Information Retrieval, 1, 175–192 (1999)

Genetic Approach to Query Space Exploration

M. BOUGHANEM bougha@irit.fr
C. CHRISMENT chrisme@irit.fr
L. TAMINE
IRIT SIG Université Toulouse III, 118, Route de Narbonne, 31062 Toulouse, France

Received July 8, 1998; Revised February 5, 1999

Abstract. This paper describes a genetic algorithm approach for intelligent information retrieval. The goal is to find an optimal set of documents which best matches the user's needs by exploring and exploiting the document space. More precisely, we define a specific genetic algorithm for information retrieval based on knowledge based operators and guided by a heuristic for relevance multi-modality problem solving. Experiments with TREC-6 French data and queries show the effectiveness of our approach.

Keywords: information retrieval, genetic algorithm, relevance feedback

1. Introduction

The explosive growth of the Internet and other sources of information access have prompted the rapid proliferation of large information collections. Therefore, the effort required to retrieve relevant information has become significantly more difficult. Several investigations continue to deal with this problem (Harman 1997). These investigations are concerned with term weighting and the popular relevance feedback technique.

Relevance feedback is still the main technique for query modification and therefore a solution to search improving. This technique has been investigated for more than twenty years in various information retrieval models, such as the probabilistic model (Robertson and Sparck Jones 1976, Robertson and Walker 1997, Haines and Croft 1993) and the vector space model (Salton 1970). In theory, relevance feedback is based on automatically changing the set of query terms as well as the weights associated to these terms according to documents retrieved and judged during the initial search. However, even though relevance feedback significantly improves the performance of the search, no information retrieval process can be expected to recall all the possible relevant documents for all searches.

Recently, there has been a growing interest in machine learning techniques, particularly the neural approach (Kwok 1989, Wilkinson and Hingston 1991, Wong et al. 1993, Boughanem and Soule-Dupuy 1997b) and the use of evolution algorithms that rely on analogies to natural processes. The genetic algorithm approach developed by Holland (1975) and based on both the principles of natural evolution processes and Darwinian survival theory has proved its efficiency as a stochastic optimization method. GA have been applied in the information retrieval domain with the purpose of optimizing document descriptions (Gordon 1988) and improving query formulation (Yang and Korfhage 1993, Chen 1995).

This is precisely the context of our work. Our goal is to build a GA that can find an optimal set of documents which best match the user's need by exploring different regions of the document space simultaneously. We have been attracted by the use of GA to handle the information retrieval process for the following reasons:

- The document base represents a high dimensional space. As genetic algorithms have been shown to be powerful search mechanisms due to their robust nature and quick search capabilities, they seem to be suitable for information retrieval. Thanks to their inherent properties of implicit parallelism, GA could perform the search in different regions of the document space simultaneously.
- Contrary to the classical retrieval models, the GA manipulates a population of queries rather than a single query. Each query may retrieve a subset of relevant documents that can be merged. We believe that this is more efficient than using a hill-climbing search based on a single query.
- The classical methods of query expansion manipulate each term independently of each other. But several experiments have already shown that the terms occur in the documents by groups. The GA would contribute in this case to preserve useful information links representing a set of terms occurring in the relevant documents.
- The classical methods of relevance feedback are not efficient when no relevant documents are retrieved with the initial query. In contrast, the probabilistic exploration induced by the GA allows the exploring of new zones in the document space independently from the initial query.

As mentioned above, some works have applied the GA to information retrieval (Gordon 1988, Yang and Korfhage 1993, Kraft et al. 1995, Chen 1995), to either document descriptors or query optimization. These are based on pure GA operators, called in this paper blind operators, and do not take advantage of the research in information retrieval. Our model, however, is characterized by its "knowledge enhanced operators" (knowledge here signifies using some known relevant information issued from the information retrieval domain) and the use of a niching technique for our GA.

The main aim of the work reported in this paper is to show the effectiveness of our GA model in improving search performance. This GA model is characterized by its knowledge-enhanced operators that explore several regions of the document space simultaneously. However, we must mention that we are not comparing our GA to other relevance feedback methods such as those based on Rocchio for example (Rocchio 1971, Salton and Buckley 1990). Our goal is to show a different method and how the GA can be used for this purpose. With this aim we first present in this paper a brief overview of genetic algorithms and the related works in IR. We then describe our GA model. The last section concerns the experiments performed on the TREC-6 French database (Harman 1997) and a discussion of the results.

2. Genetic Algorithm and Information Retrieval

Genetic algorithms are stochastic optimization methods developed by Holland (1975) and based on both the principles of natural evolution processes and Darwinian survival theory.

In such algorithms, a population of potential solutions is renewed at each generation by encouraging the reproduction of the fittest ones. After a number of generations, the program converges to the best individual representing the optimal solution.

2.1. Basic Concepts of Genetic Algorithm

Individual and Population. An individual, also called a chromosome, is the representation of a potential solution to the problem. It is implemented as a data structure where each component represents a *gene*. The value and the position of a gene are respectively named *allele* and *locus*. All the individuals manipulated by the algorithm at one generation (one iteration of the GA) compose a population; this population evolves through the generations.

Fitness Function. This function evaluates the fitness of each individual.

Genetic Operators. Some individuals of each generation undergo transformations by means of genetic operators to build new solutions that are hopefully more fit. GA applies three transformations: selection, crossover and mutation.

- Selection: consists of the reproduction of a new population according to the probability distribution based on the fitness values. Individuals that are more fit have better chances to get more copies selected in the next generation.
- Crossover: is a recombination operator applied with a given probability P_c. Each pair of individuals selected for the crossover operation exchange part of their information delimited by a crossing point which is randomly selected along the chromosome.
- Mutation: It is an unary transformation operator applied with a given probability P_m. For each individual of the crossovered population and for each gene, a random number r from the range of [0..1] is generated, if ($r < P_m$) the gene is mutated.

Evolution. The resolution of an optimization problem using a genetic algorithm requires the modelization of a generic solution and a fitness function. After the *initialization of the population*, follows *selection*, *crossover* and *mutation* operators in order to set up the next generation. The remaining iterations are simply *cyclic repetitions of the above steps* until the program reaches a predetermined number of generations or converges. Several theoretic studies have already proved the effective convergence of genetic algorithms (Holland 1975, Radcliffe 1991). Furthermore, these studies have suggested the use of heuristics to improve the control on the genetic exploration process. We note niching and speciation techniques (Goldberg 1994), knowledge based operators and adaptive control methods (Goldberg 1994, Grefenstette 1995, Sebag and Schoenauer 1996).

Main Properties. The main properties of a genetic algorithm are the following:

- implicit parallelism: When manipulating an n size population, GA explores simultaneously a number of directions running to n^3. This expresses the fundamental properties of implicit parallelism demonstrated by Goldberg (1994).

- resolution of exploration/exploitation dilemma: The genetic programming resolves efficiently the exploration/exploitation dilemma by allowing an exponentially increasing number of copies of the fitter individuals. Therefore, encouraging exploration in good directions.
- non-optimality: A genetic algorithm does not guarantee to reach the global optima. However, experiments showed that it generally returns near optimal solutions.

2.2. Related Works in GA and IR

The development of scheme theory invented by Holland (1992) and some theoretical studies in GA (Ankenbrandt 1990), have attracted scientists from several research areas. Some works and studies have been done in the IR area and we discuss a selection of these below.

Gordon (1988) adopted GA to derive better descriptions of documents. Each document is assigned N descriptions; each description is a set of indexing terms. Genetic operators and relevance judgement are applied to the descriptions in order to build the **best** document description. The author showed that the GA produces better document descriptions than the ones generated by the probabilistic model. Redescription improved the relative density of co-relevant documents by 39.74% after twenty generations and 56.61% after forty generations. Gordon exploited these results and defined a classification method (Gordon 1988) based on clustering the relevant documents for a specific query.

Yang and Korfhage (1993) proposed a GA to query optimization by reweighting the document term indexing without query expansion. They used a selection operator based on a stochastic sample, a blind crossover at two crossing points, and a classical mutation to renew the population of queries. The experiments showed that the queries converge to their relevant documents after six generations.

Chen and Kim proposed a hybrid genetic and neural network based system called GANNET (Chen 1995). This system performs concept (keyword) optimization for user-selected documents using GA and uses the optimized concepts to perform concept exploration in a Hopfield net representing related concepts. The retrieving process is cyclic and is done in two stages. The first stage is the concept optimization, the GA manipulates input documents and their associated keywords to generate an initial set of optimized concepts. The second stage is the concept exploration, the set of optimized concepts is used by the Hopfield net to produce other relevant concepts (new genes suggested by the nature) which are included in GA for the next concept optimization. This process is repeated until there is no further improvement.

Kraft et al. (1995) apply a GA programming in order to improve the weighted boolean query formulation. The documents are viewed as a vector of index terms. A weighted boolean query is represented as chromosome in Koza's genetic model (Koza 1991). The goal of the GA is to modify the query in order to improve the search performance in term of recall and precision. Their first experiments showed that the GA programming is viable method of deriving good queries.

Notations used in this article:

T	total number of stemmed terms automatically extracted from the documents
N	total number of documents
t_i	ith term
n_i	number of documents containing term t_i
d_j	jth document
tf_{ji}	frequency of t_i in d_j
d_{ji}	term weight of t_i in d_j
$Q_u^{(s)}$	query individual u at the generation (s) of the GA
$q_{ui}^{(s)}$	weight (locus) of the term$_i$ (gene) in $Q_u^{(s)}$
pop$^{(s)}$	population of individuals at the generation (s) of the GA
Pop_size	size of the population
$D_{\mathrm{r}}^{(s)}$	set of relevant documents retrieved by the pop$^{(s)}$
$D_{\mathrm{nr}}^{(s)}$	set of nonrelevant documents retrieved by the pop$^{(s)}$
D_{r}	set of relevant documents retrieved through all generations
D_{nr}	set of nonrelevant documents retrieved through all generations
Sim	Jaccard measure
Eucl_Distance	Euclidean distance

3. The GA Model for Query Space Exploration

The goal of our GA is to find an optimal set of documents which best matched the user's needs. The genetic algorithm attempts to involve, generation by generation, a population of queries towards those improving the outcome of the system. The main characteristics of our model are its knowledge-based operators, instead of the blind operators used on the previous cited works, and its niching technique.

3.1. Query Individual and Query Population

In our approach, the genetic individual is a query. Each gene or chromosome corresponds to an indexing term or concept. The *locus* (the existence or the absence of certain gene) is represented by a real value and defines the importance of the term in the considered query. Each individual representing a query is of the form:

$$Q_u(q_{u1}, q_{u2}, \ldots, q_{uT})$$

Initially, a term weight can be computed by any query term weight scheme; it will then evolve through the generations. In our case, we used the following formula (Singhal et al. 1996):

$$q_{ui} = \frac{(1 + \log(tf_{ui})) \times \log\left(\frac{N}{n_i}\right)}{\sqrt{\sum_{k=1}^{T} \left((1 + \log(tf_{uk})) \times \log\left(\frac{N}{n_k}\right)\right)^2}}$$

The initial population (set of queries), ($\text{Pop}^{(0)}$), contains the initial query and a list of relevant documents retrieved by this initial query. Thus, at the first iteration of the GA, a relevant document is considered as a query. Therefore, the initial population is not randomly constructed. Using this method, the genetic algorithm begins the exploration of the document space in "good" directions. This population is renewed after each iteration of the GA.

3.2. Fitness Function

A *fitness* is assigned to each query in the population. This fitness represents the effectiveness of a query during the retrieving stage. It is computed according to the relevance of the retrieved documents. The formula is:

$$Q\text{Fitness}\left(Q_u^{(s)}\right) = \frac{\frac{1}{\|D_r\|} * \sum_{d_j \in D_r} \text{Sim}\left(d_j, Q_u^{(s)}\right)}{\frac{1}{\|D_{nr}\|} * \sum_{d_j \in D_{nr}} \text{Sim}\left(d_j, Q_u^{(s)}\right)} \tag{1}$$

Here, $\text{Sim}(d_j, Q_u^{(s)})$ is a Jaccard function given as follows:

$$\text{Sim}\left(d_j, Q_u^{(s)}\right) = \frac{\sum_{i=1}^{T} q_{ui}^{(s)} \cdot d_{ji}}{\sum_{i=1}^{T} {q_{ui}}^2 + \sum_{i=1}^{T} d_{ji}^2 - \sum_{i=1}^{T} q_{ui}^{(s)} \cdot d_{ji}} \tag{2}$$

Thus the more a query retrieves relevant documents, the fitter the query is. This fitness function would favour the reproduction of queries that are close to relevant documents and far away from nonrelevant documents. But the problem of the document relevance function is that this function is multi-modal in the sense that relevant documents corresponding to the same information need may be located at different regions of the document space and therefore have some different vector components. Moreover, usually the classic genetic algorithm evolves moving the query population toward an optimal query that retrieves documents of similar representation. For these reasons the fitness function has been adjusted in order to encourage the construction of queries retrieving documents from different regions of the document space. Using the techniques of speciation in ecological theory (Goldberg 1994), we define a **niche** as a set of queries that retrieve documents of similar genotype. It is defined as follows:

$$\text{Niche}\left(Q_u^{(s)}\right) = \left\{ Q_v^{(s)} \text{ as } \text{Eucl\_Distance}\left(Q_u^{(s)}, Q_v^{(s)}\right) \leq \text{niching\_threshold} \right\} \tag{3}$$

$\text{Eucl\_Distance}(Q_u^{(s)}, Q_v^{(s)})$ is an Euclidean distance computed by:

$$\text{Eucl\_Distance}\left(Q_u^{(s)}, Q_v^{(s)}\right) = \text{sqrt}\left(\sum \left(q_{ui}^{(s)} - q_{vi}^{(s)}\right)^2 \right) \tag{4}$$

The niching_threshold is determined experimentally. The fitness function becomes

$$\text{Fitness}\left(Q_u^{(s)}\right) = \frac{Q\text{Fitness}\left(Q_u^{(s)}\right)}{\left\|\text{Niche}\left(Q_u^{(s)}\right)\right\|} \tag{5}$$

Thus, the query fitness is proportional to the query similarity with relevant documents and inversely proportional to both the query similarity with nonrelevant documents and the number of queries in the same niche. This would encourage the selection and reproduction in less niche query populated. Therefore, new regions will be explored according to the average fitness of the associated queries.

The query population is represented as follows:

$$\text{Pop}^{(s)} = \left\{\left(Q_u^{(s)}, \text{Fitness}\left(Q_u^{(s)}\right)\right) u = 1 \ldots \text{Pop\_size}\right\} \tag{6}$$

3.3. Genetic Operators

The genetic operators defined in this work are not pure genetic operators. They have been adapted to take advantage of techniques developed in IR. For this reason the defined operators are qualified as knowledge based operators.

3.3.1. Selection. Our selection procedure is based on roulette wheel selection (Goldberg 1994). It consists essentially of assigning to every individual of the population a number of copies in the next generation, proportional to its relative fitness. More precisely, the roulette wheel slots are sized according to the fitness of the individuals. Each individual has a certain number of slots proportional to its fitness value and the one spin of the wheel selects a single individual each time.

3.3.2. Crossover. In GA, the crossover is applied to a pair of individuals which are selected according to the crossover probability, noted P_c. These individuals exchange part of their genes delimited by randomly selected crossing points. This process leads either to one or two new individuals; the initial individuals are called the *parents* of the newly generated individuals. In our approach, the GA uses the individual queries according to their niche in order to continue the exploration in different regions of the document space. The goal is to reach relevant documents with different representations. We define two crossover operators: a crossover based on query term weight which is used for the queries belonging to the same niche; and the crossover based on term co-occurrence in the relevant documents which is used for the queries from different niches.

Crossover Based on Term Weight. This crossover does not use a crossing point, it allows us to modify the term weights according to their distribution in the relevant and nonrelevant documents. Let us consider $Q_u^{(s)}$ and $Q_v^{(s)}$ two individuals (queries) selected for crossover. The result is the new individual $Q_p^{(s)}$ defined as:

$$\left.\begin{array}{l} Q_u^{(s)}\left(q_{u1}^{(s)}, q_{u2}^{(s)}, \ldots, q_{uT}^{(s)}\right) \\ Q_v^{(s)}\left(q_{v1}^{(s)}, q_{v2}^{(s)}, \ldots, q_{vT}^{(s)}\right) \end{array}\right\} \rightarrow Q_p^{(s+1)}\left(q_{p1}^{(s+1)}, q_{p2}^{(s+1)}, \ldots, q_{pT}^{(s+1)}\right) \tag{7}$$

$$q_{pi}^{(s+1)} = \begin{cases} \text{Max}\left(q_{ui}^{(s)}, q_{vi}^{(s)}\right) & \text{if weight}\left(t_i, D_r^{(s)}\right) \geq \text{weight}\left(t_i D_{nr}^{(s)}\right) \\ \text{Min}\left(q_{ui}^{(s)}, q_{vi}^{(s)}\right) & \text{otherwise} \end{cases} \tag{8}$$

We defined the weight of term t_i in a set of documents D as follows:

$$\text{weight}(t_i, D) = \sum_{d_j \in D} d_{ji} \tag{9}$$

In other words, if the weight of term t_i in the set of relevant documents is higher than its weight in the set of nonrelevant documents, this term is retained as significant and the highest weight among $(q_{ui}^{(s)}, q_{vi}^{(s)})$ is assigned to this term in the new query $Q_p^{(s+1)}$. Otherwise, the lowest weight is assigned to it in the new query. This operator will be applied on queries of the same niche. Indeed, the goal of this operator is to adjust the descriptions of the new queries in order to recall further relevant documents in the same region of the document space.

Crossover Based on Term Co-occurrence. Let us consider $Q_u^{(s)}$ and $Q_v^{(s)}$ two individuals selected for crossover with k as the crossing point and $Q_u^{(s+1)}$ and $Q_v^{(s+1)}$ as the new individuals resulting from the crossover. In the classical GA, the crossover operator, called a blind operator in our experiment, consists of exchanging a part of the genes between the selected individuals. Our second crossover operator uses the same idea, but based on the term co-occurrence in the relevant documents. Thus, the new individuals copy the first k terms from their parents and the remaining terms are built as follows: each term among the non-copied terms of one parent, is replaced by the most co-occurent term in the other parent. Formally, this operator is defined as:

$$\left.\begin{array}{l} Q_u^{(s)}\left(q_{u1}^{(s)}, q_{u2}^{(s)}, \ldots, q_{uT}^{(s)}\right) \\ Q_v^{(s)}\left(q_{v1}^{(s)}, q_{v2}^{(s)}, \ldots, q_{vT}^{(s)}\right) \end{array}\right\} \rightarrow \begin{array}{l} Q_u^{(s+1)}\left(q_{u1}^{(s+1)}, q_{u2}^{(s+1)}, \ldots, q_{uT}^{(s+1)}\right) \\ Q_v^{(s+1)}\left(q_{v1}^{(s+1)}, q_{v2}^{(s+1)}, \ldots, q_{vT}^{(s+1)}\right) \end{array} \tag{10}$$

where

$$\begin{array}{l} q_{ui}^{(s+1)} = q_{ui}^{(s)} \text{ if } (i \leq k) \\ \text{otherwise for } i = k+1 \text{ to } T \\ \quad \text{lookup } t_l \text{ in } Q_v^{(s)} / l > k \ \wedge \ \max^D\left(C_{t_i,t_l}, Q_v^{(s)}\right) \\ \quad q_{ul}^{(s+1)} = q_{vl}^{(s)} \\ \quad \text{endfor} \end{array} \tag{11}$$

$Q_v^{(s+1)}$ is built in same way.

Here, $\max^D(C_{t_i,t_l}, Q_v^{(s)})$ returns term t_l of $Q_v^{(s)}$ that has the highest co-occurrence weight with t_i in the set of documents D.

The co-occurrence of two terms in a set of documents is defined as follows:

$$C_{t_i,t_l} = \frac{\sum_{d_j \in D_r} d_{ji} * d_{jl}}{\sum_{d_j \in D_r} d_{ji}^2 + \sum_{d_j \in D_r} d_{jl}^2 - \sum_{d_j \in D_r} d_{ji} * d_{jl}} \tag{12}$$

This operator will be applied to the queries belonging to different niches. It allows for the exchange of a section of information between two queries retrieving relevant documents

located in different regions of the document space. This would enable exploration of a new region in the document space. Notice that a variant of this operator has been defined in our works. One consists of copying all the genes different to zero from the parent to the new individual and then for each gene, from $(k+1)$ to (s), lookup its most co-occurent term in the other parent and add it to the new individual (Tamine 1998).

Blind Crossover. The blind crossover is based on the classical GA crossover operator. Let us consider $Q_u^{(s)}$ and $Q_v^{(s)}$ two individuals selected for crossover with k as the crossing point and $Q_u^{(s+1)}$ and $Q_v^{(s+1)}$ as the new individuals resulting from the blind crossover. This operator is defined as follows:

$$\left.\begin{array}{l} Q_u^{(s)}\left(q_{u1}^{(s)}, q_{u2}^{(s)}, q_{uk}^{(s)}, q_{uk+1}^{(s)}, \ldots, q_{uT}^{(s)}\right) \\ Q_v^{(s)}\left(q_{v1}^{(s)}, q_{v2}^{(s)}, q_{vk}^{(s)}, q_{vk+1}^{(s)}, \ldots, q_{vT}^{(s)}\right) \end{array}\right\} \rightarrow \begin{array}{l} Q_u^{(s+1)}\left(q_{u1}^{(s)}, q_{u2}^{(s)}, \ldots, q_{uk}^{(s)}, q_{vk+1}^{(s)}, \ldots, q_{vT}^{(s)}\right) \\ Q_v^{(s+1)}\left(q_{v1}^{(s)}, q_{v2}^{(s)}, \ldots, q_{vk}^{(s)}, q_{uk+1}^{(s)}, \ldots, q_{uT}^{(s)}\right) \end{array} \tag{13}$$

3.3.3. Mutation.

Mutation Based on Term Relevance. This consists essentially of exploring the terms occurring in the relevant documents in order to adjust the corresponding gene values in the query selected for the mutation. The mutation operator we defined is based on *term relevance*, this mutation is noted *Mutation1*.

Let us consider $Q_u^{(s)}$ as the selected individual query and $L^{(s)}$ as the set of terms from $D_{\mathrm{r}}^{(s)}$, the relevant documents retrieved in the last generation of the GA. The mutation will alter the genes of the selected individual on the basis of the $L^{(s)}$ terms and on the probability P_{m}. The $L^{(s)}$ terms are sorted according to a score value calculated as follows:

$$\mathrm{Score}(t_i) = \frac{\sum_{d_j \in D_{\mathrm{r}}^{(s)}} d_{ji}}{\left\| D_{\mathrm{r}}^{(s)} \right\|} \tag{14}$$

This score is used in the case of limiting the number of terms that can be used for the mutation process. Previous experiments have been performed using this limit (Tamine 1998). The mutation operation is done as follows:

1. for each term t_i in $L^{(s)}$
2. if (random(p) $< P_{\mathrm{m}}$) then /* mutate gene t_i*/
3. $q_{ui}^{(s)} = \mathrm{average}(Q_i^{(s)}) - \delta$
4. endif
5. endfor

random(p) generates a random number p in the range [0..1]. The average function is computed as follows: $\mathrm{average}(Q_u^{(s)}) = \frac{\sum_j^T q_{ui}^{(s)}}{n_{q_{ui}^{(s)}}}$, where $n_{q_{ui}^{(s)}}$ is number of $q_{ui}^{(s)} \neq 0$ in $Q_u^{(s)}$. δ is a parameter used to control the average value, the value used is $\delta = 0$.

Blind Mutation In blind mutation the genes are mutated by modifying their weight arbitrarily. Let us consider $Q_u^{(s)}$ as the selected individual for mutation. This operation is performed as follows:

1. for each t_i in $Q_u^{(s)}$
2. if (random(p) $<$ P_{m}) then /*mutate*/
3. $q_{ui}^{(s)} =$ arbitrary(w)
4. endif
5. endfor

arbitrary(w) returns a real w in range of [0..1].

4. Experiments and Results

4.1. Experiments

The experiments were carried out on TREC-6 French data and 21 queries. The documents are from the SDA News (Schweizerischen Depeschenagentur-Swiss News Agency—141 646 documents). They were run using the Mercure information retrieval system (Boughanem and Soule-Dupuy 1997a), as used in different TREC conferences (Harman 1997). The main goal of these experiments was to evaluate the effectiveness of our GA for IR. More precisely, at first we measure the effects of P_{c} and P_{m} probabilities and the population size, and then we compare the knowledge based operators versus the blind operators. We also evaluate the effect of our niching technique.

Before explaining the experimentation, we firstly notice that the query evaluation, corresponding to doing the search in the algorithm, is done using the spreading activation technique developed in Mercure system (Boughanem and Soule-Dupuy 1997b). This process enables the ranking of the retrieved documents. Secondly, there are fifteen (15) judged document in this experiment. This number has been chosen arbitrarily knowing that a value in the range of [10..15] is commonly used in most works performing manual relevance feedback (Robertson and Walker 1997, Harman 1992, Salton and Buckley 1990, Boughanem and Soule-Dupuy 1997b). However experiments varying this number will be done in future work. Thus, the general experimentation is done as follows:

1. Submit the initial query and do the search
2. Judge the top fifteen documents
3. build the initial population as follows
   - 3.1. some top judged relevant documents join the initial query in the set of initial individuals
   - 3.2. compute the fitness of this individuals
   - 3.3. apply the genetic operators to these individuals as follows:
     Repeat
     - parent1 = wheel-selection(Pop$^{(s)}$); parent2 = wheel-selection(Pop$^{(s)}$)
     - Crossover(P_{c}, parent1, parent2, son1, son2)

Table 1. Genetic operators.

G1	Crossover1: based on term weight	Mutation1: based on term relevance
G2	Crossover2: based on term co-occurrence	Mutation1: based on term relevance
Blind	Blind crossover	Blind mutation

- Mutation(P_m; son1, sonmut); Add population (sonmut, Pop$^{(s+1)}$)
- Mutation(P_m, son2, sonmut); Add population (sonmut, Pop$^{(s+1)}$)

until Size(Pop$^{(s+1)}$) = fixed population size. the population is then built

4. For each query in the population do a search
5. Build a merged list of documents
6. Judge the top 15 documents
7. Compute the fitness of each query
8. Apply the genetic operators; as in 3.3
9. Repeat 4, 5, 6, 7, 8 steps until a fixed number of iteration

The number of iterations has been fixed at 5. At each iteration of the GA (corresponding to a new population and called a new generation of the GA), the system presents to the user a list of new documents, the user sees the top 15 and selects the relevant ones, the GA is then applied. Fifteen new more documents are retrieved and shown to the user and so on. The goal of our GA is to recall new relevant documents at each iteration of the search. Thus, in order to evaluate the different points outlined above, and with respect to the genetic approach we have defined three basic groups of operators. Each group is composed by a specific crossover operator and a mutation operator in Table 1.

Notice that the first experiments have been performed using the G1 and G2 operators in order to measure the parameters of the GA (P_c, P_m, Pop_size). But the main operator we have defined is based on the niching technique and will be evaluated in Section 4.3.3.

4.2. Evaluation Method

The main evaluations presented in this paper measure the ability of our GA to increase the precision in the top fifteen documents retrieved at each iteration of the GA. So, because of the multiple iteration aspect of the search and the use of relevance judgement, the results reported in the paper are based on a residual ranking evaluation (Chang et al. 1971). This method is used to evaluate the effectiveness of relevance feedback methods. In this method all the documents previously judged are removed from the document rankings produced by both the initial query, which corresponds to the iteration 0 in our algorithm, and the feedback query, which corresponds to iteration 1 in our algorithm. Precision and recall are computed for these and then for both residual lists of documents. In the case of multiple iteration the comparison is done in the same way between the residual document retrieved at iteration (i) to the residual document retrieved at iteration ($i+1$). This tells us how much we gained by doing the next iteration of the GA. For further details, readers may refer to Harman (1992).

In our context we compare the results obtained with GA to those obtained with no GA. Let us consider that the iteration $(i + 1)$ of the GA is performed. In order to measure the effectiveness of the GA in this iteration, we compared the 15 top retrieved documents at that iteration to the no GA documents, which corresponds to the list resulting from just keeping the next 15 documents at iteration (i). We limited the number of documents to 15 because our main goal is to improve the precision at that number.

4.3. Results and Discussions

4.3.1. Effects of the GA parameters: P_c, P_m, Pop_size. The effectiveness of the defined GA depends primarily on a list of important genetic parameters: the crossover probability P_c, the mutation probability P_m and the population size.

Effect of the P_c and P_m Probabilities. The first experiment has been performed by varying P_c and P_m, $P_c \in \{.1, .25, .5, .7\}$ and $P_m \in \{.01, .07, .1, .25, .5\}$ for a fixed Pop_size = 5 (the extreme values 0 and 1 are not used in the GA, we respected this convention). The GA is performed by considering the G1 group only. Table 2 shows, for different P_c and P_m values, the number of relevant documents retrieved in a given iteration (totaled across all queries used at that iteration) and the cumulative total of relevant documents retrieved by that point. The columns numbered from 1 to 5 represent the iteration number, the iteration 0 corresponding to the initial search is not represented, because the results of the use of the GA begin at iteration 1.

We see from these results that, as expected, P_c and P_m play an important role in the effectiveness of the GA. The table shows that the P_c plays more role than P_m, which is in agreement with the GA literature (Davis 1991). Indeed, it can be seen that the cumulative number of relevant retrieved documents at given iteration and P_c is nearly the same for varying values of P_m. The cumulative number of relevant retrieved documents at each iteration increases, and it goes from 259 for $P_c = .1$ to 292 for $P_c = .7$.

According to these results, some questions can be asked. The main question concerns the real contribution of the mutation operator. It is well known in the GA literature (Davis 1991, Goldberg 1994) that the mutation operator has a very small effect in the GA and it is often used with very small probability, around 0.005, but, *is it worth using this operator in our algorithm*? We think that even though the effect of such an operator is very small, it is still worth using it. First, we want to be in agreement with the general GA approach. Second, even though the crossover operator efficiently recombines the existing genes (terms) it possible that the crossover operator loses some important terms. So, mutation will, in most case, protect us from this problem because the mutation operator allows us to add some important terms extracted from the relevant judged documents. For the remain experiments we have chosen $P_c = .7$ and $P_m = 0.07$.

Effect of the Population Size. The size of the population is a determinant factor, indeed a large population size induces noise and a high query run-time, whereas a small size induces silence because the genetic transformations are done in the same region of the document space.

Table 2. Effect of the P_c and P_m parameters.

P_m	1	2	3	4	5
For $P_c = .1$					
.01	91 (91)	36 (127)	43 (170)	53 (223)	36 (259)
.07	91 (91)	36 (127)	43 (170)	53 (223)	37 (260)
.1	91 (91)	36 (127)	43 (170)	53 (223)	37 (260)
.25	91 (91)	36 (127)	44 (171)	53 (224)	35 (259)
.5	91 (91)	37 (128)	47 (175)	49 (224)	36 (260)
For $P_c = .25$					
.01	89 (89)	60 (149)	43 (192)	36 (228)	41 (269)
.07	89 (89)	60 (149)	43 (192)	36 (228)	40 (268)
.1	89 (89)	60 (149)	43 (192)	36 (228)	40 (268)
.25	89 (89)	60 (149)	43 (192)	36 (228)	40 (268)
.5	89 (89)	60 (149)	43 (192)	38 (230)	38 (268)
For $P_c = .5$					
.01	92 (92)	49 (141)	49 (190)	48 (238)	37 (275)
.07	92 (92)	49 (141)	49 (190)	48 (238)	36 (274)
.1	92 (92)	49 (141)	49 (190)	48 (238)	36 (274)
.25	92 (92)	49 (141)	49 (190)	48 (238)	37 (275)
.5	92 (92)	49 (141)	47 (188)	46 (234)	34 (266)
For $P_c = .7$					
.01	94 (94)	54 (148)	69 (217)	39 (256)	36 (292)
.07	94 (94)	54 (148)	69 (217)	39 (256)	36 (292)
.1	94 (94)	54 (148)	69 (217)	39 (256)	36 (292)
.25	94 (94)	54 (148)	69 (217)	39 (256)	34 (290)
.5	94 (94)	54 (148)	68 (216)	40 (256)	34 (290)

Values represent number of relevant documents in the 15 top retrievals at given iteration (over 21 queries) and the values in parentheses represent cumulative number of relevant documents in the 15 top retrievals at given iteration (over 21 queries).

Table 3 compares the results using no GA and using our GA for various population sizes and by considering the $G1$ and $G2$ operators. The table lists the number of relevant document retrieved in a given (totaled across all queries used at that iteration) and the cumulative total number of relevant documents retrieved by that point.

We notice that with GA on both $G1$ and $G2$, the total number of relevant documents after 5 iterations is much higher than using no GA for all the population sizes greater than 2. It can be also seen that for a population size greater than 2 when using the group $G1$, the number of relevant document is much higher since the first iteration and still higher through successive iterations than when using no GA. Whereas for $G2$, this number begins to increase after the third iteration. Furthermore, the table shows that $G1$ is more effective and recalls more relevant document than $G2$.

Table 3. Effect of the population size through 5 generations and by considering $G1$ and $G2$ operators.

	$G1$					$G2$				
	iter1	iter2	iter3	iter4	iter5	iter1	iter2	iter3	iter4	iter5
Population size = 2										
No GA	90 (90)	48 (138)	33 (171)	46 (217)	24 (241)	90 (90)	59 (149)	36 (185)	40 (195)	21 (216)
With GA	72 (72)	52 (124)	36 (160)	46 (206)	24 (230)	72 (72)	42 (112)	42 (154)	35 (189)	21 (210)
Population size = 4										
No GA	90 (90)	57 (147)	50 (197)	42 (239)	42 (281)	90 (90)	54 (144)	46 (190)	40 (230)	41 (273)
With GA	96 (96)	64 (160)	40 (200)	45 (245)	42 (287)	85 (85)	40 (125)	46 (171)	58 (229)	45 (274)
Population size = 6										
No GA	90 (90)	57 (147)	39 (186)	44 (230)	37 (267)	90 (90)	64 (154)	40 (194)	31 (225)	31 (256)
With GA	96 (96)	64 (160)	55 (215)	51 (266)	30 (296)	85 (85)	64 (149)	41 (190)	51 (241)	37 (278)
Population size = 8										
No GA	90 (90)	64 (154)	42 (196)	37 (233)	42 (275)	90 (90)	64 (154)	36 (190)	40 (230)	30 (260)
With GA	96 (96)	72 (168)	50 (218)	38 (256)	29 (285)	85 (85)	64 (159)	45 (204)	36 (240)	30 (270)

Values represent number of relevant documents retrieved at iteration and the values in parentheses represent cumulative total number of relevant documents retrieved.

The main conclusion that can be drawn from this experiment is that the basic operators we defined improve the result of the search for a population size more than 2. In order to make a good compromise between run-query time and noise, the population of size 6 seems more significant in both $G1$ and $G2$. We retained this value for our algorithm for the remaining experiments.

4.3.2. Effect of the Knowledge Based Operators Versus Blind Operators. Table 4 compares the results of the GA using the blind operators and the knowledge based operators (population size = 6). The table gives the number of relevant document in the top 15 retrieved at each iteration of the GA, and the cumulative total number at that point.

We clearly notice that the knowledge-based operators are more effective than the blind ones. Indeed the cumulative total number of relevant documents after 5 iterations is almost twice as high using $G1$ and $G2$.

Table 4. Results for knowledge based operator vs. blind operators.

Operator	iter1	iter2	iter3	iter4	iter5
Knowledge-based					
$G1$	96 (96)	64 (160)	55 (215)	51 (266)	30 (296)
$G2$	85 (85)	64 (149)	41 (190)	51 (241)	37 (278)
Blind					
No GA	90 (90)	23 (113)	23 (136)	32 (168)	22 (190)
With GA	27 (27)	28 (55)	41 (96)	28 (126)	24 (150)

It can be also seen that the blind operators have a negative effect in the first iteration of the search. However, we notice that the initial search (the search with the initial query without any iteration for the GA) gives a good result although the table shows that after the first iteration the number of relevant retrieved documents increases through the generation of the GA. This is therefore in agreement with the GA literature. Here it is mentioned that the convergence of the algorithm is observed after many iterations of the GA, at least more than 20. In our context, however, a large number of iterations is not conceivable except, if this algorithm is used to build a profile in a filtering process, for example Harman (1997).

4.3.3. Effect of the Niching Technique: Niching or not Niching? We recall that the main idea of our GA is the use of the niching technique. Thus, the crossover operators have been defined in order to take into account this notion. In order to evaluate the niching technique we build a combination of genetic operators by considering specific crossover according to niching. Table 5 shows this combination.

This operator is applied as follows: when two queries are selected for transformations, if they belong to the same niche, crossover1 is applied to these queries and then mutation1 to the new query (result of the crossover1). If they are in different niches, crossover2 is applied to these queries and then mutation1 to each new query resulting from the crossover2.

To be effective the niching technique needs a suitable niching_threshold, to decide whether two queries belong to the same niche or not. But, as it can be seen in Section 3.2, the Euclidean distance used in this experiment is not normalized, it is therefore difficult to fix the superior limit of the threshold knowing that this threshold varies from 0 to P, P is a positive real. Thus in this experiment, to not consider all the possible values of P, we just considered as superior limit, the highest distance computed in this experiment through all the generations of the GA and all the queries. Thus, this experiment is performed by varying the niching_threshold $\in \{0, .1, .25, .5, 1, 2, 1.5, 5, 8\}$.

Table 6 shows the number of new relevant documents retrieved at each iteration of the GA and the total number of relevant documents retrieved through all iteration of the GA. We notice that the threshold plays an important role in the niching operator. It can be seen that a good niching_threshold is around .1–.2. However the non-normalized Euclidean distance used to determine whether two queries belong to the same niche or not is needs more investigations, because the niching threshold is difficult to tune.

Concerning the effectiveness of the niching technique, Table 7 shows the comparison between the $G1$ and $G2$ (operators without niching) and $G3$ (with the niching technique using niching_threshold $= .1$).

It can be easily seen that $G3$ is more efficient than its components taken separately (i.e., $G1$ and $G2$). We notice that beyond the first iteration, the number of relevant documents in $G3$ is much higher than $G1$ and $G2$.

Table 8 compares the results using GA ($G3$) and using no GA. We notice that when using $G3$, the number of relevant documents retrieved at each iteration is much higher than

Table 5. Niching operator.

Niching operator	Same niche	Different niche
$G3$	Crossover1, mutation1	Crossover2, mutation1

Table 6. Results of the niching according to thresholds.

Niching threshold	nb. relev. doc. in the 15 top retrieved					
	iter1	iter2	iter3	iter4	iter5	Total
0	85	64	41	51	37	278
.1	105	56	51	51	41	304
.25	94	60	51	50	37	300
.5	94	60	48	50	42	294
1	88	64	54	42	48	296
2	87	69	63	34	29	282
5	86	75	60	30	31	282
8	85	84	42	29	34	274
10	85	64	61	47	25	283

Table 7. Effect of the niching comparing to the basic operator.

	iter1	iter2	iter3	iter4	iter5
$G1$	96 (96)	64 (160)	55 (215)	51 (266)	32 (298)
$G2$	85 (85)	64 (149)	41 (190)	51 (241)	37 (278)
$G3$	105 (105)	56 (161)	51 (212)	51 (263)	41 (304)

Values represent number of relevant documents retrieved at iteration and the values in parentheses represent cumulative total number of relevant documents retrieved.

Table 8. Effectiveness of the niching.

	iter1	iter2	iter3	iter4	iter5
No GA	90 (90)	48 (138)	46 (184)	44 (228)	39 (267)
With GA	105 (105)	56 (161)	51 (212)	51 (263)	41 (304)

Values represent number of relevant documents retrieved at iteration and the values in parentheses represent cumulative total number of relevant documents retrieved.

using no GA. Therefore the cumulative total number of relevant documents using $G3$ is still higher through all the iterations than not using $G3$.

To summarize, it is clear from these tables that all the genetic operators we defined improve the performance of the search whether the niching is used or not. It seems that even though the differences between the results are not very great, the niching technique is effective in our GA. However, these results might be improved by using more suitable combinations between the operators, probably by revising the niching formula.

5. Conclusion

The experiments reported here constitute our first investigation in GA and IR. The results presented here demonstrate the effectiveness of our GA approach to improve the performance of an information retrieval system. We mainly showed:

- the manipulation of a population of queries,
- the use of the probabilistic exploration of different zones of the document space,
- the use of acknowledge based operator instead of blind operators used in the classical GA,
- the integration of the niching technique to recall relevant documents with different descriptions.

However, the results produced by the GA depend on the values of many parameters, such as the different probabilities, the population size and the different thresholds. This first experiment in TREC French data, allowed us to tune these parameters, and particularly to show another way of adapting the genetic approach to IR. We have shown that our GA for IR improves the results of the search in the interactive environment, even though the random aspect of the GA is still a big question. We also have to say that we are not comparing our method to other relevance feedback method. Our goal is to show how we can do differently and how the GA could be used for IR. However, this first investigation allowed us to be optimistic for future work in this direction, particularly when the documents are heterogeneous. In this case an IRS needs to use several queries, each one with specific representation to be efficient. We also notice that the number of generations and individuals of these experiments is very small comparing to the GA theory, further experiments will be undertaken in this direction. In addition of this point our next goal is to use our approach on the English TREC data (Harman 1997), in some TREC tasks (ad hoc and the interactive track) with more data and more queries. Then a global comparison for example between the size of the data and the GA parameters could be performed.

References

Ankenbrandt C (1990) An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. FOGA90, pp. 53–58.

Boughanem M and Soule-Dupuy C (1997a) Mercure at trec6. In: Harman DK, ed. 6th International Conference on Text REtrieval TREC6. November 21–23. NIST SP, pp. 321–328.

Boughanem M and Soule-Dupuy C (1997b) Query modification based on relevance backpropagation. In: Proceedings of the 5th International Conference on Computer-Assisted Information Searching on Internet (RIAO'97), Montreal, pp. 469–487.

Chang YK, Cirillo GC and Razon J (1971) Evaluation of feedback retrieval using modified freezing, residual collection and test and control groups. In: The Smart Retrieval System: Experiments in Automatic Document Processing, Prentice-Hall Inc., chap. 17, pp. 355–370.

Chen H (1995) Machine learning for information retrieval: Neural networks, symbolic learning and genetic algorithms. JASIS, 46(3):194–216.

Davis L (1991) Handbook of Genetic Algorithms. Van Nostram Reinhold, New York.

Haines D and Croft WB (1993) Relevance feedback and inference networks. In: ACM SIGIR International Conference on Research and Development in Information Retrieval, pp. 2–11.

Goldberg DE (1994) Algorithmes génétiques. Exploration, optimisation et apprentissage automatique. Addison-Wesley, France.

Gordon M (1988) Probabilistic and genetic algorithms for document retrieval. Communications of the ACM, pp. 1208–1218.

Grefenstette JJ (1995) Virtual genetic algorithms: First results, Technical report AIC-95-013, Navy Center for Applied Research in Artificial Intelligence.

Harman D (1997) TREC overview. In: 6th International Conference on Text REtrieval TREC6, November 21–23. Harman DK, ed. NIST SP, pp. 1–24.

Harman D (1992) Relevance feedback revisited. In: 15th Annual ACM SIGIR Conference On Research and Development in Information Retrieval Copenhague, Denmark, pp. 1–10.

Holland J (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Holland J (1992) Les algorithmes gënëtiques. Revue POUR LA SCIENCE 179, pp. 44–51.

Koza JR (1991) A hierarchical approach to learning the Boolean multiplexer function. In: Rawlins G, ed., Foundations of Genetic Algorithms. Morgan Kaufman, San Mateo, CA, pp. 171–192.

Kraft DH, Petry FE, Buckles BP and Sadisavan T (1995) Applying Genetic Algorithms to Information Retrieval Systems Via Relevance Feedback. In: Bosc and Kacprzyk J, eds. Fuzziness in Database Managment Systems. Studies in Fuzziness Series, Physica-Verlag, Heidelberg, Germany, pp. 330–344.

Kwok KL (1989) A neural network for probabilistic information retrieval. In: Proceedings of the 12th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval, Cambridge, MA, pp. 21–30.

Radcliffe NJ (1991) Equivalence class analysis of genetic algorithms. Complex Systems, 5:183–220.

Robertson S and Sparck Jones K (1976) Relevance weighting of search terms. Journal of the American Society for Information Science, 27:129–146.

Robertson S and Walker S (1997) On relevance weights with little relevance information. ACM/SIGIR International Conference on Research and development in Information Retrieval, pp. 16–24.

Rocchio JJ (1971) Relevance feedback in information retrieval. In: Salton G, ed. The Smart System Experiments in Automatic Document Processing, Prentice-Hall, Inc., Englewood Cliffs, NJ, pp. 313–323.

Salton G (1970) The SMART Retrieval System. Prentice-Hall, Inc. Englewood Cliffs, NJ.

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

Sebag M and Schoenauer M (1996) Contrôle d'un algorithme genetique. Revue d'intelligence artificielle, 2/3:389–428.

Singhal A, Buckley C and Mitra M (1996) Pivoted document length normalisation. In: Conference on Research and Development in Information Retrieval (SIGIR), pp. 21–29.

Tamine L (1998) Reformulation de requêtes dans les SRI: une approche basée sur la génétique, Master Thesis, University of Tizi-Ouzou.

Wilkinson R and Hingston P (1991) Using the cosine measure in a neural network for document retrieval. In: ACM/SIGIR International Conference on Research and Development in Information Retrieval.

Wong SKM, Cai YJ and Yao YY (1993) Computation of term associations by a neural network. In: Conference on Research and Development in Information Retrieval (SIGIR), pp. 107–115.

Yang JJ and Korfhage R (1993) Query optimization in information retrieval using genetic algorithms. ICGA, 93.