Evolutionary Algorithms for Boolean Queries Optimization

Dušan Húsek, Václav Snášel, Roman Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, Prague 8
Czech Republic

Suhail S. J. Owais, Pavel Krömer
Department of Computer Science
VŠB-Technical University of Ostrava
17. listopadu 15, Ostrava – Poruba
Czech Republic

Abstract: Usage of genetic algorithms in the Information retrieval area, especially in optimizing a Boolean query, is presented in this paper. The proposed evolution of Boolean queries should increase the performance of the information retrieval system by means of the precision and recall. Influence of both criteria is thorougly tested for different genetic algorithm settings. The quality improvement achieved by our approach is discussed.

Key–Words: Evolutionary algorithms, Genetic algorithms, Information retrieval, Boolean query.

1 Introduction

Retrieving information employs matching the words in the query against the database index (key-word searching) and traversing the database with the aid of hypertext or hypermedia links. Key-word searches can be made either more general or narrower in scope of the means of logical operators (e.g., disjunction and conjunction).

Key-word searching has been the dominant approach to text retrieval since the early 1960s; hypertext has so far been confined largely to personal or corporate information-retrieval applications. Evolving information-retrieval techniques, exemplified by developments with modern Internet search engines specially on optimizing Boolean queries. Artificial intelligence was used on such techniques that seek higher levels of retrieval precision.

The principal categories of information sources useful in modern information retrieval systems are text, video, and voice. Information filtering is concerned with finding information from unstable collections of documents such as the Internet, so in such a huge and unstable information collection, todays the greatest problem is to find relevant information for the user query. In the information filtering domain, the user query does not consists of a list of words or terms to search for but rather of combinations of words. The most important problem to solve is to optimize the user query and to obtain accurate collection statistics for calculating the term arity.

An information retrieval system is basically constituted of three main components: documentary database, query subsystem and matching or evaluation mechanism [1, 14].

For evaluation of the information retrieval system, measured by effectiveness, two statistics are used, precision and recall where these measures are evaluated over a set of documents called a collection of documents. All documents in this collection are divided into four subsets depending on user query: (1) Relevant set R_1 ”*the set of documents that are relevant to the user query*”, (2) Retrieved set R_2 ”*the set of documents that are returned to the user*”, (3) Relevant-Retrieved set R_3 ”*the set of documents that are retrieved and relevant to the user query*”, (4) and the rest set R_4 of documents ”*the set of documents that are not relevant and not retrieved*”. The quality of information retrieval is measured in terms of the following measures. The precision measure P is calculated as a ratio of the retrieved documents to the ones that are relevant to the user query, and the recall measure is calculated as a ratio of the relevant documents to the ones that are retrieved to the user query [1, 14, 7] $Recall = \frac{R_3}{R_1}$, $Precision = \frac{R_3}{R_2}$.

2 Evolutionary Algorithms

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution, which applies the principles of evolution found in nature to the problem of finding an optimal solution to a solver problem.

An evolutionary algorithm is a generic term used to indicate any population-based optimization algorithm that uses set of operators or mechanisms inspired by biological evolution, such as reproduction, mutation and recombination. Candidate solutions to the optimization problem play the role of individuals in a population, and the cost function determines the environment within which the solutions ”live” [6].

Evolution of the population then takes place after the repeated application of the above operators. Genetic algorithm is the most popular type of evolutionary algorithms [8, 9].

Most of the search engines in the internet depend on the user query and operate an information retrieval system to get the response of the user query request. Where the user query consists of a set of terms and set of logical operators; especially and, or, of, and not operator see [7]. The motivation of our work is to do an evolution of the Boolean queries using genetic programming in the information retrieval [2, 3, 16].

This section will present implementation of information retrieval using genetic algorithms (for SQL we can see [17, 13, 10, 4]). The GA is generally used to solve optimization problems [8, 11]; it starts on an initial population with a fixed size of chromosomes "*P-chromosomes*". Each individual is coded according to the chromosome length, where genes are allocated in each position in a chromosome with different data types, and each gene values called allele. In information retrieval, the queries for relevant documents represented each individual or chromosome, and all document described by a set of terms. Description d_i for document D_i, where $i = 1 \dots l$, the set of terms for D_i is T_j, where $j = 1 \dots n$, so $d_i = (w_{1_i}, w_{2_i}, \dots, w_{n_i})$. The value for each term will be 1 if this term exists in the document or 0 if not (Note: another weight for terms was mentioned in the paper [15]), it indicates that the indexing function, which is maps, a given index term t and a given document d is $F : D \times T \rightarrow [0, 1]$.

Defining a query will be a combination of a set of terms and a set of Boolean operators and, or, xor, not and of. The query set Q defined as a set of queries for documents, defines the query processing mechanism by which documents can be evaluated in terms of their relevance to a given query [12].

In this work, we develop a genetic program for implementing GA with variable length of chromosomes and mixture symbolic of information, like real values and Boolean queries values.

Each chromosome from the initial population represented a tree structure for one query; an index was defined for each node in the tree. Genetic operators were applied to individuals. Queries will be encoded as trees, where each chromosome contains a set of genes, and each gene mentioned to be a node in a tree and the value for each node known as allele. An example that shows a query encoding for chromosome in the population shown in Fig. 1.

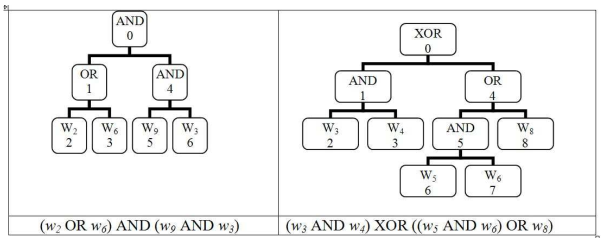

Fig. 1 *Chromosome encoding form a query.*

Genetic operators has been used in our work to evolve Boolean queries. Presented for these operators are "Fitness, Selection, Crossover, and Mutation".

For each individual the value of precision and recall will be computed and known as fitness values, see $RecallFitness\ E_1$ and $PrecisionFitness\ E_2$ respectively, which depends on the number of relevance documents r_d in the collection of documents to the user query, in the number of retrieved document f_d, and α and β being arbitrary weights. Where α and β are added specially to precision fitness function [12].

$$E_1 = \frac{\sum_d [r_d \times f_d]}{\sum_d [r_d]},$$

$$E_2 = \frac{\alpha \sum_d [r_d \times f_d]}{\sum_d [r_d]} + \frac{\beta \sum_d [r_d \times f_d]}{\sum_d [f_d]}$$

Very simple implementation is sufficient. Two individuals with the best fitness values are chosen from a population, but if there are more than two individuals with the same highest fitness values, then two of them will be chosen randomly. The two selected chromosomes will be called parent1 and parent2 and they will be used for producing two new offsprings.

Offsprings must have some inheritance from the tow parents; single point crossover will do that by exchange of the subtree from parent1 with the subtree from parent2. Positions for exchanging subtree1 and subtree2 will be selected randomly. In our work we define the selection of the position for the subtree as follows:

1. The root node of the tree.

2. Each Boolean operator node.

3. Each leaf from the tree.

An example was shown in Fig. 2.

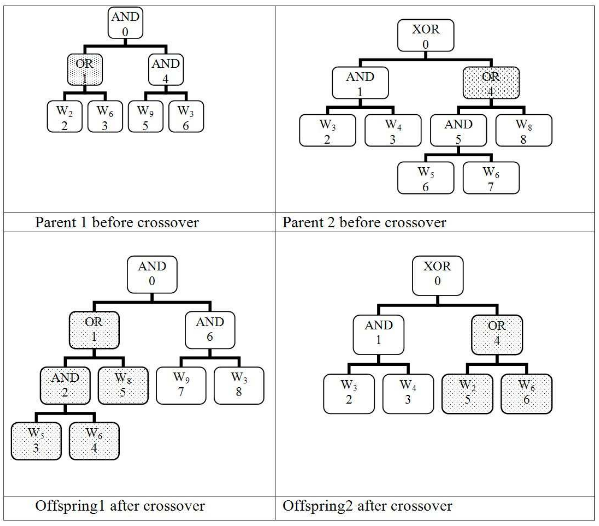
Parent 1 before crossover
Parent 2 before crossover
Offspring1 after crossover
Offspring2 after crossover

Fig. 2 *Single point crossover, Randomly select the nodes.*

Mutation, random perturbation in the chromosome representation, should assure that the current generation is connected to the entire search space, and it is necessary to introduce new genetic material into a population that has a stabilized level [12]. In our implementation, the mutation operator works as the most important operator for the evolutionary learning of the Boolean query.

Each node from the new offsprings may be mutated; that depends on the mutation value (0.2). And we work with different types of mutation shown below:

- Mutation on Boolean operator: randomly exchanging one operator to another but both must be from the same arty, such as any exchange in $(\mathrm{and}, Or, Xor$, and $\mathrm{of})$ are allowed.
- Mutation on term node or leaf node: changing one term selected randomly from the offspring by any other one but the other one will be one of:
  - The terms in a given collection of documents
  - The terms in an initial population.
  - A specified list of terms.
  - The terms appeared in the user's query.
- Mutation by inserting or deleting the operator between two nodes in the offsprings

Where mutation was implemented in this way: For a given offspring one node randomly is selected and for this node there are two possibilities – to mutate into another one or to apply inserting a unary operator before it or delete it if and only if the unary operator is of this node. Some examples were shown in Fig. 3.

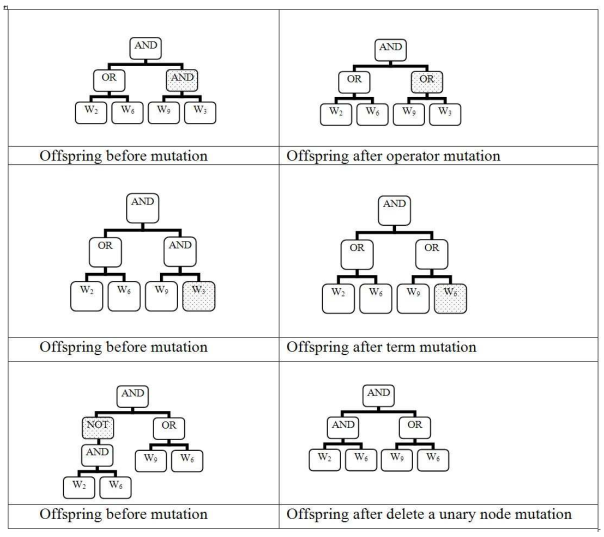
Offspring before mutation
Offspring after operator mutation
Offspring before mutation
Offspring after term mutation
Offspring before mutation
Offspring after delete a unary node mutation

Fig. 3 *Single point crossover, Randomly selected nodes.*

3 Experiments

Presenting our work now, we will show the results of our research on evolutionary learning of Boolean queries was done.

We developed a genetic program to process some experiments over a set of Boolean queries and various collections of documents, and the documents are various number of words; all collections used in our experiments are described in Tab. I:

Collection	Words	Docs
10x30	30	10
200x50	200	50
5000x1000	1000	5000

Tab. I *Document Collections*

The Genetic program ended when a given number of generations was reached or when all chromosomes in the population had a maximum possible value of the fitness function, where the maximum values for precision and recall are $\alpha + \beta$ and 1, respectively. We also used three types of mutation as described above. All the experiments were done a few times with the same options to see the differences in the results, because the results are affected by probability used during the genetic program process. In all the experiments the following fixed options were used: the arbitrary weights for $\alpha = 0.25$, and $\beta = 1.0$, crossover value = 0.8.

The mutation value is a probability of the applying mutation operator on an offspring. In this experiment we observed how the change of the mutation value affects the result of the genetic algorithm process. The type of mutation was described above. Additional options for this experiment: user query is: $(w_6 \text{ and } w_8) \text{ and not } w_{10}$, collection name: 10x30, used fitness measure: precision, 200 generations.

All terms from the initial population were used for mutation of leaves and the results obtained as shown in Tab. II:

Mutation	Generations	Precision	Recall
0.1	200	0.75	1.00
	51	1.25	1.00
0.2	24	1.25	1.00
	40	1.25	1.00
0.3	27	1.25	1.00
	17	1.25	1.00
0.4	25	1.25	1.00
	118	1.25	1.00
0.5	45	1.25	1.00
	135	1.25	1.00

Tab. II *Results with Mutation applied over leaves and terms from all initial population.*

Nearly in all experiments, the values of the fitness function for precision for all chromosomes reached in the final population to be as maximum of the precision value 1.25, and the same for the recall fitness value is 1.00, where the number of generations was variant.

All terms form the user query only used for mutation of leaves, and the results were shown in Tab. III.

Mutation	Generations	Precision	Recall
0.1	20	0.75	1.00
	200	0.75	1.00
0.2	200	0.75	1.00
	200	0.75	1.00
0.3	200	0.75	1.00
	200	0.75	1.00
0.4	200	0.75	1.00
	200	0.75	1.00
0.5	113	1.25	1.00
	200	0.75	1.00

Tab. III *Results with Mutation applied over leaves and terms from the user query only.*

In this case, nearly the maximum number of generations was reached to get the best solution especially when the precision fitness function was used.

All terms forming the whole collection were used for the mutation of leaves, and the results were shown in Tab. IV. Where the maximum number of generations was reached in some experiments and the value of precision was reached in an other maximum.

Mutation	Generations	Precision	Recall
0.1	200	0.75	1.00
	28	1.25	1.00
0.2	200	0.75	1.00
	58	1.25	1.00
0.3	65	1.25	1.00
	11	1.25	1.00
0.4	187	1.25	1.00
	143	1.25	1.00
0.5	21	1.25	1.00
	42	1.25	1.00

Tab. IV *Results with Mutation applied over leaves and terms from whole collection.*

When we used recall as a fitness function, all chromosomes in the final population had the same (maximum) value of recall, but the values of precision are mostly various; and the best of them are described in the tables bellow, where Tab. V shows the results when the mutation over leaves used only terms from the user's query. Tab VI shows the results when the mutation over leaves used terms from the initial population and Tab. VII shows the results when the mutation over leaves used terms from the whole population:

Mutation	Generations	Precision	Recall
0.1	5	0.583	1.00
	4	0.583	1.00
0.2	5	0.583	1.00
	5	0.500	1.00
0.3	5	0.500	1.00
	5	0.500	1.00
0.4	5	0.583	1.00
	5	0.500	1.00
0.5	5	0.583	1.00
	6	0.583	1.00

Tab. V *Results with Mutation applied over leaves and terms from the user's query.*

Mutation	Generations	Precision	Recall
0.1	5	0.583	1.00
	6	0.583	1.00
0.2	5	0.500	1.00
	4	0.583	1.00
0.3	5	0.500	1.00
	5	0.500	1.00
0.4	5	0.583	1.00
	8	0.500	1.00
0.5	7	0.583	1.00
	4	0.583	1.00

Tab. VI *Results with Mutation applied over leaves and terms from initial population.*

Mutation	Generations	Precision	Recall
0.1	5	0.583	1.00
	4	0.500	1.00
0.2	5	0.500	1.00
	5	0.500	1.00
0.3	6	0.500	1.00
	5	0.583	1.00
0.4	8	0.583	1.00
	5	0.583	1.00
0.5	5	0.583	1.00
	5	0.583	1.00

Tab. VII *Results with Mutation applied over leaves and terms from whole collection.*

In some cases, especially when we used only the terms from the user query for mutation over leaves and the fitness function was precision, there were worse results than in other cases as shown in Tabs. IV, V, and VI. We increased the maximal number of generations to be 1200 generations and did some experiments with the following options. The results of these experiments are shown in Tab. VIII. (maximum number of generations: 1200, user query: $((\mathrm{not}\, w_{10})\, \mathrm{and} (w_6\, \mathrm{and}\, w_8))$, mutation over leaves use terms from the user query, fitness function is precision).

Mutation	Generations	Precision	Recall
0.1	1200	0.75	1.00
	1200	0.75	1.00
0.2	1200	0.75	1.00
	1200	0.75	1.00
0.3	1200	0.75	1.00
	197	1.25	1.00
0.4	1200	0.75	1.00
	462	1.25	1.00
0.5	25	1.25	1.00
	1200	0.75	1.00

Tab. VIII *Results with Mutation applied over leaves and terms from the user query.*

After increasing the number of generations there was not a big difference in the results because in many cases the best solution had not been reached yet.

The goal of the optimization process of a Boolean query is to get a query with the highest possible values of precision and recall. The results shown above demonstrate that when using precision as a fitness function, the value of recall in the final generation is very high, even when the precision value is not the most possible. But to get these results we often needed a high number of generations. We tested this process over larger collections.

Experiment options: collection name: 200x50, user query: $((\mathrm{not}\, w_{10})\, \mathrm{and} (w_6\, \mathrm{and}\, w_8))$, maximum number of generations: 2000, all terms from the initial population were used for mutation over leaves.

When using precision as a fitness function we reach the highest number of generations without reaching the best value of precision as shown in Tab. IX, and Tab. X shows the results when we used the recall as a fitness function.

Mutation	Generations	Precision	Recall
0.1	2000	0.3779	1.00
	2000	0.3394	1.00
0.2	2000	0.3736	1.00
	2000	1.045	0.18
0.3	2000	0.3736	1.00
	2000	0.36	1.00
0.4	2000	0.3736	1.00
	2000	0.3736	1.00
0.5	2000	0.3736	1.00
	2000	0.4219	1.00

Tab. IX *Results when Precision was used as a fitness function.*

Mutation	Generations	Precision	Recall
0.1	16	0.3050	1.00
	63	0.3050	1.00
0.2	13	0.3050	1.00
	11	0.3050*	1.00
0.3	23	0.3050*	1.00
	15	0.3050	1.00
0.4	11	0.3050*	1.00
	16	0.3050	1.00
0.5	17	0.3050	1.00
	10	0.3050*	1.00

Tab. X *Results when Recall was used as a fitness function.*

$*$ – in these cases the precision value of chromosomes in the final generation was various. The number in the table is the lowest precision value in the population.

4 Conclusions

In this paper, an optimization of the Boolean query over a collection of documents is presented. We focused especially on mutation and on comparison of two fitness measures, precision and recall. Experiments were done over various models of document collections with different types of mutation over the leaves.

After the set of experiments we have obtained the following conclusions. First, when applying the mutation operator on terms in a query, it is necessary to have the largest possible set of terms at disposal for mutation. If only terms from the user query or the initial population were used for mutation, the results were worse than when terms from the whole collection were used. Only then can new queries come into existence, describing the same documents as the user query, but the containing terms not included into the user query or the initial population.

Second, when we are looking for the best optimization of a Boolean query, we should consider the number of operators in the queries in the final population. The query with fewer operators is better than a query with more operators and the same values of precision and recall. This parameter can be important during the whole genetic algorithms process.

Third, the probability of mutation (the mutation value) affects the result of the genetic algorithm process too. A higher mutation value causes higher probability of finding a good query, especially when using precision as a fitness measure.

Fourth, recall seems to be more efficient than precision. Recall as a fitness function returns quickly the expressions describing all documents relevant to the user query, but there are many non-relevant documents retrieved too. Other sides, when using precision as a fitness measure, the results are (especially for larger collections) similar but the number of generations needing to get these results is much bigger.

Acknowledgements

This research has been supported by National Research Program of the Czech Republic under Information Society project no. IET100300414.

References:

[1] Baeza-Yates R., Ribeiro-Neto B.: Modern Information Retrieval. Addison Wesley, New York, 1999.

[2] Cordon O., Herrera-Viedma E., Luque M.: Evolutionary Learning of Boolean Queries by Multiobjective Genetic Programming. J. J. Merelo Guervos et al. (Eds.): PPSN VII, LNCS 2439, pp. 710 719, 2002. Springer-Verlag Berlin Heidelberg, 2002.

[3] Chen H.: A machine learning approach to inductive query by examples: an experiment using relevance feedback, ID3, genetic algorithms, and simulated annealing, Journal of the American Society for Information Science, **49**, 8, 1998, pp. 693705.

[4] Freytag J. C.: A Rule-Based View of Query Optimization. Proceedings of ACM-SIGMOD, 1987, pp. 173-180.

[5] Goldberg D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Massachusetts: Addison-Wesley, 1989.

[6] Hartmut P., Evolutionary Algorithms: Overview, Methods and Operators. GEATbx version 3.5. 2004.

[7] Korfhage R.t R.: Information Storage and Retrieval. John Wiley & Sons, Inc., 1997.

[8] Mittchel M.: An Introduction to Genetic Algorithms. A Bradford Book The MIT Press, 1999.

[9] Melanie M., Stephanie F.: Genetic Algorithms and Artificial Life. Santa Fe Institute Working Paper 93-11-072, 1993.

[10] Kim W.: On Optimizing an SQL-like Nested Query. ACM Transactions on Database Systems, **7**, 3, 1982, pp. 443-469.

[11] Koza J.: Genetic programming. On the programming of computers by means of natural selection, The MIT Press, 1992.

[12] Kraft D. H., Bordogna G., Pasi G.: Fuzzy Set Techniques in Information Retrieval, in Bezdek, J. C., Didier, D. and Prade, H. (eds.), Fuzzy Sets in Approximate Reasoning and Information Systems, **3**, The Handbook of Fuzzy Sets Series, Norwell, MA: Kluwer Academic Publishers, 1999.

[13] McGoveran D.: Evaluating Optimizers. Database Programming and Design. January 1990, pp. 38-49.

[14] Rijsbergen C. J.: Information Retrieval (2nd edition), Butterworth, 1979.

[15] Salton G., Buckley C.: Terms-Weighting approach in automatic text retrieval. Information Processing and management, 1988, **24**, 5, pp. 513-523.

[16] Smith M. P., Smith M.: The use of genetic programming to build Boolean queries for text retrieval through relevance feedback, Journal of Information Science, **23**, 6, 1997, pp. 423 431.

[17] Yao S. Bing: Optimization of Query Algorithms. ACM Transactions on Database Systems, **4**, 2, 1979, pp. 133-155.