARS are accepted from a Web browser. Besides entering the query, a user can also specify a particular general-purpose search engine he/she would like MARS to use and the maximum number of returned results (the larger the number is, the more time it takes to process). The QueryConstructor organizes the query into a format conforming to the specified search engine. One of the MetaSearchers sends the query to the general-purpose search engine. When the results are sent back from the general-purpose search engine, DocumentParser, DocumentIndexer and Ranker process the returned URLs and list them to the user as the initial
search result. At this point, the rank is based on the original rank from the search engine.

Constrained by the amount of space available on a typical screen, only the top 10 URLs (highest ranked) and the bottom 10 URLs (lowest ranked) are listed. Once the results are displayed, the user can interactively work with MARS to refine the search results. Each time the user can mark a number of particular URLs as relevant or not relevant. Upon receiving feedback from the user, MARS updates the weight assigned to each index term within the set of documents already returned from the specified search engine, according to the algorithm MA. If a document is marked as relevant, the weights of its index terms are promoted. If marked irrelevant, they are demoted. The refined results are sorted based on the ranking scores and then displayed back to the user for further relevance feedback. This process continues until the satisfactory results are found or the user quits the search.

Figure 3. Interface of MARS
Figure 3 shows the user interface of MARS. Figure 4 shows the initial results returned by a general-purpose search engine with the search key-word being “mars”. Figure 5 shows the results after the user feedback has been processed by MARS.

Figure 4. Initial Results Returned By General-purpose Search Engine
EMPIRICAL PERFORMANCE ANALYSIS

Experiment Setting

The experiments were conducted in the summer of 2002. A collection of 72 random queries were sent to a general-purpose search engine (AltaVista was used in this study). Each of these queries resulted in a list of documents returned by the search engine. The number of returned documents was set to be 200, 150, 100, and 50, respectively. For each of the returned set of documents, the authors used the MARS meta-search engine that utilizes algorithm MA to interactively refine the search results. The returned documents from AltaVista would be marked as relevant or not relevant. The marked results were sent to MARS that would promote or demote index terms of each documents based on the feedback. The refined results were
displayed to the user for possibly more feedback. For each query the process typically involved two to three rounds of feedback, until a satisfactory set of results were found.

At the time, the MARS meta-search engine was running on a Sun Ultra-10 workstation with 256 mega-bytes of memory. The code was written in a combination of C and C++, and the executables were generated by the GNU g++ compiler. The data collection process lasted about one month.

Statistics Collected

Three types of performance measures were collected and studied. The first is the precision-recall statistics. The information retrieval standard measurements of performance are precision and recall. The precision is defined as the ratio between the number of relevant documents returned and the total number of documents returned. The recall is defined as the ratio between the number of relevant documents returned and the total number of relevant documents. In many applied information retrieval systems such as the Web, such statistics as the total number of documents and the total number of relevant documents are not available. Two alternate measures are defined in this study to approximate the standard measurements. The set of documents returned by a search engine is defined as $A$. Then $|A|$ denotes the number of total returned documents. Assume a set of $R$ documents in $A$ is relevant to the search query judged by the user(s). For a given constant $m$ (the number of top returned documents), define $R_m$ to be the set of relevant documents among the top-$m$ returned documents. Then the relative recall $R_r$ and relative precision $P_r$ are defined as follows.

$$R_r = \frac{|R_m|}{|R|}, \quad P_r = \frac{|R_m|}{m}.$$
These two measures are used to assess the performance of MARS and compare them to that of AltaVista. The second performance measure is the relative placements of the relevant results. In standard precision-recall measurements, the precision collectively measures how many relevant results have been returned. It doesn’t reflect the placement of individual results clearly. For example, if five out of the top-10 documents are relevant, at the 10-th document, the precision is 50%. It doesn’t show where these five relevant documents are, which could be placed as 1 through 5, or be placed as 6 through 10. Unless going back to each individual result, one would not know the fact whether they are 1 through 5 or 6 through 10. These different placements, though the precision statistics are the same, make a difference in practical information retrieval system such as a search engine. To alleviate this deficiency, the average rank $L_m$ of the relevant documents in a returned set of $m$ documents is defined as follows,

$$L_m = \frac{\sum_{i=1}^{C_m} L_i}{C_m}$$

where $L_i$ is the rank of a relevant document $i$ among the top-$m$ returned documents, and $C_m$ is the count of relevant documents among the $m$ returned documents. The idea here is that the average rank should be as low as possible, which means all relevant documents are among the first returned documents; and the count $C_m$ should be as large as possible which means more relevant documents are among the top $m$ documents. Note that $C_m \leq m$. The third statistic collected is the actual response time for the MARS meta-search engine to process and refine the queries. The response times are divided into two categories: the initial response time between the time issuing the query and the time receiving the response from an external search engine; and the time needed for the algorithm MA to refine the results. These two time measurements are called initial time and refine time.
Results and Analysis

The first set of statistics is reported in Table 1. The measurements were taken with $|A| = 200, 150, 100, 50$, respectively, and $m = 10$.

Table 1. Relative Precision and Recall with 72 Queries and $m = 10$

<table>
<thead>
<tr>
<th></th>
<th>(50,10)</th>
<th>(100,10)</th>
<th>(150,10)</th>
<th>(200,10)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>Mars</td>
<td>0.44</td>
<td>0.53</td>
<td>0.46</td>
<td>0.28</td>
</tr>
<tr>
<td>AltaVista</td>
<td>0.44</td>
<td>0.28</td>
<td>0.44</td>
<td>0.17</td>
</tr>
</tbody>
</table>

As can be seen from the Table, MARS in general performs better than AltaVista. When the size of return documents is 200, MARS has an average precision of 65% and an average recall rate of 20%. While a 20% recall rate doesn’t seem to be very high, notice that this is the result of 72 randomly selected queries. MARS was able, on the average, to list 20% of the all relevant documents in the top 10 positions. Note also that AltaVista holds the precision rate of 44% across different values of $|A|$ because no matter what the size of $A$ is, the relevant documents among the top-$m$ ($m=10$ in this case), $R_m$, remain the same for a general purpose search engine such as AltaVista. For MARS, because of the interactive user feedback, the value of $R_m$ varies. The larger the value $|A|$ is, the more likely that the MA algorithm is able to move a relevant document to the top.

The second set of statistics, the average rank and count of relevant documents among the top-$m$ returned documents, is reported in Table 2. The measurement is taken with $m = 20$, that is, the values in the table indicate the average rank and the average count of relevant documents among the top 20 documents. The $RankPower P_m$ is defined as the ratio between the average
rank and the count. The smaller the value of RankPower, the better it is. The results were obtained using the same set of 72 queries.

Table 2. Average Rank and Count of Relevant Documents among Top 20 Results

<table>
<thead>
<tr>
<th></th>
<th>Average Rank $L_m$</th>
<th>Average Count $C_m$</th>
<th>RankPower $P_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARS</td>
<td>6.59</td>
<td>9.33</td>
<td>0.71</td>
</tr>
<tr>
<td>AltaVista</td>
<td>10.24</td>
<td>8.50</td>
<td>1.21</td>
</tr>
</tbody>
</table>

The data indicate that the average rank of relevant documents among the top 20 documents in MARS is 6.59 and the average count is 9.33. Note that in the best case where all top 20 documents are relevant, the average rank should be $\left(\frac{\sum_{i=1}^{20} i}{20}\right) = 10.5$, the average count should be 20, and the RankPower is $10.5/20 = 0.525$. The results show that although the average number of relevant documents does not increase dramatically (9.33 vs. 8.50), their ranks are (6.59 vs. 10.24). The RankPower of MARS (0.71) is much closer to the optimal value.

The statistics in the Table 3 indicates two measures, the original time and the refinement time. The values listed are mean, standard deviation, 95% confidence interval, and the maximum. One should note that the initial time is needed to get any search results from the external search engine whether or not the algorithm MA is involved. As can be seen, the time spent in refining the search results is very small relative to the time to get the initial result.

Table 3. Response Time in Seconds

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>S.D.</th>
<th>95% C.I.</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>3.86</td>
<td>1.15</td>
<td>0.635</td>
<td>5.29</td>
</tr>
<tr>
<td>Refine</td>
<td>0.986</td>
<td>0.427</td>
<td>0.236</td>
<td>1.44</td>
</tr>
</tbody>
</table>
While the general statistics collected in the above tables show that algorithm MA performs very well under various conditions, some individual examples are of special interest. A couple of highly vague terms were chosen among the 72 queries: memory and language to see how MARS handles them. These two words may mean completely differently in different areas. The term memory can mean human memory, or the memory chips used in computers; the term language can refer to spoken language or computer programming language. The examples show that the search precision improves dramatically with very limited relevance feedback in MARS compared to a general-purpose search engine such as AltaVista.

**Memory:** The top-10 initial results sent back from AltaVista include two types of URLs, as expected. One is related to computer memory, the other is related to memory in human beings. Figure 6 shows the list (for space reason, only the top 10 results are listed). Relevant ones are preceded by an R, irrelevant ones by an X. There are five relevant documents in this list.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td><a href="http://www.memorytogo.com/">http://www.memorytogo.com/</a></td>
</tr>
<tr>
<td>X</td>
<td><a href="http://memory.loc.gov/">http://memory.loc.gov/</a></td>
</tr>
<tr>
<td>X</td>
<td><a href="http://lcweb2.loc.gov/ammem/ammemhome.html">http://lcweb2.loc.gov/ammem/ammemhome.html</a></td>
</tr>
<tr>
<td>R</td>
<td><a href="http://www.datamem.com/">http://www.datamem.com/</a></td>
</tr>
<tr>
<td>R</td>
<td><a href="http://www.samintl.com/mem/index.htm">http://www.samintl.com/mem/index.htm</a></td>
</tr>
<tr>
<td>X</td>
<td><a href="http://www.asacredmemory.com/">http://www.asacredmemory.com/</a></td>
</tr>
<tr>
<td>X</td>
<td><a href="http://www.exploratorium.edu/memory/lectures.html">http://www.exploratorium.edu/memory/lectures.html</a></td>
</tr>
<tr>
<td>X</td>
<td><a href="http://www.exploratorium.edu/memory/index.html">http://www.exploratorium.edu/memory/index.html</a></td>
</tr>
<tr>
<td>R</td>
<td><a href="http://www.satech.com/glosomemter.html">http://www.satech.com/glosomemter.html</a></td>
</tr>
<tr>
<td>R</td>
<td><a href="http://www.lostcircuits.com/memory/">http://www.lostcircuits.com/memory/</a></td>
</tr>
</tbody>
</table>

**Figure 6. Top 10 Initial Search Results for Keyword memory**

With one round of refinement where a total of four URLs were marked (two marked in the top 10 list and two marked in the bottom 10 list), four of the original irrelevant URLs were
eliminated. The revised top 10 URLs are listed in Figure 7. The number of relevant documents now increases from five to eight, with relevant documents that the user hasn’t seen before.

| R | http://www.streetprices.com/Electronics/...ware_PC |
| R | http://www.memorytogo.com/ |
| X | http://www.crpuzzles.com/mem/index.html |
| R | http://www.linux-mtd.infradead.org/ |
| R | http://fiberoptics.dimm-memory-infineon....owsides |
| R | http://www.ramplus.com/cpumemory.html |
| X | http://www.asacredmemory.com/ |
| R | http://www.computersupersale.com/shopdis..._A_cat_ |
| R | http://www.datamem.com/ |
| R | http://www.lostcircuits.com/memory/ |

**Figure 7. Refined Search Results for Keyword memory**

**Language:** Similar to the term memory, the search results for language can be roughly divided into two classes, the ones related to human languages and the ones related to computer programming language. Figure 8 lists the original list of top 10 URLs returned from AltaVista. Assume the information about programming languages is of interest here.

| X | http://chinese.about.com/ |
| R | http://www.python.org/ |
| X | http://esl.about.com/ |
| X | http://esl.about.com/homework/esl/mbody.htm |
| X | http://www.aliensonearth.com/catalog/pub/language/ |
| X | http://kidslangarts.about.com/ |
| X | http://kidslangarts.about.com/kids/kidslangarts/mb |
| X | http://pw1.netcom.com/~rlederer/rllink.htm |
| X | http://www.wordcentral.com/ |

**Figure 8. Top 10 Initial Search Results for Keyword language**
As can be seen in Figure 8, only one URL www.python.org is really relevant to the intention of the query. With a refinement of three URLs marked, one marked irrelevant from the top 10 list, one marked relevant from the top 10 list, and one marked relevant from the bottom 10 list (www.suse.de/lang.html), the refined list now contains six relevant URLs shown in Figure 9, compared to only one before refinement. Of these six URLs, one was originally in top 10 and was marked; one was originally in bottom 10 and was marked; the other four were not examined nor marked before at all. But they now showed up in the top 10 list!

**APPLYING THE MA ALGORITHM TO PERSONALIZATION AND CLUSTERING**

The multiplicative adaptive approach in the algorithm MA can be used in cooperation with other mechanisms to improve search accuracy. Here we discuss two such applications, personalization and clustering.

General-purpose search engines return a large number of URLs in response to a query. These returned URLs typically contain the key words used in the query. But they are often not what the
user is looking for because words alone without context usually cannot express the search intent accurately. For example, the search keyword *memory*, or *language* will mean completely different things in different communities and in different contexts. Personalization and clustering are two techniques that can help alleviate this problem. In the PAWS-Cluster project (Meng & Chen, 2003), the algorithm MA is used in conjunction with a personalization component and a clustering component to improve search accuracies. When a list of search results is returned from a general-purpose search engine, the PAWS-Cluster sends the list through a personalizer or a cluster based on user’s selection. These results are then fed through the MA algorithm after the user has marked relevant or irrelevant on a part of the list, similar to the case in the MARS project. The MA algorithm re-calculates the score after promoting or demoting the original search results. The revised list is then presented to the user for further refinement. The architecture of PAWS-Cluster is presented in Figure 10. Note that the only difference between PAWS-Cluster and MARS architecture (Figure 2) is that the PAWS-Cluster contains two extra components, a personalizer and a cluster.
The Personalizer is a key part of PAWS-cluster. People leave *digital traces* on the computers they use, especially on the computers dedicated to a single person. These traces include, among other things, email messages, digital news, work related documents, personal documents and others. All these traces are distinct from one person to another because of the nature of their work, their personalities and other characteristics. When a person performs a search on the Web, the information interesting to that person is ultimately related to the digital trace left on her computer. If these digital traces are used to filter the search results returned from search engines before presenting to the user, one would expect the results be much more accurate. The user can collect her own profile on the client side. This collecting process is done periodically, not every time the user wants to search for something. The collection of this profile can be a part of the client software (browser) functionality. Also possible is to have a separate program perform this task. The key issues here are that the collecting process is initiated by
individual users; the user knows exactly what is collected; and the results are not available to anyone else, including the search engines.

In the PAWS-Cluster project, the words that appeared in user's documents on his/her desk-top computer were used as the base of the profile. The profile consists of a number of most frequently used words in user's document collection. The collecting process simply traverses the directory tree of the user's computer, examining every document on its way. After some basic text processing, the result is sorted according to the appearance frequencies of these words. The top $m$ words are kept as the profile.

When a URL is retrieved from a search engine along with its brief summary, a similarity measure is computed between the profile and the URL. The similarity is measured by the popular cosine similarity (Salton, 1989). A returned URL $U$ along with its short summary contains a set of words, so does the user's profile $P$. $U$ and $P$ can be represented as an $m$-dimensional vector $<w_1, w_2, \ldots, w_m>$ where the $i$-th component $w_i$ represents the significance of the $i$-th word in the vocabulary. Thus similarity $S$ between a URL $U$ and the profile $P$ becomes as follows.

$$S = \sum_{i=1}^{m} \frac{W_U W_{ip}}{\|U\| \|P\|}$$

Since a typical general-purpose search engine returns a long list of URLs when a search query is issued, it is easier to mark the URLs as clusters. The user would not have to examine every URL in a cluster. Rather the user only needs to examine a representative from a cluster. Once the cluster is marked relevant or not, the same MA algorithm is applied to each URL in the cluster.
That is, the relevant documents are promoted and the irrelevant documents are demoted. The clusters are derived by the correlation similarities among the URLs.

If the correlation similarity between two URLs is greater than a given threshold, these two URLs are put into the same cluster.

The clustering algorithm takes two steps. First it computes pair-wise cosine similarities of all documents. This computation generates a similarity matrix $s[i:d][j:d]$ where $s[i][j]$ is the correlation similarity between document $i$ and document $j$. The documents are then divided into clusters based on their correlation similarities. The algorithm goes through the correlation similarity matrix row by row and if the similarity between the two documents is greater than a given threshold, the two documents are put into the same cluster.

The following two examples illustrate how the system works. One is a personalization example, and other is a clustering example. These experiments were performed during the winter of 2003.

**Personalization Example:** Search keyword *memory*. Assume the initial search is meant for computer memory chips. Without personalization, the places for the relevant URLs are 2, 3, 7, 8, 10, or an average of 6 for the five relevant URLs returned by the general-purpose search engine. With personalization, the places become 1, 3, 5, 6, 8, 9, or an average of 5.3 for the six relevant URLs. For this example, at least, personalization helped bring more relevant URLs and placed the relevant URLs higher in the list.
Cluster Example: Search keyword *xiannong*, first name of one of the authors. Before clustering, all relevant URLs are scattered among the returned URLs. After applying the cluster algorithms many related URLs are clustered together. Some URLs that were seemingly un-related are now put into the same cluster because their contents (a brief description that was sent from the general-purpose search engine) are indeed related. Two examples are quoted here.

*Cluster 13*

http://www.cs.montana.edu/~bhz/pubs.html

http://www.cs.montana.edu/~bhz/recent.html


In this example, the last URL contains one of the authors (Xiannong Meng) that appeared in the first two URLs as one of the co-authors in a list of publications of Binhai Zhu. In the original returned list, the places of these three URLs were 43, 44, and 64, respectively.

*Cluster 14*

http://www.asis.org/Publications/JASIS/vol52n8.html


The first URL in Cluster 14 points to JASIS (Journal of the American Society for Information Science and Technology) in which the two authors published a paper. The second URL is the table of contents of another journal (KAIS, Knowledge and Information Systems) where the two authors published a separate paper. Originally, these two URLs were 42 places apart (48 and 90).

**TRENDS AND CHALLENGES**

Search engines have become dominant tools to find information over the Web in the few past years. How to accurately and efficiently locate a piece of information among hundreds or thousands of Web pages is an increasingly important and challenging issue. With ever growing
number of Web pages available and the ambiguous nature of human languages, it is just not possible for a general-purpose search engine to return exactly what the user wants when a query is given in most cases. Refining what a general-purpose search engine will return in response to a search query becomes inevitable. Researchers are taking many different approaches to tackle the problem. The authors believe that adaptive refinement using relevance feedback is an important way of solving the problem. Some basic questions would have to be answered before real progresses can be made.

- **How to represent and capture user preferences:** This is a question that both ends of the search would have to answer. A user may use a particular set of vocabulary to represent his/her search intention. A search engine has to be able to capture and understand the true search intention when a query is received. Personalized profiles may help narrow this gap. But profiling raises the issues of privacy and scalability.

- **How to make adaptive refinement efficient:** Search engines can refine the search when receiving users’ feedback. To effectively use the feedback to narrow the search results, the search engines would have to understand the feedback and be able to refine the search. AI techniques, machine learning in particular, may help improve this process.

- **Collaboration between search engines and browsers:** Currently the search engines and the browsers work as two independent camps. For adaptive refinement to succeed, browsers should understand how search engines work and carry out certain processing that is traditionally done by the search engines.

**CONCLUDING REMARKS**

The motivations of the work in this chapter come from the reality of Web search: Web search users usually have no patience to try, say, more than five iterations of relevance feedback for
some intelligent search system in order to gain certain significant search precision increase. In contrast to the adoption of linear additive query updating techniques in existing algorithms, a new algorithm, multiplicative adaptive query expansion algorithm MA, is designed. The algorithm uses multiplicative query updating techniques to adaptively improve the query vector. Algorithm MA has been implemented in project MARS to show its effectiveness and efficiency. The algorithm has provable better performance than the popular Rocchio's similarity-based relevance feedback algorithm in learning a user preference determined by a linear classifier with a small number of non-zero coefficients over the real-valued vector space. Experiments indicate that algorithm MA substantially improves the search performance.

**URLS Referenced in the Chapter**


**REFERENCE**


