Order-Based Fitness Functions for Genetic Algorithms Applied to Relevance Feedback

Cristina López-Pujalte, Vicente P. Guerrero-Bote
Facultad de Biblioteconomía y Documentación, Universidad de Extremadura, 06071 Badajoz, Spain. E-mail: clopez@alcazaba.unex.es; vicente@alcazaba.unex.es

Félix de Moya-Anegón
Facultad de Biblioteconomía y Documentación, Universidad de Granada, Campus Cartuja, Granada, Spain. E-mail: felix@goliat.ugr.es

Recently there have been appearing new applications of genetic algorithms to information retrieval, most of them specifically to relevance feedback. The evolution of the possible solutions are guided by fitness functions that are designed as measures of the goodness of the solutions. These functions are naturally the key to achieving a reasonable improvement, and which function is chosen most distinguishes one experiment from another. In previous work, we found that, among the functions implemented in the literature, the ones that yield the best results are those that take into account not only when documents are retrieved, but also the order in which they are retrieved. Here, we therefore evaluate the efficacy of a genetic algorithm with various order-based fitness functions for relevance feedback (some of them of our own design), and compare the results with the Ide dec-hi method, one of the best traditional methods.

Introduction

The genetic algorithms (Davis, 1991; Goldberg, 1989; Holland, 1992; Michalewicz, 1995) represent an Artificial Intelligence search technique that emulates the process of the evolution of species. These algorithms are especially suited to exploring complicated high-dimensional spaces. They have proven their usefulness in binary spaces, but they have also been used in real multidimensional spaces. The document spaces that derive from the application of the vector model are real high-dimensional spaces that are well suited to the use of these algorithms in exploring for improved solutions. For this reason, there has been a gradual but steady appearance of applications of genetic algorithms (GAs) to information retrieval (Belew, 1989; Chen, 1995; Chen, Chung, & Ramsey, 1998; Chen & Iyer, 1998; Cordon, Moya, & Zarco, 2000, 2002; Gordon, 1988a, b, Gordon, 1991; Horng & Yeh, 2000; Kraft et al., 1994, 1995, 1997; López-Pujalte, 2000; López-Pujalte, Guerrero Bote, & Moya Ang&eacuteon, 2002; Martín-Bautista, 2000; Martín-Bautista, Vila, & Larsen, 1999; Raghavan & Agarwal, 1987; Raghavan & Birchard, 1979; Robertson & Willet, 1994, 1995, 1996; Sanchez, 1994; Sanchez, Miyano, & Bracket, 1995; Sanchez & Pierre, 1994; Smith & Smith, 1997; Vrajitoru, 1997, 1998; Yang & Korfhage, 1992, 1993, 1994).

But how do these algorithms work? As input they have a population of individuals known as *chromosomes*, which represent the possible solutions to the problem. These are randomly generated, although if there is some knowledge available concerning the said problem, it can be used to create part of the initial set of potential solutions (Michalewicz, 1995). These individuals change (evolve) in successive iterations known as *generations*, by means of processes of *selection, crossover, and mutation.* These iterations halt when the system no longer improves, or when a preset maximum number of generations is reached. The output of the GA will be the best individual of the end population, or a combination of the best chromosomes of that population.

For each problem to be solved, one has to supply a *fitness function, f*, and indeed its choice is crucial to the good performance of the GA. Given a chromosome, the fitness function must return a numerical value that represents the chromosome's utility. This score will be used in the parent selection process so that the best-adapted individuals will have the greatest likelihood of being chosen. The fitness function must, therefore, be appropriate for the problem being dealt with, because the GA's effectiveness will to a large degree be determined by how faithfully the fitness function characterizes the function that is to be optimized.

Received July 5, 2001; revised February 8, 2002; accepted June 25, 2002

Most works on GAs applied to information retrieval use the vector space model. They fall mainly into three main groups according to their application: document indexing, clustering, and relevance feedback (Cordón, Moya, & Zarco, 1999). The last group is the most numerous, which is not surprising because GAs had been used previously to solve problems in which there is feedback from the environment. In particular, they were used to fit parameters for instance in oil-field simulations, market analysis, classification, etc. (Glover, 1987; Goldberg, 1989; Grefenstette, 1986, 1987; Hilliard & Liepins, 1987; Robertson, 1987).

We know that in the operations of information retrieval, most of the users, who do not know the details of the structure of the collection and of the retrieval environment, have difficulty in formulating a well-designed query for their immediate retrieval purposes. For this reason, the first retrieval operation must be seen as a test, as a trial run only, whose goal is to retrieve some useful elements from the collection. These initially retrieved elements may then be examined to assess their relevance, and then used to construct a new improved definition of the query to retrieve additional useful elements in subsequent searches: hence, the importance of the techniques that allow the user's queries to be adapted to obtain better results.

One of the most popular strategies for modifying database queries is relevance feedback. This process, introduced in the 1960s, is a controlled and automatic form of modifying the queries. The main idea is to use the information provided by previously retrieved documents, identified as relevant or irrelevant by the user, to adapt the query, so that more documents are retrieved like the relevant ones, and fewer than like the irrelevant ones (Salton & Buckley, 1990).

In applying GAs to relevance feedback, one can start from a set of possible solutions (queries) and then let them evolve under the algorithm attempting to achieve optimization. To guide this evolution of the possible solutions, one has to design fitness functions that evaluate how good these solutions are, using for the purpose feedback from the user.

In a recent exhaustive study that implemented and compared the different applications of GAs to relevance feedback (López-Pujalte, 2000; López-Pujalte et al., 2002), it was found that the design of the fitness function was fundamental for the GA to optimally modify the query. Indeed, the main differentiating feature each of those applications was which fitness function it used. The best results in the literature were obtained with functions that took into account not only the documents that were retrieved, but also the order in which they were retrieved, that is, these were fitness functions that scored not only whether the possible solution retrieves many relevant and few irrelevant documents, but also whether the relevant documents were given at the beginning of the list or at the end.

In the present work, we follow up on this finding by implementing a relevance feedback GA and running it with different order-based fitness functions. The results show that this type of function does indeed present an excellent behavior, unlike the rest of the functions implemented in the specialized literature, and one can achieve a considerable improvement in the original query with these GAs (some times the improvement was 127%). These results match or even surpass in some cases the Ide dec-hi method, which is possibly the best of the classical methods used in relevance feedback (Salton & Buckley, 1990).

The Genetic Algorithm and Fitness Functions Used

We used a GA that had been optimized and adapted for relevance feedback previously (López-Pujalte, 2000). We shall next describe the characteristics of this GA, chosen for having shown the best performance, and then we shall describe three different fitness functions, all based on the order of retrieval, which we used to guide the algorithm in the search process.

The Characteristics of the GA

Representation of the chromosomes. The vectors corresponding to the documents supplied as feedback are converted, using the procedure described by Chen (1995), into the chromosomes that our GA will work with. These chromosomes have the same number of genes (components) as there are terms with nonzero weights in the query and in the documents of the feedback. One first calculates the set of different terms contained in those documents and in the query, and the size of the chromosomes is equal to the number of terms in that set.

Example 1 (adapted from Chen, 1995): Consider the following representation of the user's query:

Q: Information, Retrieval, Indexing (three terms).

Let the following be the documents provided in the feedback and represented by their nonzero terms:

> DOC 1: Data, Retrieval, Database, Computer, Networks, Improvements, Information, Method, Multiple, Query, Relation (11 terms).
> DOC 2: Information, Retrieval, Storage, Indexing, Keyword (five terms).
> DOC 3: Artificial, Intelligence, Information, Retrieval, Systems, Indexing, Natural, Language, Processing (nine terms).
> DOC 4: Fuzzy, Set, Theory, Information, Retrieval, Systems, Indexing, Performance, Query (nine terms). *DOC 5: Information, Retrieval, System, Indexing, Stairs (five terms).*

The total set of terms representing all the documents is the following:

> {Information, Retrieval, Indexing, Data, Database, Computer, Networks, Improvements, Method, Multiple, Query, Relation, Storage, Keyword, Artificial, Intelligence, Systems, Natural, Language, Processing, Fuzzy, Set, Theory, Performance, Stairs}

The GA's chromosomes will therefore have a length of 25, which is the number of different terms with nonzero

weights in the set formed by the query and the five documents supplied in the feedback (in the present example).

Hence, the chromosomes that represent each document and the query will be the following (for the sake of simplicity, we shall assume a binary representation):

C1 = (1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
C2 = (0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0)
C3 = (0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0)
C4 = (0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 0)
C5 = (0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1)
Q = (1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

Observe that with the method described above, although the number of genes of the chromosomes is kept fixed for the whole population, it will vary (as will the population) according to the query that is being processed and the documents supplied in the feedback. There is therefore no preset chromosome size.

Population. Our GA receives an initial population consisting of the chromosomes corresponding to the relevant documents, to the irrelevant documents, and to the latter with their terms negated (in all cases, of course, we are referring to the feedback documents), and lastly to the query that some other method has supposedly optimized (which, as Davis, 1991, points out, sets it among the hybrid genetic algorithms).

Including the chromosomes corresponding to the negated irrelevant documents (i.e., chromosomes with all their components made negated), this is a direct consequence of the relevance feedback, in which the optimal query vector has to be brought ever closer to the relevant documents and farther away from the irrelevant documents. This is the reason why the classical methods in some way subtract the irrelevant document vectors from the original query vector. In this regard, one also sees in the work of Robertson and Willet (1996) that the GA improves markedly with the introduction of negative weights in the chromosomes. This is because the negative weights correspond to terms that appear frequently in the collection but infrequently in the relevant documents, i.e., which should not be selected to generate the optimal query.

Selection. The GA uses *simple random sampling* (Goldberg, 1989; Holland, 1992), as a selection mechanism. This consists of constructing a roulette wheel with as many slots as there are individuals in the population, and where the slot sizes are directly related to the individuals' fitness value. This is implemented by assigning to each individual a selection probability equal to its fitness value divided by the sum of the fitness values of all the individuals. With the roulette wheel implemented in this way, the selection process is to spin the wheel N times, each time selecting a chromosome for the intermediate population. In this way, the best chromosomes will naturally give rise on average to more copies, and the worst chromosomes to fewer copies.

Our GA also uses the strategy of élitism (De Jong, 1975) as a complement to the selection mechanism. If, after generating the new population, the best chromosome of the previous population is no longer present simply as the result of the whims of fortune, the worst individual of the new population is withdrawn, and the missing best individual is put back.

Genetic operators. We used one-point crossover as the crossover operator (Goldberg, 1989; Holland, 1992; Michalewicz, 1995). It is defined as follows:

> Given two parent chromosomes $C_1 = (a_1 \ldots a_m)$ and $C_2 = (b_1 \ldots b_m)$, one generates two offspring chromosomes $H_1 = (a_1, \ldots, a_i, b_{i+1}, \ldots, b_m)$ and $H_2 = (b_1, \ldots, b_i, a_{i+1}, \ldots, a_m)$, where i is a random number in the interval $[1, m - 1]$ and m is the length of the chromosome.

Mutation in our algorithm is implemented as a random process (Michalewicz, 1995). A real random number is generated in a given interval, in our case [0,1], and that number is taken as the new value for the gene that has to mutate. We also found that the GA's behavior was improved by including a procedure that normalized all the chromosomes of the population each time that the genetic operators were applied to them (even though they already came from normalized documents).

Control parameters. All the control parameters were fixed experimentally, carrying out many trials to obtain their optimal values. The control parameters, crossover probability p_c and mutation probability p_m, that led to the best results were considerably higher than those that are normally used, especially p_m ($p_c = 0.8$ and $p_m = 0.2$). The reason is that this helps to maintain the population's diversity, and to avoid the premature convergence of the algorithm. Also, we used 20 generations (trials with 10, 20, 40, 80, 100, and 200 generations were performed), because from that point on the population no longer improved.

Solution. The GA ends by returning as the solution both the best chromosome found during the process, and the centroid of those chromosomes in the final population that have a fitness value equal to the maximum found in that generation. This second method of finding the solution responds to the idea of avoiding the useless loss of potentially valuable information, because, if in the last generation there is more than one chromosome with the maximum fitness value, under what criterion would we select one and reject the rest? We implemented and evaluated both methods.

Formally, the centroid calculation is defined as follows:

$$t_{iR} = \frac{\sum_{j=1}^{T} f(\max) \cdot C_{ij}}{\sum_{j=1}^{T} f(\max)}$$

where t_{iR} is the term i of the resulting chromosome; T is the total size of the population; $f(\max)$ is a function that returns 1 if the chromosome's fitness is equal to the value of the population's maximum fitness, and zero otherwise; and C_{ij} is the term i of chromosome j.

The Fitness Functions

We ran the GA described above with different order-based fitness functions. In particular, as well as a fitness function of this type given in the literature (Horng & Yeh, 2000), we designed two new functions that reward the fact that the relevant documents appear at the top of the list of retrieved documents. We shall next describe each of these functions in detail.

Fitness 1. This fitness function, due to Horng and Yeh (2000), is very innovative. As well as taking into account the number of relevant and of irrelevant documents, it also takes account of the order of their appearance, because it is not the same that the relevant documents appear at the beginning or at the end of the list of retrieved documents.

The said fitness function (Chang & Hsu, 1999; Kwok, 1997) is constructed as follows: One calculates the similarity of the query vector with all the documents (using the scalar product), and sorts the documents into decreasing order of similarity. Finally, one calculates the fitness value of the chromosome with the following formula:

$$F = \frac{1}{|D|} \sum_{i=1}^{|D|} \left(r(di) \sum_{j=1}^{|D|} \frac{1}{j} \right)$$

where $|D|$ is the total number of documents retrieved, and $r(d)$ is the function that returns the relevance of document d, giving a 1 if the document is relevant and a 0 otherwise. We shall refer to this fitness function as *fitness 1*.

One sees that the function represents an ingenious method of summing fractions whose denominator indicates the positions of the documents, with the relevant documents producing more fractions depending on the position that they occupy.

Fitness 2. With this fitness function, the GA will include an additional parameter A, which will determine the values of the factors to be used by the said function. The function will also compare the indicated chromosome and the feedback documents, using the cosine as the measure of similarity. After sorting the documents into relevance order, one accumulates for all the retrieved documents (with similarity greater than zero):

$$\frac{1}{A} \cdot \left(\frac{(A-1)}{A} \right)^{(\text{pos}-1)}$$

where pos is the position of the document under consideration.

Fitness function 3

```
Begin
  For each of the feedback documents, Repeat
    begin
      Calculate similarity s between the chromosome and the document;
        If s >0
          Store result;
    end
  Sort stored results by decreasing order of similarity;
  /*** mean interpolated precision method for intervals of recall ***/
  Initialize interval to the greatest interval;
  Initialize i to the last position of the sorted document table;
  P = 0;
  While i > 0, Repeat
    begin
      Calculate precision P_I for position i of the retrieval list;
      Calculate recall R_I for position i of the retrieval list;
      If P_I > P
        P = P_I;
      While R_I < interval
        begin
          Store P;
          Decrement interval;
        end
      Decrement i ;
    end
  Process remaining intervals;
  Find the arithmetic mean of the stored precisions (P);
  Return (meanPrecisions);
End
```

FIG. 1. Algorithm to calculate the fitness function 3.

This accumulation has an important feature: it will be positive when the document is relevant, and negative otherwise.

Lastly, the total accumulated result is multiplied by the recall of the retrieval to give the value returned by the function as the chromosome's fitness.

One sees that this function, as is the case with the other order-based functions, needs no threshold or document cut-off. Instead, it considers all the documents that were retrieved, i.e., with a similarity greater than zero.

As was the case with the previous function, this method guarantees that each classification will have a different value, i.e., although the number of relevant documents retrieved is the same, their order of appearance directly influences the calculation.

Fitness 3. This function, as also was the previous function, is of our own design, and consists in finding the mean precision in nine recall intervals, 0.1, 0.2, . . . , 0.9, but performing an interpolation process to eliminate any ambiguities.

The process followed by this fitness function is illustrated in detail in the pseudocode of the algorithm in Figure 1.

As one sees, this function calculates the chromosome's similarity with the feedback documents by calling a function that uses a cosine similarity measure. The results are

stored in an auxiliary table, as long as the similarity of the document with the chromosome is greater than zero. This table is sorted into decreasing order of similarity. Then begins the procedure to find the mean interpolated precision in the nine recall intervals. This procedure is basically to assign as an interval's precision the highest precision of all the points of recall that precede it to the right.

Before returning, the function checks whether the precisions of all the intervals have been found. If not, the rest of the intervals are assigned the last precision calculated (*treatment of remaining intervals*).

Lastly, the mean of all the stored precisions is calculated, one for each interval, and this mean will be the fitness value of the given chromosome.

Obviously, these stored fixed-recall-interval precisions would change if the relevant documents appeared in a different order. Hence, for a new order and the resulting different precisions, the mean of the precisions would also be different.

The complexity of the three fitness functions is high, and fairly similar, and, as there will never be a great number of feedback documents, it will not significantly affect processing times.

Experiment and Evaluation

As we mentioned above, one of our prime objectives is to check the supposedly good behavior of different GAs that use order-based fitness functions, as well as making comparisons with some traditional method, in particular the Ide dec-hi method, which had given the best results in the study of relevance feedback by Salton and Buckley (1990). (In this last study, six of the traditionally most often used methods were examined in depth.)

The Ide dec-hi method (Ide, 1971), which we have taken as the model against which to test our feedback experiments, is highly intuitive and very simple, at the same time as being amazingly effective. It consists simply in adding directly to the weights of the original query those of all the relevant documents of the set of documents supplied for feedback, and subtracting from them the weights of the first irrelevant document obtained in the retrieval that belongs to the said set.

Formally, the query vector is reformulated as follows:

$$Q' = Q + \sum_{\text{all relevant}} D_i - S$$

where Q is the original query vector, D_i is the vector of the relevant document i, and S is the vector of the top-ranked irrelevant document.

Test Collection and Document Vectorization

To perform our experiments, we had to generate a test database. We created this from one of the test collections that is best known and of greatest prestige amongst investigators, the *Cranfield* collection (López-Pujalte, 2000; López-Pujalte et al., 2002; Robertson & Willet, 1996; Salton & Buckley, 1990; Yang & Korfhage, 1994). This consists of 1,398 documents on diverse aspects of aeronautical engineering, and 225 queries for which the relevance judgements are known. One of the main reasons for choosing this collection is that it has been used on a great many occasions for feedback and GA experiments, so that it offered the possibility of comparing results.

The number of documents that were finally fixed for the feedback implementation was 15 (we also performed trials with 10, 20, and 25 documents), i.e., for each query the first 15 documents retrieved were examined to determine their relevance, and this information was supplied to the algorithm as feedback.

It was necessary to make a selection of the 225 queries associated with this collection, so as to be left with those queries that were suitable for testing the relevance feedback technique that the present work is studying, because not all the queries in the collection were suitable. Thus, for instance, queries that retrieve all the relevant documents among the first 15 are not suitable, because there are no documents left that are interesting to retrieve. Likewise, neither are queries appropriate that fail to retrieve any relevant document among these first 15, because they can then provide no information for feedback. In particular, we selected for the evaluation a group of queries (33) that had at least three relevant documents retrieved among the first 15, and at least 5 relevant documents yet to be retrieved.

First, to determine which terms we would use to describe the documents of the collection, we performed the following steps (Guerrero & Moya Anegón, 2001; Guerrero, Moya Anegón, & Herrero Solana, 2002):

(1) Extract all the words from each of the documents.
(2) Eliminate the stop words, using a list of stop words generated from the frequency dictionary of Kuccera and Francis (1967), as we had also done in other studies (Guerrero et al., 2001).
(3) Stem the remaining words. For this purpose we used the *Porter Stemmer*, which is the most commonly used stemmer in English (Frakes & Baeza-Yates, 1992; Porter, 1980).

The final number of terms from the documents of the Cranfield collection after completion of the above process was 4,307, so that we shall be working with 1,398 document vectors of 4,307 components.

Next, to assign the weights, we used the scheme described in Salton and Buckley (1990) to equiparate in as far as possible our experiments with theirs. The formula is:

$$a_{ij} = \frac{\left(0.5 + 0.5\frac{tf_{ij}}{\max tf}\right) \cdot \log\frac{N}{n_i}}{\sqrt{\left(0.5 + 0.5\frac{tf_{ij}}{\max tf}\right)^2 \left(\log\frac{N}{n_i}\right)^2}}$$

where a_{ij} is the weight assigned to the term t_j in the document D_i; tf_{ij} is the number of times that the term t_j appears

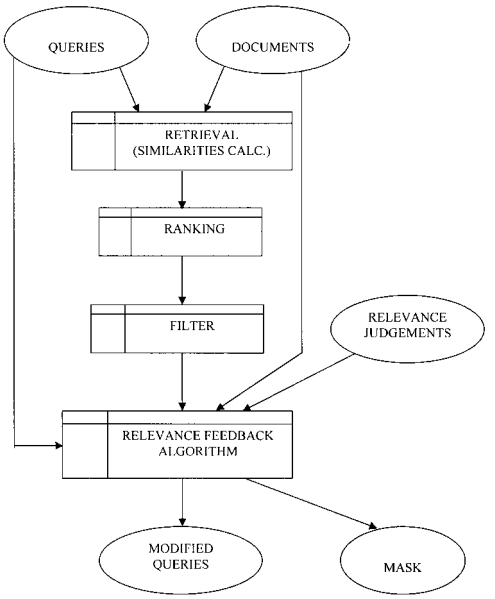

FIG. 2. Data flow chart corresponding to the experimental design.

in the document D_i; n_j is the number of documents indexed by the term t_j; and N is the total number of documents in the database.

Last, we normalized the document vectors, dividing them by their Euclidean norm, because according to the study of Noreault, McGill, and Koll (1981), the best similarity measures are those that make angular comparisons between vectors.

A similar process is then carried out on the queries associated with the collection. This gives the normalized vectors corresponding to the queries, which we shall be attempting to optimize with relevance feedback.

Experimental Design

The scheme of the experiment to implement relevance feedback by means of the different methods (our GAs and the Ide dec-hi method, which will be used for comparison) is very simple (López-Pujalte, 2000):

(1) Each of the collection's queries is compared with all of its documents, using the cosine similarity measure. This gives a list of the similarities of each query with all the collection's documents.
(2) This list is sorted into decreasing order of degree of similarity.
(3) The normalized document vectors corresponding to the top 15 documents of the list, together with their relevance judgements and the normalized query vector, are presented as input to the algorithm responsible for optimizing the query.
(4) The program will also generate as output a masking file. This contains for each query all the documents that are not to be considered in the evaluation process (they will be the first 15 that have been used in modifying the queries), because we shall be following the residual collection method as described by Salton & Buckley (1990).

Figure 2 shows the flow chart corresponding to this process.

It should be noted that, while the Ide dec-hi method only has to be run once, because it will always yield the same output; this is not the case with the GAs, because their random nature means that they give different (while similar) outputs on each run. The final results presented are the maximum values obtained in all the trials carried out for each GA.

Evaluation

We next evaluated the retrieval results by means of the classical measures of *recall and precision*. We calculated the interpolated precision at fixed recall intervals (of width 0.1) as described by Salton and McGill (1983), and the average precision at all the recall points as is done by Salton and Buckley (1990) in their feedback study, so as to be able to compare the different systems.

We also used the *residual collection* method (Chang, Cirillo, & Razon, 1971), in which all the documents previously seen by the user (whether relevant or not) are extracted from the collection, and both the initial and the revised queries are evaluated on this residual collection. This is done because the relevance feedback operation has to be judged on its ability to retrieve new documents that have not been originally examined by the user. Indeed, this is the standard method of evaluating relevance feedback, because its estimate is more realistic and unbiased (Baeza-Yates & Ribeiro-Neto, 1999; Harman, 1992; Salton & Buckley, 1990).

Results

The final results of our experiment are listed in Table 1, which gives the mean precision in three recall intervals (0.25, 0.5, and 0.75, representing low, medium, and high levels of recall, respectively) for each of the algorithms that were implemented, as well as the percentage improvement over the initial unoptimized query. We also represent these results graphically in Figure 3.

TABLE 1. Results of different relevance feedback methods for the Cranfield collection.

	Best		Centroid	
Method	Mean	Improve	Mean	Improve
No feedback	0.098			
Ide (dec-hi)	0.218	120.8%		
GAs				
GA with fitness 1	0.218	120.8%	0.220	123.6%
GA with fitness 2	0.218	120.8%	0.210	114.2%
GA with fitness 3	0.218	120.8%	0.224	127.2%

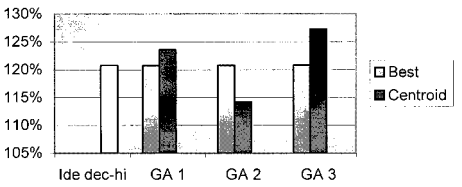

FIG. 3. Improvement percentages with respect to the original query as obtained by different relevance feedback methods for the Cranfield collection.

As one sees in the table, the results after applying the relevance feedback technique present a very major improvement (around 120%) with respect to the original query, whether the implementation is via the classical Ide dec-hi method or with any of our GAs. This is not always the case when using GAs, which employ other types of fitness function (López-Pujalte, 2000; López-Pujalte et al., 2002).

As expected, the GAs behave very satisfactorily, because they all use order-based fitness functions. Two of them (*GA 1* and *GA 3*) improve on the best of the traditional relevance feedback methods in information retrieval, with the third GA equalling the traditional method.

One thus sees that the relevance feedback technique that yields the best results is a GA, in particular that which uses our fitness function 3 that we described in detail above.

It has to be noted that the performance of these order-based functions appears to be inversely proportional to the decay rate, at least in the present case (fitness 1 decays geometrically with respect to order, fitness 2 exponentially, and fitness 3 more slowly than geometrically). This may be a very useful clue to how future fitness functions should be implemented for this type of genetic algorithm.

With respect to the form of calculating the solution, as we noted above, there are two alternatives, as is seen in the table: (1) The *best* solution: the best chromosome found throughout the process; and (2) the *centroid* solution: the centroid of the final population's chromosomes whose fitness value is the maximum value calculated for this population.

As one sees from Table 1, in two cases the GA yields better results with the *centroid* method. The other case, *fitness 2*, presents the better results with the first (the *best*) method. One notes that the two GAs that improve on the Ide dec-hi method do so when using the centroid method. This method is therefore very promising for functions of this type, which give especial importance to the order in which documents are retrieved.

In sum, the improvement achieved with feedback in our experiments was from 120.8% to 127.2% using the Cranfield collection. This even surpassed the 120.8% improvement obtained with the Ide dec-hi method. As is shown in Figure 3, the greatest improvement (127.2%) was attained with the GA using fitness function number 3 and the centroid method to calculate the solution. The GA surpasses the classical method by almost 7%. Although this improvement is small, it is nonetheless statistically significant under both Student's *t*-test and the sign test.

With respect to the computation times of the different methods, obviously the traditional Ide dec-hi method is far less time consuming. The times of the GAs are, however, still quite acceptable, around 2 seconds per query. No significant differences in time were observed between using one or another fitness function. The computing system used was a Pentium III, at 700 MHz, with 256 MB of RAM and 26 GB of hard disk space.

Conclusions

The document spaces derived from the application of the vector model are very high-dimensional real spaces. Because GAs have proven their effectiveness in exploring large complicated spaces, they can be used to search the said document spaces and indeed are found to lead to good results. The GAs find a major field of application in information retrieval.

Given the relative ease of implementing relevance feedback (whether by GAs or by more traditional methods), and the excellent results that are achieved, any information retrieval system worth considering should incorporate a feedback module.

To guide the evolution of the possible solutions in genetic methods, one has to design fitness functions that evaluate the goodness of the intermediate solutions, using the information provided by feedback from the user. It is in these fitness functions where experiments most differ from each other. They are the key to achieving a good level of improvement, because, as we have observed, whether the exploration results in success or utter failure depends on these functions.

We here tested three different functions that had in common the evaluation not only of the retrieved documents themselves but also the order in which they were retrieved, because earlier work had clearly indicated that this was a good design alternative.

The results showed that these GAs allow one to considerably improve the original query using relevance feedback (in a single iteration). The improvement was from 120.8% to 127.2% using the Cranfield collection (the greatest improvement being with the GA using fitness function 3). This even surpassed the 120.8% improvement obtained with the Ide dec-hi method, which is one of the best classical methods used in relevance feedback according to the work of Salton and Buckley (1990).

As we had expected, these results improved on those obtained with GAs reported in the literature that use other fitness functions (López-Pujalte, 2000, López-Pujalte et al., 2002). They thus allow one to state that the retrieval order is a fundamental factor to take into account in implementing these functions to guide the algorithm in performing the relevance feedback task, and thereby optimize the original query. In the present area, therefore, one can conclude that it is desirable to use fitness functions that value not only whether the possible solution retrieves many relevant documents and few irrelevant documents, but also whether the

relevant documents are at the top of the retrieval list or at the end.

As we noted above, there seems to be a pointer to the direction that future work should follow in that the performance of the three tested fitness functions was inversely proportional to the rate of decay with respect to order.

Acknowledgments

This work was financed by the Junta de Extremadura-Consejería de Educación Ciencia & Tecnología and the Fondo Social Europeo, as part of research project IPR99A047.

References

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). Modern information retrieval. Essex, UK: Addison-Wesley.

Belew, R.K. (1989). Adaptive information retrieval. Proceedings of the association for computing machinery special interest group on information retrieval (ACM/SIGIR) 12th annual international conference on research and development in information retrieval, 25–28 June 1989 (pp. 11–20). Cambridge, MA. New York: ACM, Inc.

Chang, C.-H., & Hsu, C.-C. (1999). The design of an information system for hypertext retrieval and automatic discovery on WWW. Ph.D. Thesis,. Department of CSIE. National Taiwan University.

Chang, Y.K., Cirillo, C., & Razon, J. (1971). Evaluation of feedback retrieval using modified freezing, residual collection, and test and control groups. In G. Salton (Ed.), The smart retrieval system—Experiments in automatic document processing (pp. 355–370). Englewood Cliffs, NJ: Prentice Hall Inc.

Chen, H. (1995). Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms. Journal of the American Society for Information Science, 46(3), 194–216.

Chen, H., & Iyer, A. (1998). A machine learning approach to inductive query by examples: An experiment using relevance feedback, ID3, genetic algorithms, and simulated annealing. Journal of the American Society for Information Science, 49(8), 693–705.

Chen, H., Chung, Y., & Ramsey, M. (1998). A smart itsy bitsy spider for the web. Journal of the American Society for Information Science, 49(7), 604–618.

Cordón, O., Moya, F., & Zarco, M.C. (1999). Breve estudio sobre la aplicación de los algoritmos genéticos a la recuperación de la información. IV Congreso ISKO (Granada) (pp. 179–186).

Cordón, O., Moya, F., & Zarco, M.C. (2000). A GA-P algorithm to automatically formulate extended boolean queries for a fuzzy information retrieval system by means of GA-P techniques. Mathware & Soft Computing, 7(2–3), 309–322.

Cordón, O., Moya, F., & Zarco, M.C. (2002). A new evolutionary algorithm combining simulated annealing and genetic programming for relevance feedback in fuzzy information retrieval systems. Soft Computing, 6(5), 308–319.

Davis, L. (Ed.). (1991). Handbook of genetic algorithms. New York: Van Nostrand Reinhold.

De Jong, K.A. (1975). An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan.

Frakes, W.B. (1992). Stemming algorithms. In: Frakes, W.B., & Baeza-Yates, R. (Eds.), Information retrieval: Data structures & algorithms. Englewood Cliffs, NJ: Prentice Hall.

Glover, D.E. (1987). Solving a complex keyboard configuration problem through a generalized adaptive search. In: Davis, L. (Ed.) Genetic algorithms and simulated annealing (pp. 12–31). San Mateo, CA: Morgan Kaufmann.

Goldberg, D.E. (1989). Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley.

Gordon, M.D. (1988a). Probabilistic and genetic algorithms for document retrieval. Communications of ACM, 31(10), 1208–1218.

Gordon, M.D. (1988b). The necessity for adaptation in modified Boolean document retrieval systems. Information Processing and Management, 24(3), 339–347.

Gordon, M.D. (1991). User-based document clustering by redescribing subject descriptors with a genetic algorithms. Journal of the American Society for Information Science, 42(5), 311–322.

Grefenstette, J.J. (1986). Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, SMC-16(1), 122–128.

Grefenstette, J.J. (1987). Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), Genetic algorithms and simulated annealing (pp. 42–60). Los Altos: Morgan Kaufmann Publishers.

Guerrero Bote, V.P., & Moya Anegón, F. (2001). Reduction of the dimension of a document space using the fuzzified output of a Kohonen network. Journal of the American Society for Information Science and Technology, 52, 1234–1241.

Guerrero Bote, V.P.; Moya Anegón, F., & Herrero Solana, V. (2002). Document organization using Kohonen's algorithm. Information Processing & Management, 38, 79–89.

Harman, D.K. (1992). Relevance feedback and other query modification techniques. In: W.B. Frakes & R. Baeza-Yates (Eds.), Information retrieval: Data structures and algorithms (pp. 241–263). Englewood Cliffs, NJ: Prentice Hall.

Hilliard, M.R., & Liepins, G.E. (1987). A classifier-based system for discovering scheduling heuristics. Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms (pp. 231–235).

Holland, J.H. (1992). Adaptation in natural and artificial systems (2nd ed.). Cambridge, MA: MIT Press.

Horng, J.-T., & Yeh, C.-C. (2000). Applying genetic algorithms to query optimization in document retrieval. Information Processing and Management, 36, 737–759.

Ide, E. (1971). New experiments in relevance feedback. In G. Salton (Ed.), The SMART retrieval system (pp. 337–354). Englewood Cliffs, NJ: Prentice-Hall.

Kraft, D.H., Petry, F.E., Buckles, B.P., & Sadasivan, T. (1994). The use of genetic programming to build queries for information retrieval. Proceedings of IEEE Symposium on Evolutionary Computation. Orlando, FL.

Kraft, D.H., Petry, F.E., Buckles, B.P., & Sadasivan, T. (1995). Applying genetic algorithms to information retrieval systems via relevance feedback. In P. Bosc & J. Kacprzyk (Eds.), Fuzzy sets and possibility theory in database management systems, studies in fuzziness series (pp. 330–346). Heidelberg, Germany: Physica-Verlag.

Kraft, D.H., Petry, F.E., Buckles, B.P., & Sadasivan, T. (1997). Genetic algorithms for query optimization in information retrieval: Relevance feedback. In E. Sanchez, T. Shibata, & L.A. Zadeh (Eds.), Genetic algorithms and fuzzy logic systems. Soft computing perspectives (pp. 155–173). Singapore: World Scientific.

Kucera, H., & Francis, N. (1967). Computational analysis of present-day American English. Providence, RD: Brown University Press.

Kwok, K.L. (1997). Comparing representations in Chinese information retrieval. ACM/SIGIR (pp. 34–41), Philadelphia, PA.

López-Pujalte, C. (2000). Algoritmos genéticos aplicados a la retroalimentación por relevancia. PhD Thesis, Library and Information Science Faculty, University of Granada.

López-Pujalte, C., Guerrero Bote, V.P., & Moya Anegón, F. (2002). A test of genetic algorithms in relevance feedback. Information Processing and Management, 38(6), 795–807.

Martín-Bautista, M.J. (2000). Modelos de computación flexible para la recuperación de información. PhD Thesis, Computer Science Faculty, University of Granada.

Martín-Bautista, M.J., Vila, M.A., & Larsen, H.L. (1999). A fuzzy genetic algorithm approach to an adaptive information retrieval agent. Journal of the American Society for Information Science, 50(9), 760–771.

Michalewicz, Z. (1995). Genetic algorithms + data structures = evolution programs. Berlin: Springer-Verlag.

Noreault, T., McGill, M., & Koll, M.B. (1981). A performance evaluation of similarity measures, document term weighting schemes and representation in a Boolean environment. Information Retrieval Research. London: Butterworths.

Porter M.F. (1980). An algorithm for suffix stripping. Program 14(3), 130–137.

Raghavan, V.V., & Agarwal, B. (1987). Optimal determination of user-oriented clusters: An application for the reproductive plan. Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms and their applications (pp. 241–246).

Raghavan, V.V., & Birchard, K. (1979). A clustering strategy based on a formalism of the reproduction process in natural systems. Proceedings of the second International ACM/SIGIR Conference on Information Retrieval (pp. 10–22).

Robertson, A.M., & Willett, P. (1994). Generation of equifrequent groups of words using a genetic algorithms. Journal of Documentation, 50(3), 213–232.

Robertson, A.M., & Willett, P. (1995). Use of genetic algorithms in information retrieval. British Library. Research and Development Department. BLRD Report 6201.

Robertson, A.M., & Willett, P. (1996). An upperbound to the performance of ranked-output searching: Optimal weighting of query terms using a genetic algorithm. Journal of Documentation, 52(4), 405–420.

Robertson, G. (1987). Parallel implementation of genetic algorithms in classification systems. In L. Davis (Ed.), Genetic algorithms and simulated annealing (pp. 129–149). San Mateo, CA: Morgan Kaufmann.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science, 41(4), 288–297.

Salton, G., & McGill, M.J. (1983). Introduction to modern information retrieval. New York: McGraw-Hill.

Sanchez, E. (1994). Fuzzy logic and genetic algorithms in information retrieval. Proceedings of the third international conference on fuzzy logic, neural networks and soft computing (pp. 29–35).

Sanchez, E., & Pierre, P. (1994). Fuzzy logic and genetic algorithms in information retrieval. Third international conference on fuzzy logic, neural networks and soft computing, Izuaka, Japan (pp. 29–35).

Sanchez, E., Miyano, H., & Brachet, J. (1995). Optimization of fuzzy queries with genetic algorithms. Application to a data base of patents in Biomedical Engineering. VI IFSA Congress, Sao Paulo, Brazil (vol. II, pp. 293–296).

Smith, M.P., & Smith, M. (1997). The use of genetic programming to build boolean queries for text retrieval through relevance feedback. Journal of Information Science, 23(6), 423–431.

Vrajitoru, D. (1997). Apprentissage en recherche d'informations. Doctoral thesis University of Neuch,tel, Faculty of Science.

Vrajitoru, V. (1998). Crossover improvement for the genetic algorithm in information retrieval. Information Processing and Management, 34(4), 405–415.

Yang, J.J., & Korfhage, R.R. (1992). Adaptive information retrieval systems in vector model. Proceedings of the symposium on document analysis and information retrieval (pp. 134–150), Las Vegas, NV.

Yang, J.J., & Korfhage, R.R. (1993). Query optimization in information retrieval using genetic algorithms. In Proceedings of the fifth international conference on genetic algoritms (pp. 603–611), Urbana, IL.

Yang, J.J., & Korfhage, R. (1994). Query modification using genetic algorithms in vector space models. International Journal of Expert Systems, 7(2), 165–191.