Int. J. Appl. Math. Comput. Sci., 2006, Vol. 16, No. 4, 537–549

GUIDED LOCAL SEARCH FOR QUERY REFORMULATION USING WEIGHT PROPAGATION

ISSAM MOGHRABI

Computer Science Department
Faculty of Science, Beirut Arab University
P.O. Box 11–5020, Beirut, Lebanon
e-mail: imoghrabi@bau.edu.lb

A new technique for query reformulation that assesses the relevance of retrieved documents using weight propagation is proposed. The technique uses a Guided Local Search (GLS) in conjunction with the latent semantic indexing model (to semantically cluster documents together) and Lexical Matching (LM). The GLS algorithm is used to construct a minimum spanning tree that is later employed in the reformulation process. The computations done for Singular Value Decomposition (SVD), LM and the minimum spanning tree are necessary overheads that occur only initially and all subsequent work is based on them. Our experimental results reveal the effectiveness of the new technique.

Keywords: relevance feedback, clustering, latent semantic, query reformulation

1. Introduction

In a relevance feedback cycle, the user is presented with a list of retrieved documents and, after examining them, marks those which are relevant. In practice, only the top 10 (or 20) ranked documents need to be examined. The main procedure consists in selecting important terms, or expressions, attached to the documents that have been identified as relevant by the user, and enhancing the importance of these terms in a new query formulation. The expected effect is that the new query will focus on targeting the relevant documents and skip the nonrelevant ones (Baeza-Yates and Ribeiro-Neto, 1999, p. 118). Relevance Feedback (RF) has been found to improve precision by up to 60% (Daniels and Rissland, 1995).

Some of the most advanced RF techniques used in operating interactive information retrieval systems are query reformulation using classification, clustering vectors (Rijisbergon, 1979), Latent Semantic Indexing (LSI) (Letsche and Berry, 1997) and query expansion. Query expansion could be both interactive and automatic (Ruthven *et al.*, 2001).

Standard relevance feedback algorithms do not usually perform better given negative relevance judgment evidence (Dunlop, 1997), although in this paper we will exploit a negative feedback.

2. Latent Semantics

The LSI model was described in detail in (Letsche and Berry, 1997). In this paper we will briefly review some of those details. LSI is an extension of the vector retrieval method. The representation of the documents is based mainly on the relationships between the terms.

It is assumed that there is some underlying, or "latent", structure in the pattern of word usage across documents. Statistical techniques can be used to estimate this latent structure. A description of terms, documents and user queries based on the underlying ("latent semantic") structure, rather than a surface level word choice, is used for retrieving information.

Truncated SVD is used to estimate this latent structure. The computation of truncated SVD of a large sparse term by document matrices is expensive, but it is performed only once. The most common method used for the solution of the SVD problem is incorporated in the Golub-Kahan SVD algorithm, which requires $O(mn^2)$ time on a single processor computer, where m represents the number of terms and n stands for the number of documents. Vectors obtained from truncated SVD are used for the retrieval process.

In an LSI model, an $m \times n$ matrix A is used to represent the terms and documents, respectively. Each of the m unique terms in the document collection are assigned a row in the matrix, while each of the n documents in the collection is assigned a column in the matrix.

The SVD of the matrix A is defined as the product of three matrices,

$$A = U\Sigma V^T,$$

where the columns of U and V are the left and right singular vectors, respectively, corresponding to the monotonically decreasing (in value) diagonal elements of Σ, which

are called singular values of the matrix A. Two theorems in (Letsche and Berry, 1997) illustrate how SVD can reveal important information about the matrix structure.

The first k columns of U and V and the first k singular values of A are used to construct a rank-k approximation to A via

$$A_k = \Sigma u_i \sigma_i v_i^T$$

for $i = 1, \ldots, k$. Truncated SVD is illustrated in Fig. 1.

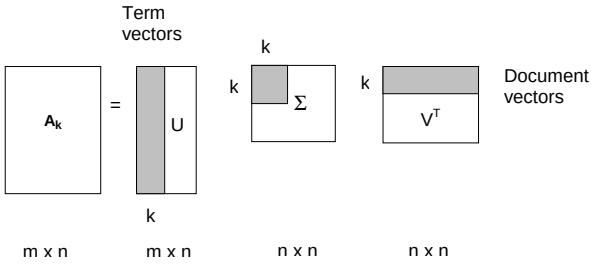

Fig. 1. Truncated SVD.

The user query is represented as a vector in the k-dimensional space by

$$\text{query} = q^T U_k (\Sigma_k)^{-1},$$

where q is simply the vector of words in the user's query. The query vector is compared with all document vectors using a common measure of similarity such as the cosine between the query vector and the document vector. The corresponding relevance feedback query is given by

$$f = q^T U_k (\Sigma_k)^{-1} + d^T V_k,$$

where q specifies the terms in the original query, and d signifies a vector whose elements specify the documents in the query.

3. Guided Local Search

Guided Local Search (GLS) (see Voudouris, 1997) for a more detailed description) is a meta-heuristic based on local search. Local search is the basis of many heuristic methods for combinatorial optimization problems. When a given local search algorithm settles in a local optimum, GLS changes the objective function. Iterative calls are made to the local search, each time using a new modified objective function which is intended to bring the algorithm out of a local optimum neighborhood. The key is in the way that the objective function is modified.

GLS employs solution features to distinguish between solutions with different characteristics, so that bad characteristics can be penalized by GLS, and hopefully removed by the local search algorithm. The choice of solution features depends therefore on the type of the problem, and also, to some extent, on the local search algorithm. A feature f_i is represented by the following indicator function:

$$I(s) = \begin{cases} 1 & \text{if a solution has the property } i, \\ 0 & \text{otherwise,} \end{cases}$$

to indicate whether or not the feature is present in the current solution.

A feature must have other components such as the cost function $c(s)$, which gives the cost of having the feature present in a solution, and a penalty p_i, initially set to zero, used to penalize the occurrences of the feature in local minima.

When the local search algorithm returns a local minimum, s, which is not a legal solution, GLS increments the penalty (penalize) of all the features present in that solution which have maximum utility, util(s, f_i), defined as

$$\text{util}(s, f_i) = I_i(s) \frac{c_i(s)}{(1 + p_i)}.$$

The idea is to penalize features which have high costs, although the usefulness of doing so decreases as the feature is successively penalized. Local search is guided out of the local minimum by using an augmented cost function. This is done by penalizing features present in that local minimum. The idea is to make the local minimum costlier than the neighboring search space where these features are not present. The augmented cost function is defined as follows:

$$h(s) = g(s) + \lambda \sum_{i=1}^{n} p_i I_i(s).$$

The strengthening of the search for solutions is controlled by the parameter λ. A low value will result in a more concentrated search for the solution, where the basins and plateaus in the search landscape are searched with greater care. A higher value of λ will result in a more diverse search, where plateaus and basins in the search are searched less carefully.

4. Relevance Feedback and Weight Propagation

The weight propagation method was introduced by Yamout *et al.* (2006). A relevant document propagates positive weights to neighboring documents and negative weights if it is nonrelevant. Initially, each document is represented by a vector of terms.

Let $G = \langle N, A \rangle$ be a connected graph where N is the set of nodes and A is the set of edges. Consider N to represent the documents in the database and the edges to be the distances between them using a symmetric distance matrix $D = [\text{distance}(i, j)]$, which gives the distance between any two documents i and j. With LM the

documents are viewed as vectors in the multidimensional term space of dimension m, where m is the number of terms in the database. Thus, the matrix D contains the distances between the documents using any similarity measures such as the dot product (or inner product) to find a Euclidean distance. Alternatively, the Hamming distance function may be used to measure the "difference" between two vectors (Susanna, 1990), instead of merely using the similarity measure.

For illustration, in Fig. 2 *doc01* is a relevant document. Therefore, it propagates positive weights to *doc2* and *doc3*, while *doc4*, which is not relevant, propagates negative weights to the same documents.

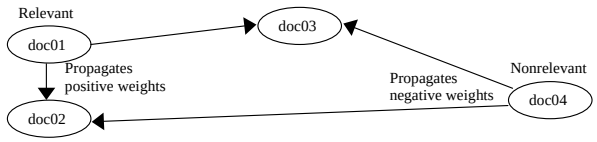

Fig. 2. Documents propagating negative and positive weights.

The propagation of weights is affected by how far the documents are from a chosen origin. For example, a document that is close to a relevant one is affected more than a distant one. For this reason, the weight w_{ij}, propagated from the document i to the document j, is based on the distance between the two documents defined by

$$w_{ij} = 1/\text{distance}(\text{doc}\, i, \text{doc}\, j).$$

This process is repeated for all the relevant and nonrelevant documents. The propagated weights are summed up for each document, and the documents with positive weights are to be retrieved as the result of the relevance feedback process. Figure 3 shows how weights are added for the documents 2 and 3.

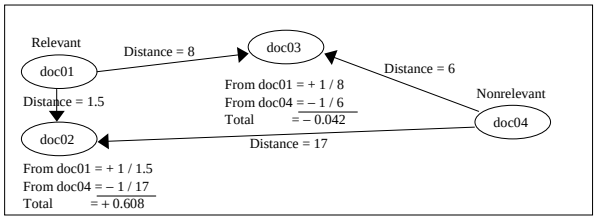

Fig. 3. Adding up weights.

The document *doc4* propagates a $-1/6$ weight to *doc03* and a $-1/17$ one to *doc02*, while the document *doc01* propagates a $+1/8$ weight to the document *doc03* and a $+1/1.5$ one to *doc02*. It should be emphasized how the weights propagated are affected by the distance. As a result, the weights are added up and *doc03* accumulates -0.042 while *doc02* accumulates $+0.608$. It is evident that *doc02*, with a total weight of $+0.608$, will be retrieved as a result of the relevance feedback process.

The relevance feedback used in the LSI model (see Section 2) is rather different. The user query is represented as a vector in the k-dimensional space by

$$\text{query} = q^T U_k (\Sigma_k)^{-1},$$

where q is the vector of terms in the user's query. The coordinates for the query are computed (see Appendix B for an example) and represented by the vector labeled *query* in Fig. 4.

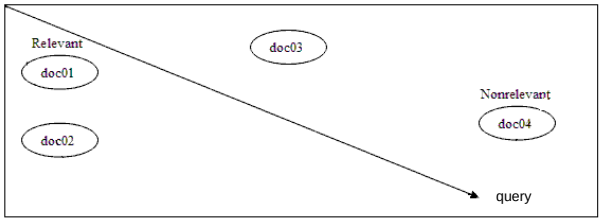

Fig. 4. User query represented as a vector in two dimensions.

All documents whose cosine with the query vector is greater than a threshold are returned (Fig. 5); the documents are *doc01*, *doc02*, *doc03* and *doc04*. The user then

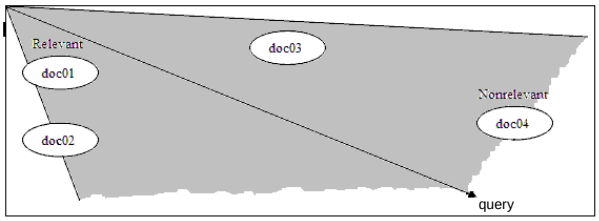

Fig. 5. Documents whose cosine with the query vector is greater than a threshold.

chooses, e.g., *doc01* as relevant and *doc04* as nonrelevant. The relevance feedback query is given by

$$f = q^T U_k (\Sigma_k)^{-1} + d^T V_k,$$

where d is the vector of documents chosen as relevant. The coordinates for the feedback query are computed and represented by the vector labeled f in Fig. 6.

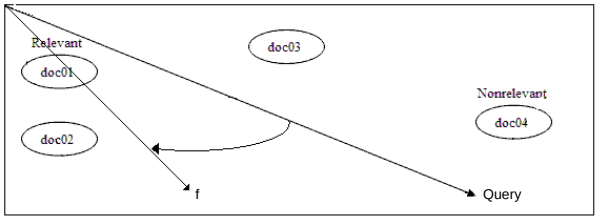

Fig. 6. Feedback query represented by the point labeled f.

The relevance feedback query vector is compared with all documents using the cosine measure of similarity between the relevance feedback query vector and the documents. All documents whose cosine with the feedback query vector is greater than a threshold are returned; (Fig. 7) the documents are *doc01* and *doc02*.

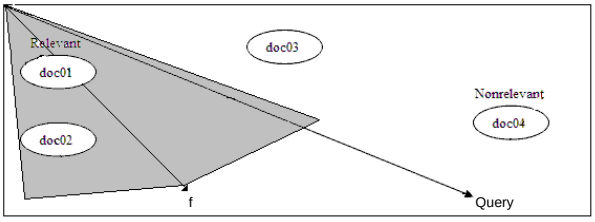

Fig. 7. Documents whose cosine with feedback query vector is greater than a threshold.

Since, in our technique, the weight magnitude is based on the distance between documents, the weight propagated to a distant document is insignificant. Therefore, it would be more profitable to remove lengthy edges (shown using a dashed line). A minimal cluster is appropriate for this problem since it will return a subset of edges such that all nodes remain connected and the sum of the edge lengths is as small as possible. Thus, for each document, only those connected to it with an edge are considered the nearest ones.

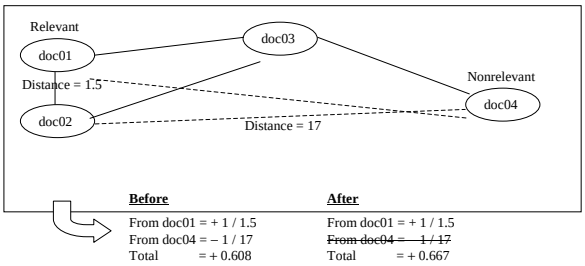

Fig. 8. Removing lengthy edges.

5. Guided Local Search and Query Reformulation

The GLS algorithm is used to build a minimal cluster for which the total distance between documents is minimal. Recall from Section 3 that GLS finds a set of solution features that are responsible for part of the overall solution cost, and that it employs solution features to distinguish between solutions with different characteristics. It is essential to determine features that depend completely on the type of the problem. In this paper the goal of GLS is to find a tour which visits each document exactly once and is of minimum length. This tour includes a number of edges and the solution cost (the tour length) is the sum of the edge lengths in the tour. Therefore, edges are ideal features for our problem.

Each edge carries a cost equal to the edge length, given by the distance matrix $D = [\text{distance}(i, j)]$. The indicator function, $I(s)$, which indicates whether or not the feature is present in the current solution, is applied to decide whether or not a tour includes an edge.

Local search starts from an arbitrary solution to find a local minimum. The basic GLS algorithm described so far is depicted in Fig. 9.

procedure GuidedLocalSeach$(S, g, l, [I_1, \ldots, I_M], [c_1, \ldots, c_M], M)$
begin
$k \leftarrow 0$;
$s_0 \leftarrow$ random or heuristically generated solution in S;
for $i \leftarrow 1$ until M **do** /* set all penalties to 0 */
$p_i \leftarrow 0$;
while StoppingCriterion **do**
begin
$h \leftarrow g + \lambda * \Sigma p_i * I_i$;
$s_k + 1 \leftarrow$ LocalSearch(s_k, h);
for $i \leftarrow 1$ until M **do**
uti $l_i \leftarrow I_i(s_k + 1) * c_i/(1 + p_i)$;
for each i such that utili is maximum **do**
$p_i \leftarrow p_i + 1$;
$k \leftarrow k + 1$;
end
$s* \leftarrow$ best solution found with respect to cost function g;
return $s*$;
end

Notation: S: search space, g: cost function, h: augmented cost function, λ: regularization parameter, I_i: indicator function for feature i, c_i: cost for feature i, M: number of features, p_i: penalty for feature i.

Fig. 9. Guided local search algorithm.

GLS increments the penalty of all the features present in a local minimum which have maximum utility, util(s, f_i). Therefore, each edge e_{ij} connecting the documents i and j is assigned a penalty (i, j), initially set to 0, and GLS will penalize the edges that appear in some local minimum, using the utility function. The edge penalties can be arranged in a symmetric penalty matrix $P = [\text{penalty}(i, j)]$. Due to the symmetry, only $n(n-1)/2$ components have to be stored.

Penalties have to be combined with the problem's cost function to form the augmented cost function minimized by local search. This can be done by considering the auxiliary distance matrix

$$D' = D + \lambda P = \Big[\text{distance}(i, j) + \lambda \text{ penalty}(i, j)\Big].$$

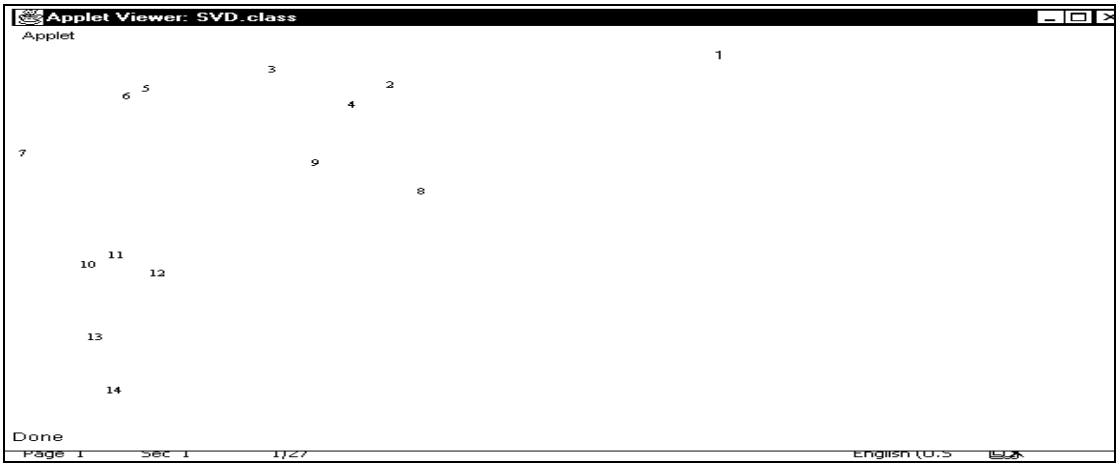

Fig. 10. Documents represented on the Cartesian plane.

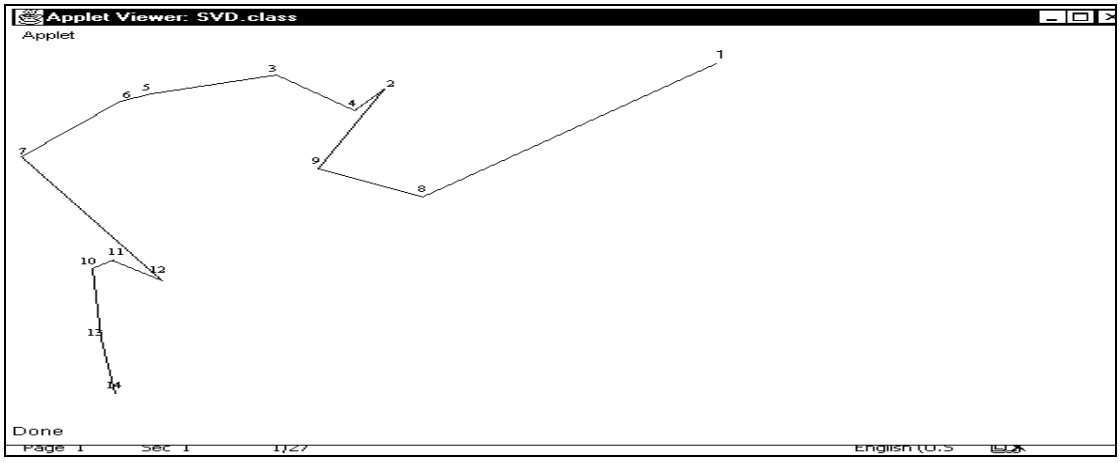

Fig. 11. Shortest path returned by GLS.

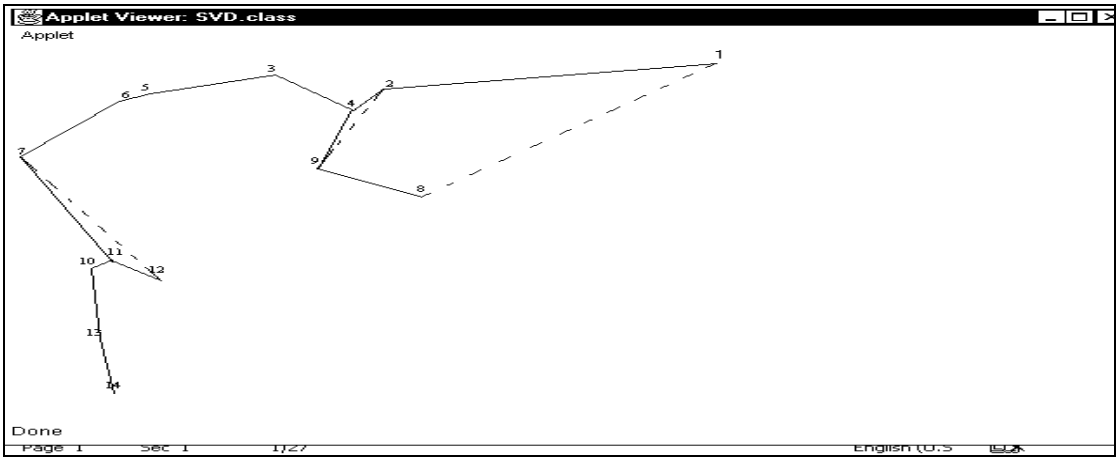

Fig. 12. Minimum spanning tree.

Local search must use D' instead of D in move assessments. GLS modifies P and thereby D whenever the local search reaches a local minimum.

After the penalties have been increased, local search is restarted from the last local minimum to search for a new local minimum. This is repeated until a termination criterion is satisfied. In the general case, the criterion is either a maximum number of iterations or a time budget (Voudouris and Tsang, 1994).

We modify the basic GLS algorithm to match it to our objectives of building a minimal cluster. The local search algorithm embedded in the GLS is to disallow it from returning to the initial document it started from. This will result in a minimal cluster for the documents rather than a closed path. Consider, e.g., a database with 14 documents. Figure 10 shows these documents represented on the Cartesian plane using the SVD algorithm. The GLS algorithm will return a near optimal solution that reveals the shortest path that traverses all the documents in the document space as shown in Fig. 11.

Consider, e.g., the document 6. The document will propagate to the documents 7 and 5, while the document 11 will propagate to the documents 10 and 12.

An additional modification is needed to adjust the edges, since it would not be appropriate for the document 12 to propagate to the document 7 while the document 11 is closer to it than 12. Therefore, the edge that connects the document 12 to 7 is removed, and replaced by an edge that connects the document 11 to 7 instead. These steps are prescribed in the for loop in the line 04 in the *Construct_DS* algorithm (cf. Fig. 15). The edge removed is shown as a dashed line. The result is a minimum spanning tree. The results are presented in Fig. 12.

In order to perform relevance feedback using weight propagation, an intermediate mixed data structure (MDS) is built for collecting information pertaining to documents. First, the representation of documents and the neighborhood can (in our view) be best captured in a linked list, where nodes represent documents and edges represent the connection to the nearby documents. A linked list is constructed for each document. All the documents that are in close proximity are inserted into the corresponding linked list. Second, an array of documents is constructed such that each entry forms a pointer to a linked list of documents which contains the nearest documents. Fig. 13 illustrates the data structure that represents the documents in Fig. 12.

For example, the document 4 has an entry in the array and is connected to a linked list that contains the documents 2, 3 and 9. The distances between the documents in the array and the nearest ones are kept in a linked list. For example, in Fig. 14 the distances between the document 4 and the nearest documents are stored in a linked list.

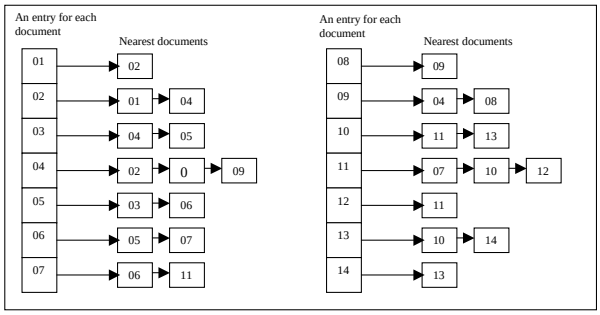

Fig. 13. Linked representation of the documents in Fig. 12.

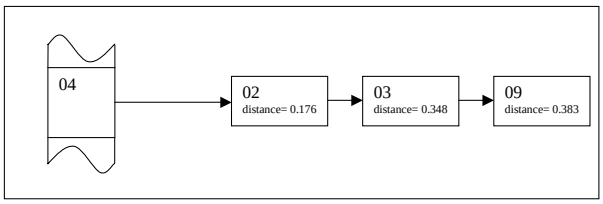

Fig. 14. Example of the nodes stored in a linked list.

The algorithm for constructing the MDS traverses the documents returned by GLS and for each document connected to it – a linked list that contains the nearest ones. The steps are formalized in Fig. 15.

*// **function**: Construct_DS (sol[ ], D)*
//
// Construct the Mixed Data Structure (MDS) for relevance
// feedback using weight propagation
// sol: array that contains the documents traversed by GLS
// D: distance matrix. Contains the edge lengths that connect
// two documents
// n: number of documents in the database
// distance(ij): distance between document i and j
// A: array used in the MDS. It contains elements of linked
// lists type
// nd: represents a node in the linked list. It contains two
// variables:
// nd.document: document number
// nd.distance: distance

01 **for** $i = 1$ to $(n-1)$ **do**
02 min = distance$(\mathrm{sol}[i], \mathrm{sol}[i+1])$
03 $i_{\mathrm{short}} = i//i_{short}$ is an index to reference the closest document to document$_{i+1}$
04 **for** $j = 1to(i-1)$ **do**
05 **if** distance(sol$[j]$,sol$[i+1]$) < min
06 min = distance(sol$[j]$,sol$[i+1]$)
07 $i_{\mathrm{short}} = j$
08 **create** a new node nd with nd.document = sol$[i+1]$ and nd.distance = distance(sol$[i_{\mathrm{short}}]$,sol$[i+1]$)
09 **add** nd to the linked list at A[sol$[i_{\mathrm{short}}]$]
10 **create** a new node nd with nd.document = sol$[i_{\mathrm{short}}]$ and nd.distance = distance (sol$[i+1]$,sol$[i_{\mathrm{short}}]$)
11 **add** nd to the linked list at A [sol$[i+1]$]
12 **return** MDS

Fig. 15. *Construct_DS* algorithm.

The algorithm for the reformulation process using weight propagation (as explained earlier) is specified in Fig. 16. It is worth mentioning here that either the Hamming Distance (HD) or LM is used to calculate the matrix D in GLS.

```
// function: Reformulation (MDS, q)
//
// Reformulation process based relevance feedback using
// weight propagation
// MDS: Mixed data structure that contains an array where
// each of its element represents a document in the
// database. Each element is connected to a linked list that
// contains the nearest documents
// q: contains the documents retrieved initially by the user's
// query
// RFL: Linked list that contains the documents to be returned
// as a result of the reformulation process
// rf_nd: represents a node in the RFL linked list.
//       It contains two variables:
//         rf_nd.document: document number
//         rf_nd.weight: distance
// A: array used in the MDS. It contains elements of linked
// lists type
// nd: represents a node in the linked list connected to an
// element of the array in the MDS.
//       It contains two variables:
//         nd.document: document number
//         nd.distance: distance

01 repeat
02     doc < - document removed from q
03     if doc is relevant then
04         sign = +1
05         create a new node rf_nd with rf_nd.document = doc
           all relevant documents are retrieved
06         add rf_nd to RFL
07     else sign = -1
08     traverse the linked list connected to A[doc] and for each
       node nd
09         locate in RFL for a node with rf_nd.document =
           nd.document
10         if node not found
11             create a new node rf_nd with rf_nd.document =
               nd.document
12             add rf_nd to RFL
13         rf_nd.weight = rf_nd.weight + sign*(1/nd.distance)
14 until q is empty
15 return RFL
```

Fig. 16. Reformulation algorithm.

6. Complexity Analysis

Let q be the number of documents retrieved initially by the query and marked relevant or nonrelevant by the user, and n' be the maximum number of documents found in a linked list pointed to by an element in the data structure array. For each of the q documents, we will have at most n' documents to traverse in the linked list. Therefore, the complexity of the proposed technique is $O(qn')$. The value of q depends on the choices made by the user. Recent experiments have shown that the user will mark on average 10 documents as relevant or nonrelevant. We will show next that the value of n' is actually very small.

The maximum number of elements (documents) in a linked list would not exceed n, which is the number of documents in the database, since a document may be compared with at most $n-1$ documents. We will show next that, on average, the number of documents to traverse in a linked list is at most two. Initially, each of the documents found in the minimal cluster is connected to at most two documents. The total number of edges is not affected by the adjustments of the edges, since each edge added to a node is faced with the removal of another.

We have performed some experiments to examine the maximum and minimum nodes a linked list could possibly have. These experiments are based on inverted files with different sizes, generated with their corresponding linked lists. To determine the average number of 1s required per document, three document collections were used: the standard King James version of the Bible (Table 1), the full text of *Alice in Wonderland*, and a set of several hundred securities filings from the *Securities and Exchange Commission's EDGAR database* (Fray, 1996).

For the Bible, the average number of terms per document is 21.6 out of 9,232 terms (i.e., the number of 1s is equal to 0.25% of the total number of terms). For *Alice in Wonderland*, the average number of terms per document is 25.7 out of $1,891$ terms (i.e., the number of 1s is equal to 1.40% of the total number of terms). For EDGAR, the average number of terms per document is 940.8 out of $20,391$ terms (i.e., the number of 1s is equal to 5.00% of the total number of terms). The average number of terms is 2.5% (the mean of 0.25%, 1.40% and 5.00%) of the total number of terms. Therefore, 2.5% of the terms were assigned randomly to each document.

The results are shown in Tables 2–5, where for each inverted file (IF) the maximum number of nodes, n', in the linked lists is shown.

7. Experimental Results

This section describes the experiments conducted on different test collections using the existing and some state-of-the-art techniques. It starts with the small Medline test collection in two environments, the first one in the LSI model and the second one using LM methods. The experiments are then conducted on two large test collections, WT10G and WT18G. The results are measured by the mean of 11 different recall values (0.0, 0.1, ... 0.9, 1.0) followed by efficiency results. The Rocchio and Ide techniques are used together in conjunction with the WP

Table 1. Three documents collections used to determine the number of 1s.

	Style	Documents	Terms	No. of 1's	No. of 1's per term	No. of 1's per document
Bible	Each verse is a document	31,219	9,232	673,106	73	21.6
Alice	Each paragraph is a document	835	1,891	21,433	11	25.7
Edgar	Each filing is a large document	605	20,391	569,183	28	940.8

Table 2. Maximum number of nodes for fifty inverted files of size 50×50.

Inverted files generated with size = 50×50									
IF	n'	IF	n'	IF	n'	IF	n'	IF	n'
1	5	11	4	21	4	31	4	41	7
2	5	12	4	22	4	32	4	42	6
3	4	13	6	23	5	33	5	43	4
4	5	14	4	24	5	34	4	44	7
5	4	15	5	25	5	35	5	45	7
6	4	16	4	26	6	36	6	46	5
7	7	17	8	27	5	37	4	47	5
8	6	18	5	28	3	38	5	48	4
9	5	19	5	29	5	39	8	49	4
10	5	20	4	30	5	40	4	50	5
Maximum linked list varies between 3 and 8									

Table 3. Maximum number of nodes for fifty inverted files of size 100×100.

Inverted files generated with size = 100×100									
IF	n'	IF	n'	IF	n'	IF	n'	IF	n'
1	21	11	10	21	13	31	19	41	12
2	13	12	13	22	16	32	8	42	11
3	17	13	14	23	13	33	8	43	14
4	10	14	14	24	11	34	12	44	9
5	14	15	19	25	15	35	10	45	8
6	15	16	15	26	16	36	16	46	12
7	13	17	12	27	14	37	13	47	12
8	14	18	10	28	12	38	13	48	12
9	12	19	12	29	7	39	17	49	18
10	6	20	14	30	14	40	13	50	14
Maximum linked list varies between 6 and 21									

method and tested with different values of n' (the number of documents affected by the propagation) using positive and negative feedback.

Table 4. Maximum number of nodes for fifty inverted files of size 200×200.

Inverted files generated with size = 200×200									
IF	n'	IF	n'	IF	n'	IF	n'	IF	n'
1	43	11	66	21	62	31	53	41	53
2	59	12	70	22	56	32	42	42	66
3	45	13	57	23	49	33	42	43	53
4	44	14	52	24	65	34	61	44	58
5	43	15	65	25	46	35	50	45	50
6	75	16	69	26	48	36	54	46	66
7	68	17	60	27	51	37	46	47	47
8	56	18	44	28	60	38	60	48	61
9	63	19	56	29	43	39	72	49	48
10	58	20	57	30	82	40	89	50	50
Maximum linked list varies between 42 and 89									

Medline Document Collection. The Medline collection is a small-size test collection (1.1 Megabytes). It consists of 1,033 documents and 8,915 terms after indexing. The experiments were conducted first using the LSI model preceded by computing SVD to estimate the underlying (or "latent") structure in the pattern of word usage across documents and by determining optimal dimensionality that will correctly capture the underlying semantics that exist between the documents. Our experiments show that peak performance is achieved at dimension 80. The obtained results demonstrate the performance using the average precision followed by the efficiency of the techniques in terms of time.

Medline and LSI. The 11-point average precision for the Medline collection is 0.6626 for the query formulation alone, without query reformulation. The average precision for LSI with Rocchio weighing and positive feedback, but no WP, is somewhat higher at 0.72335. The average precision is still better when WP is used in conjunction with LSI and positive Rocchio weighing for any number of propagated documents. Peak performance is found for 24 and 42 propagated documents, when the av-

erage precision is 0.855 (see Fig. 6). When the experiments were repeated with both positive and negative Rocchio weighing, the performance of LSI without WP was found to be poorer than when positive weighting alone had been used, with an average precision of 0.668. The performance of LSI with WP became less degraded by the incorporation of negative feedback, and peak precision of 0.825 was obtained when weights were propagated to 48 or 50 documents (cf. Fig. 17).

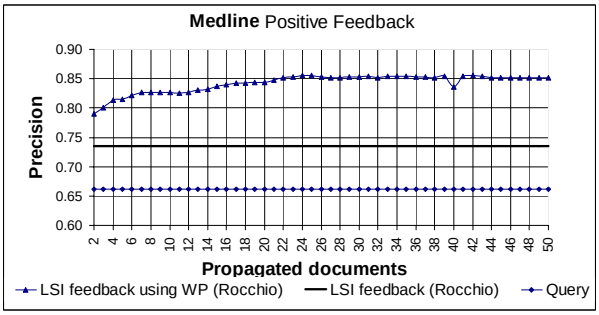

Fig. 17. Precision comparison for different methods.

When the experiments are repeated by using Ide weighing instead of Rocchio one, an average precision of 0.744 is obtained for LSI with positive weighing and no WP, and a best average precision of 0.865 when WP is propagated to 40 documents or more (cf. Fig. 18). Thus, the performances of both the Rocchio and Ide weighing formulas when using positive feedback are similar.

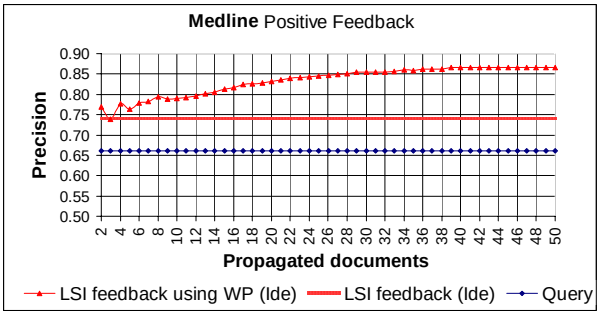

Fig. 18. Precision comparison for different methods.

The performance using the Ide formula degraded when both positive and negative feedback was used, both with and without WP (see Fig. 19).

When LSI is used in conjunction with Ide weighing with both positive and negative feedback, the average precision is 0.536 without WP (which is poorer than in the case where no relevance feedback was used at all), and the highest precision when using WP was 0.877 for propagation to 46 or more documents. The main conclusion here is that the average precision when using WP is better than that obtained without WP for all the Medline experiments. The peak performance of WP does not degrade

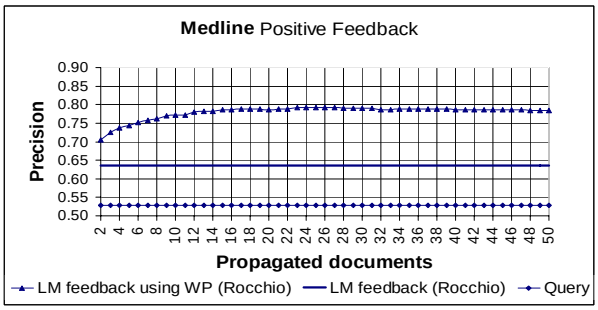

Fig. 19. Precision comparison for different methods.

when negative feedback is taken into consideration, although precision without WP did so. The performance of the Ide formula was more degraded by negative feedback than the Rocchio formula, but otherwise the two formulas produced comparable performances. Table 5 summarizes the results.

Table 5. Precision for the Medline test collection using LSI.

		LSI Query = 0.662	
		Positive feedback (Rocchio)	Positive feedback (Ide)
LSI RF =		**0.735**	**0.741**
Weight propagation with different values of n'	2	0.790	0.770
	4	0.814	0.776
	6	0.822	0.778
	8	0.826	0.793
	10	0.826	0.790
	12	0.827	0.795
	14	0.832	0.805
	16	0.840	0.817
	18	0.843	0.826
	20	0.844	0.831
	22	0.851	0.839
	24	0.855	0.844
	26	0.853	0.846
	28	0.852	0.851
	30	0.853	0.854
	32	0.852	0.855
	34	0.854	0.860
	36	0.853	0.862
	38	0.852	0.863
	40	0.836	0.865
	42	0.855	0.865
	44	0.852	0.865
	46	0.851	0.865
	48	0.852	0.865
	50	0.851	0.865

Medline using LM Methods. The Medline experiments were all repeated, using LM as the initial query formulation method rather than LSI as before (see Figs. 19 to 21). For the baseline evaluation (no reformulation), the precision was poorer for LM than LSI. Otherwise, the results were very similar for both sets of experiments, with the following findings:

(a) Without WP, a positive feedback outperforms a combination of positive and negative feedbacks.

(b) When using WP, precision is less degraded by negative feedback than without using WP.

(c) The Ide formula produced better precision than the Rocchio one. Table 6 summarizes the results.

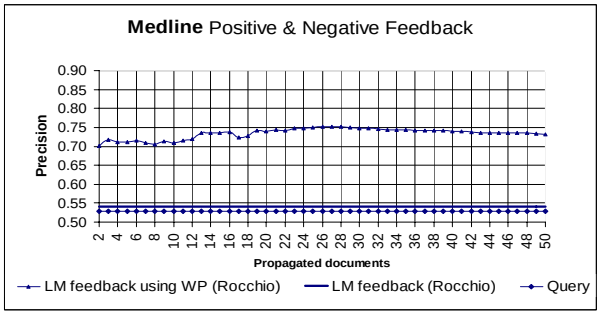

Fig. 20. Precision comparison for different methods.

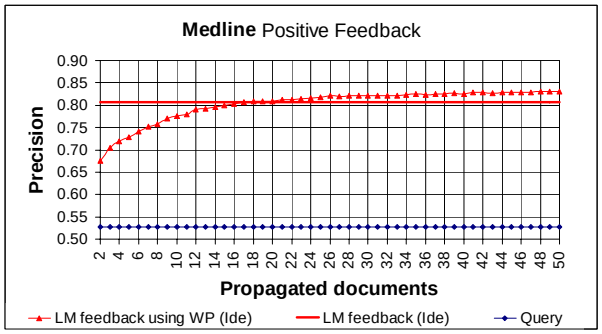

Fig. 21. Precision comparison for different methods.

Time Required by the Medline Experiments. Table 7 compares the times needed to perform RF for a single query in the collection, just for one iteration, using the LSI model and the weight propagation technique. The number of terms is assumed to increment at the same rate as the number of documents. The table shows the time required to perform relevance feedback in addition to the time required for the sorting needed to display the documents in the order of increasing weight magnitudes. The time required for query reformulation with WP is much less than that required for query reformulation with LSI. The time required by LM depends only on the constant k, which is the number of dimensions in LSI to which the dataset is reduced, and is therefore much less than the time required for either WP or LSI.

Table 6. Precision for the Medline test collection using LM using LSI.

		LM Query = 0.528	
		Positive feedback (Rocchio)	Positive feedback (Ide)
LM RF =		**0.636**	**0.807**
Weight propagation with different values of n'	2	0.706	0.676
	4	0.738	0.719
	6	0.751	0.741
	8	0.761	0.757
	10	0.772	0.776
	12	0.780	0.791
	14	0.783	0.795
	16	0.786	0.803
	18	0.789	0.808
	20	0.786	0.809
	22	0.789	0.813
	24	0.792	0.816
	26	0.792	0.821
	28	0.791	0.821
	30	0.790	0.822
	32	0.787	0.822
	34	0.789	0.823
	36	0.788	0.824
	38	0.788	0.825
	40	0.789	0.826
	42	0.787	0.829
	44	0.787	0.828
	46	0.786	0.829
	48	0.785	0.830
	50	0.784	0.830

WT18G. WP gave better precision than the baseline (no reformulation) when tested on the WT18G collection. When positive feedback based on Rocchio weighing was employed, the precision improved from 0.309436 to 0.366467, whereas WP improved the precision to 0.538477 when the weights were propagated to 27 or more documents. When negative feedback was considered, the precision was 0.385 409 for LM and 0.490 536 for WP when weights were propagated to 23 or more documents. The performance was thus improved by WP (see Fig. 22). Applying the Ide technique, WP also revealed better precision. Peak precision was 0.561470 with WP, and 0.464366 without WP for positive feedback, and 0.536490 with WP and 0.325 without 252 for negative feedback (see Figs. 22 and 23).

Table 7. Times required by query reformulation algorithms.

Matrix Size	LSI (miliseconds)			WP (miliseconds)		
	Relevance feedback	Sorting	Total	Relevance feedback	Sorting	Total
$10,000 \times 10,000$	191	10	201	0	20	20
$20,000 \times 20,000$	390	20	410	0	20	20
$30,000 \times 30,000$	601	20	621	0	31	31
$40,000 \times 40,000$	791	40	831	0	30	30
$50,000 \times 50,000$	1,011	50	1,061	0	50	50
$60,000 \times 60,000$	1,202	50	1,252	0	60	60
$70,000 \times 70,000$	1,412	80	1,492	0	70	70
$80,000 \times 80,000$	1,863	80	1,943	0	80	80
$90,000 \times 90,000$	2,123	90	2,213	0	80	80
$100,000 \times 100,000$	2,133	100	2,233	0	110	110
$110,000 \times 110,000$	2,233	110	2,343	0	120	120
$120,000 \times 120,000$	2,503	120	2,623	0	150	150
$130,000 \times 130,000$	2,654	140	2,794	0	150	150
$140,000 \times 140,000$	2,864	141	3,005	0	140	140
$150,000 \times 150,000$	3.124	171	3,295	0	350	350
$160,000 \times 160,000$	3,345	170	3,515	0	150	150
$170,000 \times 170,000$	3,575	190	3,765	0	180	180
$180,000 \times 180,000$	3,816	210	4,026	0	250	250
$190,000 \times 190,000$	4,557	450	5,007	0	160	160
$200,000 \times 200,000$	4,821	315	5,136	0	190	190

Table 8. Positive feedback precisions for Rocchio and Ide.

		Positive feedback (Rocchio)	Positive feedback (Ide)
LM RF =		**0.471**	**0.469**
Weight propagation with different values of n'	2	0.460	0.449
	4	0.482	0.466
	6	0.496	0.483
	8	0.500	0.491
	10	0.509	0.501
	12	0.513	0.505
	14	0.518	0.508
	16	0.521	0.514
	18	0.522	0.518
	20	0.522	0.519
	22	0.522	0.521
	24	0.525	0.525
	26	0.526	0.527
	28	0.527	0.530
	30	0.528	0.531
	32	0.531	0.534
	34	0.530	0.534
	36	0.531	0.534
	38	0.535	0.539
	40	0.535	0.539
	42	0.536	0.541
	44	0.537	0.542
	46	0.536	0.542
	48	0.537	0.546
	50	0.536	0.552

8. Conclusions

In this paper, we presented a new technique for query reformulation using weight propagation. The technique uses GLS with SVD and LM. In weight propagation, positive and negative weights are propagated to documents in a given vicinity within semantic and nonsemantic spaces. The proposed technique improves precision since the documents are treated as independent vectors instead of merging them into a single vector, as is the case with traditional approaches. In addition, the developed approach consumes less computation time since it inspects only nearby documents.

The proposed technique has a complexity of $O(qn')$, where q is the number of documents chosen as relevant or nonrelevant by the user, and n' is the maximum number of

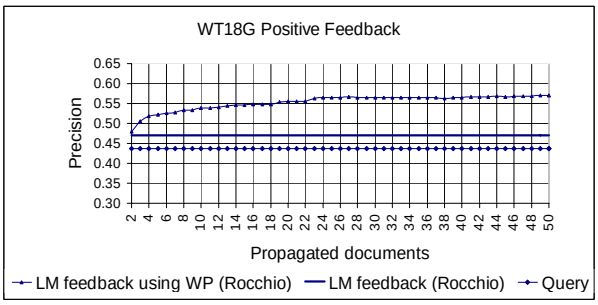

Fig. 22. Precision comparison for different methods for WT 18G.

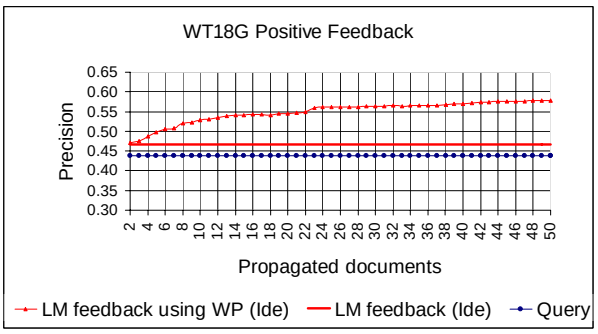

Fig. 23. Precision comparison for different methods for WT 18G.

Table 9. Precision for the WT18G test collection using LM.

		LM Query = 0.437	
		Positive feedback (Rocchio)	Positive feedback (Ide)
LM RF =		**0.470**	**0.467**
Weight Propagation with different values of n'	2	0.480	0.471
	4	0.519	0.488
	6	0.525	0.505
	8	0.534	0.521
	10	0.538	0.529
	12	0.541	0.534
	14	0.547	0.540
	16	0.548	0.542
	18	0.549	0.541
	20	0.555	0.546
	22	0.555	0.550
	24	0.564	0.561
	26	0.565	0.562
	28	0.564	0.562
	30	0.564	0.563
	32	0.564	0.565
	34	0.564	0.565
	36	0.564	0.566
	38	0.563	0.567
	40	0.565	0.570
	42	0.566	0.574
	44	0.568	0.575
	46	0.568	0.576
	48	0.569	0.577
	50	0.570	0.578

documents found in a linked list pointed to by an element in the MDS array. We showed that the value of n' is very small and so is q.

We believe that the results obtained from the proposed technique are better than the ones achieved by the LSI model. Alternative and more efficient data structures and algorithms for the construction of the MDS are currently being explored.

References

Baeza-Yates R. and Ribeiro-Neto B. (1999): *Modern Information Retrieval*. — New York: Prentice Hall.

Daniels J. and Rissland, E. (1995): *EXPRESS: A case based approach to intelligent information retrieval*. — Proc. SIGIR'95 Conf., Seattle, WA, USA 1995, pp. 31–43.

Dunlop M.D. (1997): *The effect of accessing non-matching documents on relevance feedback*. — ACM Trans. Inf. Syst., Vol. 15, No. 2, pp. 137–153.

Epp S.S. (1990): *Discrete Mathematics with Applications*. — London: Wadsworth Publishing Company.

Letsche T.A. and Berry M.W. (1997): *Large-scale information retrieval with latent semantic indexing*. — Inf. Sci., Vol. 9, No. 2, pp. 111–121.

Lopez-Pujalte C., Guerrero-Bote V.P., de Moya-Anegon F. (2002): *A test of genetic algorithms in relevance feedback*. — Inf. Process. Manag., Vol. 38, No. 7, pp. 793–805.

Ruthven I., Tombros A. and Jose J. (2001): *A study on the use of summaries and summary-based query expansion for a question-answering task*. — Proc. 23rd BCS European Annual Colloquium on *Information Retrieval Research*, Berlin, Germany, pp. 48-54.

Ruthven I., White R. and Jose J.M. (2001): *Web document summarization: A task-oriented evaluation*. — Proc. Int. Workshop *Digital Libraries*, Proc. 12-th Int. Conf. *Database and Expert Systems Applications*, (DEXA 2001), Munich, Germany, pp. 52–61.

Van R. (1979): *Information Retrieval*, 2nd Ed., London: McGraw Hill.

Voudouris C. and Tsang E. (1994): *Tunneling algorithm for partial CSPs and combinatorial optimization problems*. — Tech. Rep. No. CSM-213, Dept. of Computer Science, University of Essex, Colchester, UK.

Voudouris C. (1997): *Guided Local Search for Combinatorial Optimisation Problems*. — Ph.D. thesis, Dept. Computer Science, University of Essex, Colchester, UK.

Appendix A

The documents referred to in this paper are the following:

Doc01: Optimal clustering of relations in a database system

Doc02: Web document summarization – a task-oriented evaluation

Doc03: One term or two

Doc04: Detecting change in categorical data – mining contrast sets

Doc05: Criteria for testing exception-handling constructs in Java programs

Doc06: Evaluating document clustering for interactive information retrieval

Doc07: Effective information retrieval using genetic algorithms based matching function (adapted)

Doc08: Interactive information organization technique

Doc09: Using Dempster-Shafer's theory of evidence to combine aspects of information use

Doc10: Topic models for summarizing novelty

Doc11: Incorporating aspects of information use into relevance feedback

Doc12: An application of text mining – bibliographic navigator powered by extended association rules

Doc13: A study on the use of summaries and summary-based query expansion for a question-answer task

Doc14: Improved force-directed layouts

Doc15: Text categorization based on regularized linear classifications

Doc16: A probabilistic approach to crosslingual information retrieval

Doc17: Guided local search for combinatorial optimization problems

Doc18: A general language model for information retrieval

Doc19: Singular value decomposition on processor arrays

Doc20: Guided local search – an illustrative example

Doc21: Poisson mixtures

Received: 21 November 2005
Revised: 1 August 2006