Large Population or Many Generations for Genetic Algorithms? Implications in Information Retrieval

Dana Vrajitoru[1][2]

[1] University of Neuchâtel, Computer Science Department, Pierre-à-Mazel 7, 2000 Neuchâtel, Switzerland
[2] EPFL, Department of Mathematics, CH-1015 Lausanne, Switzerland, email: dana.vrajitoru@epfl.ch

Abstract. Artificial intelligence models may be used to improve performance of information retrieval (IR) systems and the genetic algorithms (GAs) are an example of such a model. This paper presents an application of GAs as a relevance feedback method aiming to improve the document representation and indexing. In this particular form of GAs, various document descriptions compete with each other and a better collection indexing is sought through reproduction, crossover and mutation operations. In this paradigm, we are searching for the optimal balance between two genetic parameters: the population size and the number of generations. We try to discover the optimal parameter choice both by experiments using the CACM and CISI collections, and by a theoretical analysis providing explanation of the experimental results. The general conclusion tends to be that larger populations have better chance of significantly improving the effectiveness of retrieval.

1 Introduction

Probabilistic algorithms are relatively recent in computer science but their range of applications has increased rapidly. They present the advantage of being able to take different decisions at different moments while solving the same problem (Brassard and Bratley 1994 [2]). If they do not find the solution to a problem the first time, they can still find it in another trial. The GAs are a special case of such algorithms. Since their development (Holland 1975 [10]), they have been applied to various problems, and information retrieval is an example.

Inspired by the natural selection of living organisms, the GAs are adaptable to a large number of problems because they offer a very general paradigm, where the domain-specific knowledge can easily be plugged in. Their robustness, simplicity, and variety of solutions they can find make them attractive in various fields and especially for problems difficult to solve by more traditional approaches (De Jong and Spears 1989 [4], Sushil and Gong 1997 [21]).

The GAs work within a space of possible solutions to a given problem. Starting with a number of such potential solutions, they will seek better ones by operations of reproduction, crossover and mutation. For the GAs to be efficient, the user needs to provide a good representation of their own problem and a fitness function describing how 'close' a solution guess is to the goal of the search, departing 'good' from 'bad' solutions. These two aspects represent the main difficulty of the GAs.

Information retrieval researchers have suggested these algorithms to improve the performance of their systems. Gordon (1988 [8]), and Blair (1990 [1]) have used them to improve document indexing. Chen (1995 [3]), Petry *et al.* (1993 [14]), Yang *et al.* (1992 [24], 1992 [24]), Kraft *et al.* (1992 [11]) and Sanchez *et al.* (1992 [19]) present an approach based on GAs to enhance the query description. Finally, Gordon (1991 [9]) has employed them to build document clusters.

In our previous research (Vrajitoru 1997 [22]), we have used the GAs in information retrieval to improve the document representation within the vector space model (Salton 1971 [15]). Our results have shown that this paradigm can be an interesting approach for this research field. As a side effect, we have also found that the parameter settings are very important when the problem size is rather large.

In the present research we start from the idea that information retrieval is a problem of large size. Thus, the available memory imposes limitations concerning the size of the genetic population and the number of generations. Following this idea, the present paper analyses the importance of the population size for the GAs in general, and its implications for real information retrieval tasks. The question we would like to address is: for the same computational effort, what advantage can we take from constructing larger populations, or, on the contrary, can a greater number of generations lead to better performance ?

To answer this question, this paper presents an experimental and a theoretical approach. Thus, Section 2 introduces the GAs, the problem we want to use them for, and its genetic representation. Section 3 presents our experiments and their results, as well as a theoretical discussion about the implications of the parameter choice we have studied.

2 Genetic algorithm and information retrieval

This section presents the GAs and the genetic representation of our information retrieval problem. In the first subsection, we present the main guidelines and terminology of the GAs. In a second, we expose how the GAs may be used within the context of information retrieval. Finally, the last subsection introduces the main parameter setting we have used in our research.

2.1 The functionality of GAs

The GAs are evolutionary algorithms initially elaborated for optimization problems, but that can be used in many other contexts.

Problem. Let E be a set of potential solutions to a problem. A fitness function taking real values $f : E \rightarrow R$ tells us, for each element of E, how good a solution it is for the given problem. We search for one of the elements in E that maximize the fitness function:

$$e_0 \text{ such that } f(e_0) = \max(f(e),\ e \in E) \tag{1}$$

In the context of GAs, each solution is represented as a vector of length L, where each position is called a locus and the vector's value at that position is called a gene. The elements of this form are called individuals or chromosomes. The genes are usually binary, a representation that is easy to handle and offers many possible combinations.

$$ind =< i_1,\ i_2,\ \ldots,\ i_L > \tag{2}$$

A GA will start with a number of individuals chosen by various criteria from E and will seek a better solution by making these individuals and their descendants compete with each other through a number of iterations or generations. The simplest GA constructs a new generation from an old one following three steps:

- reproduction
- crossover
- mutation

If P_0 is the initial population, the reproduction operation chooses a number of individuals from P_0 equal to its cardinal, using a random selection *with replacement.* The selection is biased according to the fitness of the underlying individuals. Thus, the 'good' individuals have a better chance to be selected, and can appear several times after reproduction, while the 'weak' individuals tend to disappear. This form of reproduction is called the 'roulette wheel' or 'fitness-proportionate' (Goldberg 1989 [7]).

The crossover operation builds two new individuals or children from two parents. We have used the 1-point crossover (Goldberg 1989 [7]) which cuts both parents at a random position $1 \leq site \leq L$, and then swaps the second part of the parents resulting from the cut.

$$\begin{array}{c} 0011 \mid 0011101 \\ 1001 \mid 1011001 \\ \Downarrow \\ 0011 \mid 1011001 \\ 1001 \mid 0011101 \end{array}$$

Finally, the mutation simply replaces a random gene in an individual to its opposite:

$$01\ \mathbf{0}\ 01011 \rightarrow 01\ \mathbf{1}\ 01011$$

This last operation is introduced to guarantee that every value {0, 1} may always appear in every position or locus and to simulate spontaneous information income.

These three operations are repeated a number of times called the generation number. This parameter can be chosen by the user in advance, or can be determined by a stop condition like the detection of convergence or the achievement of an upper bound for the fitness function. In our research, the generation number is always fixed in advance and its 'optimal' value is one of the goals of this paper.

2.2 From information retrieval to genetic algorithms

General Problem. Given a document collection $D = \{d_i, i = 1 \ldots m\}$ and a query q, find the set of documents $\{d_r, r = 1 \ldots R\}$ that are relevant to the query.

Our starting point is the vector space model (Salton 1971 [15]). According to it, after removing the common words and the suffixes, each term t_j occurring in a document d_i is attributed a weight t_{ij} reflecting its importance in the document representation. More precisely, we have computed this weight as:

$$t_{ij} = ntf_{ij} \cdot nidf_j \text{ where } ntf_{ij} = \frac{tf_{ij}}{\max_k tf_{ik}} \text{ and } nidf_j = \frac{\log(m) - log(df_j)}{\log(m)} \tag{3}$$

In Equation (3), we denoted by ntf_{ij} the normalized frequency of the term t_j in the document d_i. It is computed as the actual frequency of the term t_j in the document d_i divided by the maximal frequency over all the terms t_k occurring in the document d_i. The component $nidf_j$ denotes the normalized inverted frequency of the term t_j in the collection. In the formula defining it, m in the size of the collection, and df_j is the number of documents in which the term t_j occurs.

Intuitively, Equation (3) means that terms that are frequent in a document will get higher weights (component ntf). On the other hand, we must reduce the weights of a term that is frequent in the whole collection (component $nidf$).

The query is processed in the same way as the documents according to Equation (3). For each document in the collection, its similarity with the query is computed with the cosine measure (Salton 1971 [15]):

$$sim(d_k, q) = \frac{\sum w_{qi} t_{ki}}{\sqrt{\sum w_{qi}^2 \sum t_{ki}^2}} \tag{4}$$

within which, w_{qi} represents the weight of the term t_i in the query q.

To evaluate the response of the system, the user must specify which of these documents are really relevant to its needs. In practice, there exist several test collections provided with a set of queries whose relevance judgments are known. We have used the CACM collection (Communications of the Association for Computing Machinery) and the CISI collection (Collection of the Institute for Scientific Information). Table 1 presents a short description of these collections.

Table 1. Statistics of our test collections

	CACM	CISI
Number of documents	3204	1460
Number of queries	50	35
Number of unique indexing terms	5935	5823
Average number of terms by query	11.24	7.43
Average number of relevant documents by query	15.84	49.77
Average number of indexed terms by document	58.57	119.80

Knowing the relevance judgments, there exist two well-known measures to evaluate the system's answer to a query: the precision and the recall.

$$\begin{aligned} precision &= \frac{number\ of\ retrieved \cap relevant\ documents}{number\ of\ retrieved\ documents} \\ recall &= \frac{number\ of\ retrieved \cap relevant\ documents}{number\ of\ relevant\ documents} \end{aligned} \tag{5}$$

We have used a combined measure, the 'average precision at 11 recall points' (Salton and McGill 1983 [17]). This method has been adopted by the scientific community thanks to the work of Cleverdon, to the Cranfield project (Lesk, 1997 [13], Section 7.6), and to the work of Sparck Jones (1977 [20]).

This measure is computed by fixing the recall at the values (0.0, 0.1, ..., 1.0), by interpolating the precision at these values and by computing the average at the 11 precision values obtained by interpolation. To compute the precision at a given recall value, the list of retrieved documents is cut at the corresponding number of relevant documents. For example, to compute the precision at a recall value of 0.3, the list of retrieved documents is cut as soon as 30% of the relevant documents have been retrieved. If the entire list contains less that 30% of the relevant documents, this value is obtained by an interpolation making the precision depend on the recall in a monotonous way.

Specific Problem. Given a set of queries with known relevance judgments, how can this information be used to improve the retrieval effectiveness of the search system over time ?

The information contained in the relevance judgments of past queries can sometimes be used to increase the performance of the system on future requests. This process of learning is known as the 'relevance feedback'. The methods in this category can be classified by the object they modify: some will try to improve the query representation (Dillon and Desper 1980 [5], Salton and Buckley 1990 [18]) and others the documents indexing (Salton 1971 [15], Vrajitoru 1997 [22]).

In our work, we have chosen to improve the document representation using a form of relevance feedback. To apply the GAs to this context, the genetic individuals must contain a representation of the whole collection.

Gordon (1988 [8]) has applied GAs to a similar problem by improving the indexing of one document at a time. In this case, a genetic individual is a particular description of a document.

If the collection is large, the cost of improving the document descriptions one by one can become too large. Considering this, in our model a genetic individual contains a particular description of all the documents in the collection.

There are various ways to describe a document, and even two indexers would give different answers to this problem. Several sources of information can be taken into account, like the various logical sections of the document (the title, the abstract, etc.) or the relevance feedback (our special interest). The idea is to make all these sources of information compete with each other with the help of GAs, and hope that the collection description coming out of this operation will be significantly better than what we have started with.

We will now mathematically define the notions of document and collection description. For a given document d_j, where $j = 1 \ldots m$, and a set of terms t_k where $k = 1 \ldots n$, a description of d_j takes the form:

$$d_j = < t_{1j},\ t_{2j},\ \ldots\ ,\ t_{nj} > \tag{6}$$

The value t_{ij} shows the importance of the term t_i in description of the document d_j and comes from the $ntf \cdot nidf$ indexing (Equation 3). For performance reasons, we have discretized the t_{ij} values into the integer interval [0,10] using a histogram of the weight values. These new term weights are coded on 4 binary genes using the canonical transformation from the base 16 to the base 2. This operation opens the way for higher weights than those obtained by indexing the collections, but it does not present any technical problem.

After the discretizing operation, each pair (document, term) is represented by four binary genes. Thus, four '0' genes mean that the term is absent from the document description, but the '1' genes contain now more information than the presence of the term in the document description.

By putting together the description of all the documents in the collection we obtain an individual (chromosome):

$$ind = < d_1,\ d_2,\ \ldots\ ,\ d_m > = \begin{pmatrix} t_{11},\ \ldots, t_{n1} \\ t_{12},\ \ldots, t_{n2} \\ \ldots \\ t_{1m}, \ldots, t_{nm} \end{pmatrix} \tag{7}$$

We have noticed that in these individuals, the number of ‘1’ values is significantly smaller than the number of ‘0’ values. The reason for this is in the fact that the average number of terms per document is smaller than the total number of terms in the collection (see Table 1), and the ‘1’ genes only appear in the 4 genes representing a term present in a document. Concretely, the number of ‘1’ values in the matrix from Equation (7) represent around 1% of its size for the CACM collection, and around 2% of its size for the CISI collection. This particularity leads us to represent the individuals in the rare matrix form, which is nothing else than the usual indexing of the collection.

Based on the average precision at 11 fixed recall points (Salton and McGill 1983 [17]), there are two possibilities to compute the fitness function : recurrent and transient.

According to the *recurrent* method, for each new individual, the fitness function is computed as an average over all the test queries, of the average precision at 11 recall points. Thus, the size of the individual is equal to the total number of terms in the collection multiplied by the number of documents in the collection. Our previous research has shown that in this case, the problem size is too large for the GA to be able to significantly improve the performance in a reasonable amount of time (Vrajitoru 1997 [22]).

In the *transient* approach, the GA considers only one query at a time. For each individual, its fitness value is computed as the average precision at 11 recall points considering its answer to the current query. The size of the individual is equal to the number of terms present in the indexing of the current query multiplied by the number of documents in the collection and by 4. As the individual size decreases, the GA can perform a more effective search.

The GA starts from a new initial population for each query, and selects the best individual obtained after a given number of generations. In the end, the performance of the experiment is computed as the average of the results obtained by the best individual for each query. We have already obtained interesting results with this approach (Vrajitoru 1997 [23]), that we extend in the present research.

2.3 Initial populations

There are various ways to construct the initial population, but we are confronted with a special constraint. As the goal of this research is to estimate whether it is better to have a large population or many generations, we have to build the initial population in such a manner that we can vary its size without loosing or adding information. This means that, for any position in the individual, the set of genes corresponding to that position in all the individuals from the population should be the same, independently of the population size.

In our case, as the genes are binary, the constraint can be expressed by the fact that if a gene is equal to '0' in an individual from one starting population of the family, than any other starting population must contain at least one individual with the same gene equal to '0'. This can be expressed by the fact that the *and* operation applied to that particular position to all the individuals in each starting population must be constant for that family.

The following two starting populations contain the same information concerning the '0' values:

	1011110111
	1001011110
1100010110	1010110110
1001011010	1011011011
1001110110	1001010110
1000010010	1000010010

To obtain the result on the last line, we have applied the *and* operation on each column.

If we impose the same constraint for the '1' values, we must apply the *or* operation on each column. The two populations we have considered do not contain the same information concerning the '1' values:

	1011110111
	1001011110
1100010110	1010110110
1001011010	1011011011
1001110110	1001010110
1101111110	1011111111

We can now express mathematically the two constraints by the following. If $G \subset \mathbb{N}$ is an arbitrary set of integer numbers, and $\{P_{sg}, sg \in G\}$ is a family of starting populations with size sg, where sg varies inside G, and if we denote the individuals in each starting population by:

$$P_{sg} = \{ind_{sg1},\ ind_{sg2},\ \ldots,\ ind_{sg\ sg}\},$$

then we must have:

$$ind_{sg1} \text{ and } ind_{sg2} \text{ and } \ldots \text{ and } ind_{sg\ sg} = ind_{const0},\ \forall sg \in G \tag{8a}$$

$$ind_{sg1} \text{ or } ind_{sg2} \text{ or } \ldots \text{ or } ind_{sg\ sg} = ind_{const1},\ \forall sg \in G \tag{8b}$$

where ind_{const1} is a constant representing the entire set of genes equal to '1' in the individuals from the starting population, like in the second example, and ind_{const0} is also a constant representing the same concept for the '0' values, as in the first example.

Equation (8b) expresses the fact that the union of all the genes equal to '1' in any of the individuals of a population must be the same for any population in the family. The condition (8a) incorporates the similar idea for the '0' genes.

In our case, we have only insured that the condition (8b) holds, for two reasons:

- the '1' values appear in cases of presence of a term in a document description and we are more interested in what is present than in what is absent,
- the number of '1' values is much smaller than the number of '0' values, but have a greater impact on the fitness function.

We have found two ways to construct the family of starting populations satisfying the (8b) constraint, and we named them as the 'past queries' population and the 'empty' population.

The question that arises is how to form the first generation. The $ntf \cdot nidf$ weighting scheme of the collection (Equation 3) provides us with an individual that we called 'automatically indexed'. As it represents the baseline solution we want to improve, this individual will be an element of each starting population, for each of the construction methods we chose .

To form other individuals, the *past queries* population contains one automatically indexed individual and a variable number of individuals built from the known relevance judgments of past queries in the following manner:

```
for $id = 1$ to $sg - 1$
    $individual_{id} = \bar{0}$
$id = 1$;
for each ( query $q$)
        for each (term $t_j \in q$ with weight $w_{qj}$)
            for each (relevant document $d_i$ for $q$)
                $t_{ij} = w_{qj}$ in $individual_{id}$;
                $id = id \bmod (sg - 1) + 1$;
$individual_{sg}$ = the automatically indexed individual
```

where the *mod* operator represents the rest of the integer division, usually called *modulo*.

According to this strategy, the first individual in the population is the automatically indexed one. Then the entire set of tuples (*query, term, relevant document*) is partitioned to form the rest of the population (from $individual_1$ to $individual_{sg-1}$). The partition is accomplished with the round-robin strategy, which consists in adding the first tuple to the first individual, the second tuple to the second individual, and so on. When we arrive at the last individual, we start again from the first one. The operation of adding the tuple (q, t_j, d_i) to an individual consists in setting the 4 genes corresponding to t_{ij} to the 4 binary values in w_{qj}. This means that the term t_j will have a weight in the relevant document d_i equal to its weight in the query q.

To check that the condition (8b) is fulfilled, we can notice first that the automatically indexed individual is present in any starting population, so we must only check the condition (8b) for the $sg - 1$ other individuals. We can notice that the *or* operation applied to each position in one of the tuples (*query, term, relevant document*) gives the result '1', and applied to any other position it has the result '0'. This means that the condition (8b) is fulfilled.

The *empty* starting population uses again the automatically indexed individual and a variable number of individuals having all genes equal to 0. We can clearly duplicate this kind of individual without changing the information contained in the initial population according to Equation (8b). We have called this starting population empty because these individuals contain nothing in the rare matrix representation, and they also give no information about the content of the documents in the collection.

2.4 Evaluation methodology

In our research we have paid a special attention to the evaluation issues.

The first important question concerns the evaluation of the fitness function for each individual. If the GA knows the relevance judgments of the current query, its evolution will be biased. To remove this bias, we have simulated the user's implication in the genetic evolution by showing him the first 30 documents appearing in the retrieved list for each new individual.

Concretely, the list of relevant documents for the current query is empty in the beginning, and each relevant document found in the top 30 of a list retrieved by each of the individual is added to this list as the generations are constructed. When the genetic evolution is over, the best individual from the last generation is evaluated according to the complete list of relevance judgments, and these result are presented in all the tables that follow.

A second question concerns the 'past queries' family of starting populations.It is obvious that is the relevance judgments of the current query are used to build the initial population, the results will be incredibly high from the beginning, but will not show th real learning possibilities of the GA.

In a real situation, the system would keep track of the user's judgments for each submitted request, and use this information to improve the retrieval effectiveness on new queries. To evaluate this assumption, we only dispose of a rather limited number of queries with known relevance judgments in our test-collections, which makes it difficult to evaluate the learning system in an accurate way (Kulikowski and Weiss 1991 [12]). It would not be statistically correct to use the same queries to train and to test the system, so we must clearly state the 'past-future' distinction. This is a well known problem for classifier systems (Efron 1986 [6]).

As the number of samples (test queries) is small (35-50), the 'leaving-one-out' model presents various advantages (Efron 1986 [6]). This method consists in separating one sample (the test set) from all the others (the training set), in repeating the experiment for each sample, and in computing the average result.

The theory shows that this strategy estimates the error rate in a very accurate way, even if the sample set is small.

In our case, the 'leaving-one-out' method is the following:

```
for each query $q_i \in Q = \{q_1, q_2, \ldots, q_s\}$
    $training\_set_i = Q \backslash \{q_i\}$
    $test\_set_i = \{q_i\}$
    build a number of generations based on $training\_set_i$
    $result_i$ = evaluate the best individual from the last generation
                 using $test\_set_i$
$result = \frac{1}{s} \sum_{i=1}^{s} result_i$
```

In this algorithm, Q is the entire set of queries with known relevance judgments. The evaluation of the best individual uses the average precision at 11 recall points as described in Section 2.2. The meaning of this algorithm is that for each query, the construction of the starting population will use the relevance judgments of all the queries except the current one. The relevance judgments of the current queries are used to evaluate each individual created by the GA in the way we have described in the beginning of this subsection. This method is therefor fair and unbiased.

3 Large population size or many generations ?

The number of individuals contained in the initial population is an important parameter for the GAs. Usually, this choice is limited by the available memory, especially in our context where the individual size L is relatively large. In this section we analyze the influence of this parameter on the performance of the GAs, first from an experimental perspective and second in a theoretical way. In other terms, for the same computational effort, should the results be better when starting with a large population, or when the GA explores many generations?

3.1 Experimental approach

We have already described the main aspects of the problem representation, parameter settings, and some evaluation issues in the previous section. In this subsection, we are concerned by two parameters, namely, the population size and the number of generations. To evaluate our experiments correctly, two essential conditions must be respected:

- The number of individuals generated on the whole in one run must be the same in all experiences within the same family of starting populations. This value is equal to the number of generations multiplied by the number of individuals by generation. This obvious condition is the expression of the goal of our experiences:

  if $P = \{P_{sg}, sg \in G\}$ is a family of starting populations of size sg,

  then we must have $\forall sg \in G,\ sg * number\ of\ generations = const_P$

- The information contained in any initial population from a family must be the same, no matter what its size is, otherwise we could not fairly compare their results. In the previous section we have expressed this constraint with Equation (8b). We have also shown that the two families of starting populations ('past queries' and 'empty') both fulfill this condition.

Tables 2 and 3 present the results of the experiments on the two families of starting populations with the number of generations multiplied by the population size ($const_p$) being equal to 80 in all runs. The choice of this constant is essentially due to limitations concerning memory consumption and computational time.

As mentioned in the previous section, the numbers in Tables 2 and 3 represent the average precision at 11 recall points computed in a transient way (see Section 2.2).

The baseline performance represents the fitness value of the best individual from the starting population. For the *empty* family of starting populations, this is the automatically indexed individual.

For the past queries population, the baseline performance varies with the population size, and has been mentioned for each experiment. For both populations, the number inside parenthesis represents the percentage of change from the baseline. Usually, if this percentage is equal or greater to 5%, the difference will be considered as significant (Sparck Jones [20]). Finally, all these results represent the performance of the best individual occurring in the last generation, which is also the best individual among the 80 individuals totally generated, as our GA is monotonic (see Section 2).

Table 2. Results obtained from the ‘past queries’ family of initial populations

Population size/	CACM		CISI	
number of generations	baseline	best individual	baseline	best individual
2/40	36.26	37.24 (+2.71%)	20.29	22.15 (+9.17%)
4/20	35.37	37.37 (+5.65%)	20.01	21.71 (+8.53%)
6/13	35.10	37.96 (+8.15%)	20.80	23.21 (+11.61%)
8/10	34.42	38.16 (+10.84%)	20.52	24.90 (+21.34%)
10/8	35.48	39.43 (+11.15%)	20.48	24.14 (+17.87%)
14/6	34.54	40.62 (+17.61%)	20.55	24.77 (+20.5%)
16/5	34.72	37.96 (+9.34%)	20.54	24.24 (+18.02%)
20/4	34.36	41.61 (+21.11%)	21.48	24.96 (+16.23%)

Table 3. Results obtained from the ‘empty’ family of initial populations

Population size/	best individual	
number of generations	CACM	CISI
baseline	32.70	19.83
2/40	33.05 (+1.09%)	21.06 (+6.21%)
4/20	33.59 (+2.74%)	21.86 (+10.24%)
6/13	35.18 (+7.60%)	21.75 (+9.70%)
8/10	36.00 (+10.11%)	22.88 (+15.38%)
10/8	36.17 (+10.63%)	22.85 (+15.26%)
14/6	36.71 (+12.28%)	23.73 (+19.68%)
16/5	37.65 (+15.16%)	22.81 (+15.04%)
20/4	38.30 (+17.13%)	22.88 (+15.40%)

We would like to interpret these results to deduce the total gain in performance obtained by variation of these parameters. For this, we have compared, in each family of starting populations, the parameter values having shown the worst and the best performance, as shown in Table 4.

Table 4. The best and the worst parameter values

	CACM		CISI	
	past queries	empty	past queries	empty
worst parameter values	2/40	2/40	4/20	2/40
performance	37.24	33.05	21.71	21.86
best parameter values	20/4	20/4	20/4	14/6
performance	41.61	38.30	24.96	23.73
difference between them	+11.73%	+15.87%	+14.97%	+12.69%

The parameter values of 20 individuals by generation and 4 generations produce almost always the best performance. The exception is the empty population for the CISI collection where the best performance is given by a population of 14 individuals and 6 generations.

The case of 2 individuals with 40 generations represents the worst choice for almost all the experiences, except for the past queries population for the CACM collection, where the worst results are given by the case of 4 individuals by generation and 20 generations.

From Table 4 we can conclude that a larger population size is a better choice than many generations. We should also remark that the difference between the worst and the best performances is significant.

To inquire more about the meaning of this conclusion, an overall measure like the mean may hide some irregularities. We have used the fact that each of these results is an average over 50 queries (CACM) and 35 queries (CISI). As a consequence, more comparison measures can be imagined based on an analysis query by query. Thus, Table 5 presents a more detailed comparison following this idea.

First, we want to compare the average best and worst parameter choices to each other for each query. More precisely, we would like to know the number of queries where each of the parameter choices has shown better performance than the other in a simple and significant way. We have expressed these measures by the two first questions in Table 5. For example, on the CACM collection and the 'past queries' starting population, the average best parameter choice performs better than the average worst parameter choice on 38 queries. The reverse happens for 10 queries. As the total number of queries is 50, we can deduce that their performance is equal on 2 queries.

Table 5. Analysis query by query

Number of	CACM		CISI	
queries	past queries	empty	past queries	empty
Total	50		35	
Which population is better ?				
the best	38	44	24	30
the worst	10	1	9	4
Which population is significantly better ?				
the best	24	28	14	22
the worst	7	1	5	2
Does the population improve the baseline ?				
the best	48	45	33	33
the worst	10	9	23	18
Does the population significantly improve the baseline ?				
the best	32	30	24	27
the worst	7	5	16	8

Second, we thought it interesting to know how each parameter choice improves the baseline performance on each query. This measure gives the next two questions in Table 5. For example, on the CISI collection and the ‘empty’ starting population, the average worst parameter choice improves the baseline performance in a significant way on 8 queries.

The new comparison measures enforce our conclusion, that large starting populations are a better choice than many generations.

3.2 Theoretical analysis

In this subsection we intend to give a partial explanation of the experimental results by theoretically analyzing some of the implications of the variation of the two parameters.

Convergence .

The first factor that is strongly influenced by the population size is the convergence of the genetic population to an individual representing an optimum for the fitness function. This phenomenon is the greatest danger for the GAs, especially when the dominate individual is a suboptimal solution. The evolutionary potential of a population is closely related to having very different parents to explore different solutions and usually the mutation rate is too small to ensure it.

We will now consider the case where an individual in the starting population has an important fitness advantage over the others. We will compute its expected number of occurrences in future generations under the hypothesis of the fitness-proportionate selection.

Let i_{max} be an individual whose fitness value is f_{max}. The population of size sg contains $sg - 1$ more individuals of average fitness value f_{min}, where $f_{max} \gg f_{min}$. Let e_k be the expected number of occurrences of i_{max} in generation number k. The fitness-proportionate selection tells us that the probability that i_{max} gets selected in one selection operation is proportionate to its fitness value:

$$P(i_{max}) = \frac{f_{max}}{f_{max} + (sg-1) \cdot f_{min}} \tag{9}$$

According to the fitness-proportionate selection, the probability that an individual is selected in one selection operation is equal to its fitness value divided by the sum of the fitness values of all the individuals in the population. In Equation (9), the denominator on the right side is equal to this sum.

If there are e_k occurrences of i_{max}, then $P(i_{max})$ is multiplied by this number and the denominator also changes according to it:

$$P(i_{max}) = \frac{e_k \cdot f_{max}}{e_k \cdot f_{max} + (sg - e_k) \cdot f_{min}} \tag{10}$$

As $P(i_{max})$ is the probability of selecting i_{max} in one selection operation, and on the whole we have sg selection operations, then we can express e_k as a recurrent sequence:

$$\begin{aligned} &e_0 = 1 \\ &e_{k+1} = sg \cdot P(i_{max}) = sg \cdot \tfrac{e_k \cdot f_{max}}{e_k \cdot f_{max} + (sg-e_k) \cdot f_{min}} \end{aligned} \tag{11}$$

The sequence e_k converges in two situations, that are if the sequence is monotonically ascendant or descendant, giving two possible limits, e_{lim1} and e_{lim2}. These values can be computed by imposing the following condition:

$$\begin{aligned}
&e_{k+1} = e_k \Rightarrow \\
&\tfrac{sg\cdot e_k\cdot f_{max}}{e_k\cdot f_{max}+(sg-e_k)\cdot f_{min}} = e_k \Rightarrow \\
&a)\ e_{lim1} = 0, \text{ or, by division with } e_k \neq 0 \\
&b)\tfrac{sg\cdot f_{max}}{e_k\cdot f_{max}+(sg-e_k)\cdot f_{min}} = 1 \Rightarrow \\
&sg \cdot f_{max} = e_k \cdot f_{max} + (sg - e_k) \cdot f_{min} \Rightarrow \\
&sg \cdot (f_{max} - f_{min}) = e_k \cdot (f_{max} - f_{min}) \Rightarrow \\
&\text{if } f_{max} \neq f_{min} \text{ then } e_{lim2} = sg
\end{aligned} \tag{12}$$

From Equation (12) we can deduce that if the sequence is monotonically ascendant, then it converges to sg, the population size, and if it is monotonically descendant, it converges to 0. The monotony condition can be expressed by:

$$\begin{aligned}
&e_{k+1} \geq e_k \Leftrightarrow \\
&\tfrac{sg\cdot e_k\cdot f_{max}}{e_k\cdot f_{max}+(sg-e_k)\cdot f_{min}} \geq e_k \Leftrightarrow \\
&\tfrac{sg\cdot f_{max}}{e_k\cdot f_{max}+(sg-e_k)\cdot f_{min}} \geq 1 \Leftrightarrow \\
&sg \cdot f_{max} \geq e_k \cdot f_{max} + (sg - e_k) \cdot f_{min} \Leftrightarrow \\
&(sg - e_k) \cdot f_{max} \geq (sg - e_k) \cdot f_{mim} \Leftrightarrow \\
&f_{max} \geq f_{mim}
\end{aligned} \tag{13}$$

To solve the inequality (13), we have used the facts that $e_k > 0$ and that $sg - e_k > 0$. It is clear that $e_k \geq 0$ and $sg \geq e_k$. Both equalities happen in cases of convergence, as shown in Equation (12). We have assumed the strict inequalities because we are checking the monotonicity of the sequence before it converges.

And last, the principle of fitness-proportionate selection in the form we have used it, only works in the case where the fitness function f is strictly positive, which is true in our case. Even the empty individuals present a very low but non zero fitness, due to interpolation reasons (see Tables 6 and 7). Thus, we are sure that $e_k \cdot f_{max} + (sg - e_k) \cdot f_{min} > 0$, and we have also used this fact to solve the inequality (13).

Equations (12) and (13) signify that the population converges towards the best individual and that the others tend to disappear. The convergence rate is faster if sg is small.

In our case, the individual obtained from automatic indexing (see Subsection 2.3), presents a much higher fitness value than the others. The difference is even more important for the 'empty' family of populations. In this case, the number of occurrences of the automatically indexed individual for the CACM collection is expected to increase according to the sequence in Table 6.

Table 6. CACM, $f_{max} = 32.70$, $f_{min} = 1.43$

g	e_0	e_1	e_2
2	1	1.92 (95.8%)	2.00 (99.8%)
4	1	3.64 (90.9%)	1.99 (99.1%)
20	1	10.88 (54.4%)	19.26 (96.3%)

From Table 6 we can see that for a population size of 2, in one generation the non-dominate individual has less that 5% of $2 = 0.1$ expected occurrences, which practically means that it vanishes very quickly. For a population of 20 individuals, in one generation the dominate individual occupies about half of the population, which makes the evolution still possible.

For the CISI collection, the convergence rate of the population to the dominate individual in 3 generations is also impressive, but slower because the difference between f_{mim} and f_{max} is less important (see Table 7).

Table 7. CISI, $f_{max} = 19.83$, $f_{min} = 4.65$

g	e_0	e_1	e_2	e_3
2	1	1.60 (80.2%)	1.87 (93.6%)	1.96 (98.0%)
4	1	2.34 (58.5%)	3.43 (85.7%)	3.84 (96.0%)
20	1	3.67 (18.3%)	9.26 (46.3%)	15.71 (78.5%)

Tables 6 and 7 can explain the fact that if the population size is small, the number of queries for which the system's performance improves is very small (see Table 5).

Crossover selection .

The condition (8b) which imposes the fact that all starting populations from a family must contain the same prior information, can lead to the situation where a number of individuals are almost identical. This is actually the case for the 'empty' populations. We can also remark that individuals of high fitness value, like the automatically indexed one, could be selected several times for reproduction, which is equivalent to the previous case.

This also means that we can expect several crossover operations to be applied to the same parents. In this case, we will show that the probability that these individuals produce interesting offspring increases with the population size.

For two individuals, let p_1 be the probability of randomly choosing a 'good' cross position. For example, if the parents were the automatically indexed individual and an empty individual, p_1 would be the percentage of crossover sites that produce children of fitness values superior to their parents.

If p_2 is the same probability for the case where we perform two crossover operations between the same individuals, then p_2 can be computed as the union of two independent events of probability p_1:

$$p_2 = p_1 + p_1 - p_1^2 = 1 - (1 - p_1)^2 \tag{14}$$

We can generalize this sequence by:

$$p_k = p_1 + p_{k-1} - p_1 \cdot p_{k-1} = 1 - (1 - p_1) \cdot (1 - p_{k-1}) \Rightarrow$$
$$p_k = 1 - (1 - p_1)^k \tag{15}$$

As k is the number of performed crossover operations, we have $k \leq sg/2$.

For $sg \to \infty$ and $k \to sg/2$, the sequence $p_k \to 1$ because $1 - p_1 < 1 \Rightarrow (1 - p_1)^k \to 0$.

We can conclude that if the population size is sufficiently large, then the baseline performance may be improved with a big probability. This observation can also be an argument for choosing larger starting populations.

Crossover combination .

In this paragraph we will demonstrate that larger populations can allow better crossover combinations. However, a minimal number of generations is always necessary to obtain the 'best' solution.

First, we notice that the composition of crossover operations is commutative. The proof is obvious, and Figure 1 illustrates it. Thus, the result of the successive application of the crossovers of sites labeled 1 and 2 does not depend on their order.

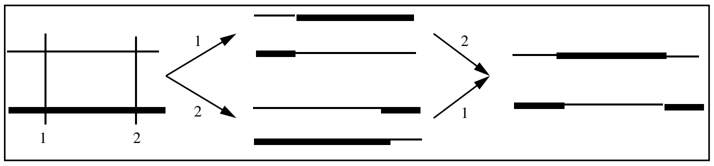

Fig. 1. Two crossover operations

The second step is to show that three crossover operations can be done in two generations. Figure 2 shows the result of three successive crossover operations in three generations.

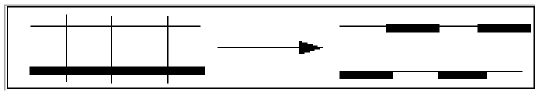

Fig. 2. Three crossover operations

A generation contains only individuals one crossover away from the previous generation. We must find out if a crossover between individuals obtained from two different crossover operations can give the same result as in Figure 2. They can, indeed, and Figure 3 shows how.

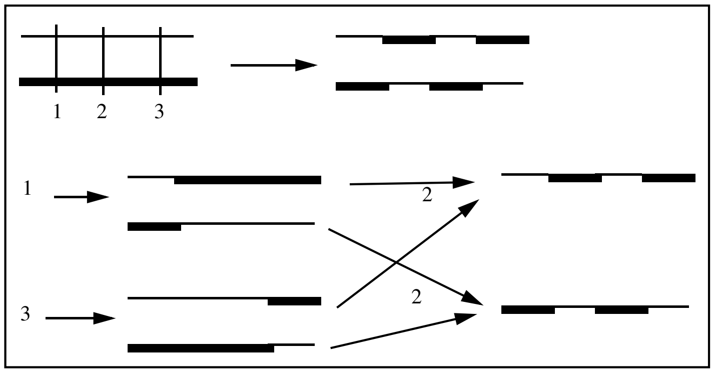

Fig. 3. Crossover combination

Let us consider now an optimal solution found after a number of generations ng. We can express this process as a binary tree with the root representing the optimal solution, where each leaf belongs to the initial population, and where any ancestor node is obtained from its descendants by a crossover operation (Figure 4). In this tree, the ancestor-descendant notation is the reverse of the genetic parent-child notation, and for more clarity, the arrows in Figure 4 show the direction of action of the crossover operations.

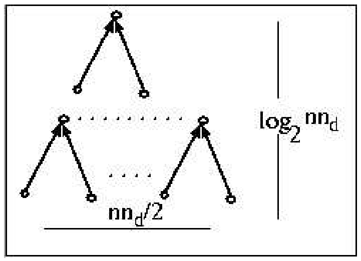

Fig. 4. Binary tree with nn_d nodes

Then the solution is at a crossover distance from the initial population equal to the total number of nodes in the tree nn_d. The number of generations ng represents the depth of the tree. It is well known that the minimal value for ng is $\lceil \log_2 nn_d \rceil$ when the tree is complete. This tells us that by increasing the population size, the tree linking the initial population to the optimal solution can gain width and loose depth, which would make the search more balanced and increase the chances for good performance.

As the crossover operations are arranged in a tree, we know that a tree with nn_d nodes cannot have a depth less that $\lceil \log_2 nn_d \rceil$. This means that to find a given optimal solution from a given starting population, we need a minimal number of generations that do not depend on the probabilistic behavior of the GA. This number is equal to the binary logarithm of the distance between the optimal solution and the starting population in terms of crossover operations.

We can conclude that if the information contained in the initial population is the same, we should expect a limit for the gain in performance we obtain by increasing the population size.

Related to this observation, Figures 5 and 6 show the evolution of the performance according to the population size. We could say that the limit we have predicted is achieved for the CISI collection and the 'empty' starting population, but we think that the parameter values we have used are too small compared to the problem size to allow us to trust this conclusion.

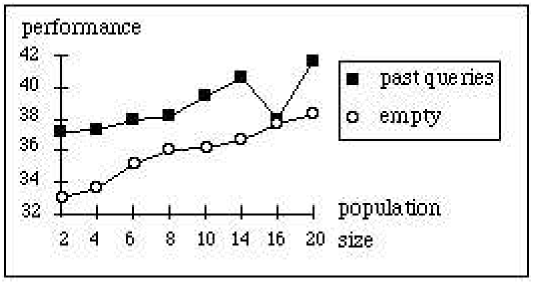

Fig. 5. Plot of results for the CACM collection

The three factors we have analyzed in this subsection also stand for the choice of large populations. If the problem size is large, as for our experiment, the available memory space should affect this choice more than the limitations presented in the last paragraph.

4 Conclusion

When GAs must deal with very large search spaces, as in information retrieval, the difficulty of the problem imposes a fine parameter tuning without

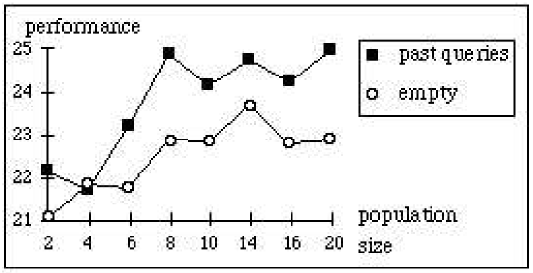

Fig. 6. Plot of results for the CISI collection

which the search for good solutions may simply fail. In this context, this paper has tried to answer the question of whether it is better to build many generations or to start with a large population.

In Section 2 we introduced the general notions underlying the GA approach to information retrieval, and the way GAs can be applied to our problem. We also presented the specific difficulties for this problem and some practical details reducing them. Section 3 presented and discussed the results of our experiences, as well as a mathematical analysis that shows the theoretical implications of the parameter choice we have studied.

Both the experimental and the theoretical investigations lead to the same conclusion. It seams that larger populations are a better choice than many generations, as far as memory space allows it. We have also shown some theoretical arguments against a too small number of generations that could eventually have influenced one of our results (see Figure 6). The comparison between the best and the worst choices has shown that the more difficult a problem is, the more advantage we can take of choosing the right values for the genetic parameters (see Tables 4 and 5).

Our experiences have also proved that GAs may significantly improve the performance of information retrieval systems and that the relevance judgments of past queries can be very useful for this task.

Finally, we can say that although the GAs are widely employed in many fields, various aspects of their behavior are still barely known and can open interesting directions of exploration.

Acknowledgments - This research was supported in part by the FNRS (Swiss National Science Foundation) under grant 20-43'217.95.

References

1. Blair D.C. (1990) Language and Representation in Information Retrieval. Elsevier, Amsterdam (NL).
2. Brassard G., Bratley P. (1994) Fundamentals of Algorithmics. Prentice-Hall.

3. 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.
4. De Jong K., Spears W. (1989). Using Genetic Algorithms to Solve NP-Complete Problems. International Conference on Genetic Algorithms. George Mason University, Fairfax (VA), 124-132.
5. Dillon M., Desper J. (1980) Automatic Relevance Feedback in Boolean Retrieval Systems. Journal of Documentation. **36**, 197-208.
6. Efron B. (1986) How Biased Is the Apparent Error Rate of a Prediction Rule. Journal of the American Statistical Association. **81(394)**, 461-470.
7. Goldberg D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (MA).
8. Gordon M. (1988) Probabilistic and Genetic Algorithms for Document Retrieval. Communications of the ACM. **31(10)**, 1208-1218.
9. Gordon M. (1991) User-Based Document Clustering by Redescribing Subject Descriptions with a Genetic Algorithm. Journal of the American Society For Information Science. **42(5)**, 311-322.
10. Holland J.H. (1975) Adaptation in Natural and Artificial Systems. Ann Arbor, University of Michigan Press.
11. Kraft D.H., Petry F.E., Buckles B.P., Sadavisan T. (1997) Genetic Algorithms for Query Optimization in Information Retrieval: Relevance Feedback. In Sanchez E., Zadeh L.A., Shibata T. (Eds.), Genetic Algorithms and Fuzzy Logic Systems, Soft Computing Perspectives. World Scientific. 155-173.
12. Kulikowski A.C., Weiss M.S. (1991) Computer Systems That Learn. Morgan Kaufmann, San Mateo (CA).
13. Lesk M. (1997) Practical Digital Libraries: Books, Bytes and Bucks. Morgan Kaufmann, San Francisco (CA).
14. Petry F., Buckles B., Prabhu D., Kraft D. (1993) Fuzzy Information Retrieval Using Genetic Algorithms and Relevance Feedback. Proceedings of the ASIS Annual Meeting. 122-125.
15. Salton G. (ed.) (1971) The SMART Retrieval System - Experiments in Automatic Document Processing. Prentice-Hall Inc., Englewood Cliffs (NJ).
16. Salton G., Fox E., Wu U. (1983) Extended Boolean Information Retrieval. Communications of the ACM. **26(12)**, 1022-1036.
17. Salton, G., McGill M. J. (1983) Introduction to Modern Information Retrieval. McGraw-Hill (NY). Chapter 5
18. Salton G., Buckley C. (1990) Improving Retrieval Performance by Relevance Feedback. Journal of the American Society for Information Science. **26**, 361-372.
19. Sanchez E., Pierre Ph. (1994) Fuzzy Logic and Genetic Algorithms in Information Retrieval. Proceedings of the 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing, Iizuka, Japan, 29-35.
20. Sparck Jones K., Bates R. G. (1977) Research on Automatic Indexing 1974-1976. Technical Report. Computer Laboratory, University of Cambridge.
21. Sushil J.L., Gong L. (1997) Augmenting Genetic Algorithms with Memory to Solve Traveling Salesman Problems. Proceedings of the Joint Conference on Information Science. Duke University, 108-111.
22. Vrajitoru D. (1997) Apprentissage en recherche d'informations. Doctoral thesis, University of Neuchâtel, Switzerland.
23. Vrajitoru D. (1998) Crossover Improvement for the Genetic Algorithm in Information Retrieval. Information Processing and Management. **34(4)**, 405-415.

24. Yang J.-J., Korfhage R.R., Rasmussen E. (1992) Query Improvement in Information Retrieval Using Genetic Algorithms. Proceeding of the fith ICGA. 603-611.
25. Yang J.-J., Korfhage R.R. (1993) Query Optimization in Information Retrieval Using Genetic Algorithms: Report on the Experiments of the TREC Project. Proceedings of TREC'1. NIST, Gaitherburgs (MD). 31-58.