Proceedings of the Ninth IASTED International Conference
INTERNET AND MULTIMEDIA SYSTEMS AND APPLICATIONS
August 15-17, 2005, Honolulu, Hawaii, USA

USING GENETIC ALGORITHMS FOR BOOLEAN QUERIES OPTIMIZATION

Dušan Húsek and Václav Snášel
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2
Prague 8, Czech Republic
email: dusan@cs.cas.cz

Suhail S. J. Owais and Pavel Krömer
Department of Computer Science
VŠB-Technical University of Ostrava
17. listopadu 15
Ostrava - Poruba, Czech Republic
email: pavel.kromer@vsb.cz

ABSTRACT
Most of information retrieval systems depend on Boolean queries. The performance of an information retrieval system is usually measured in terms of two different criteria, precision and recall. This way, the optimization of any of its components is a clear example of a multiobjective problem. However, although evolutionary algorithms have been widely applied in the information retrieval area, in all of these applications both criteria have been combined in a single scalar fitness function by means of a weighting scheme. In this paper, we deal with using of Genetic algorithms in Information retrieval specially in optimizing of a Boolean query.

KEY WORDS
genetics algorithm, information retrieval, Boolean query, genetic programming.

1 Introduction

Ever since the advent of the public network Internet, the quantity of available information is rapidly rising. One of the most important uses of this public network is to find suitable information for such user query request. In such a huge and unstable information collection, todays greatest problem is to find relevant information to the user query.

Information filtering is concerned with finding information from unstable collections of documents such as the Internet. In the information filtering domain, the user query does not consist of a list of words or terms to search for but rather of combinations of words extracted from various examples. The most important problem to solve is to optimize the significance of the user query and obtaining accurate collection statistics for calculating the term arity.

After using evolutionary techniques for single-objective optimization during more than two decades, the incorporation of more than one objective in the fitness function has finally become a popular area of research.

An information retrieval system is basically constituted of three main components: documentary database, query subsystem and matching or evaluation mechanism [1, 13].

2 Evaluation of Information Retrieval System

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 of documents are divided into four subsets: Relevant set; set of documents that are relevant to the user query, Retrieved set; set of documents that are returned to the user query, and Relevant-Retrieved set; set of documents that are retrieved and relevant to the user query, and finally the rest set of documents; set of documents that are not relevant and not retrieved. Where precision the percentage of the retrieved documents that are relevant to the user query and recall the percentage of the relevant documents that are retrieved for the requested query.

$$Recall = \frac{RelevantRetrieved}{Relevant}$$

$$Precision = \frac{RelevantRetrieved}{Retrieved}$$

In our work we introduce to use Genetic Programming for implementing the Information Retrieval system with Boolean queries, trying to evolve Boolean queries by genetic algorithm.

3 Genetic Algorithms

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 consist of set of terms and set of logical operators; especially and, or, of, and not operator see [6]. For this our motivation in our work is to do the evolution of the Boolean queries using genetic programming in the information retrieval [2, 3, 16].

Genetic Algorithm is an algorithm that used to find approximate solutions to problems that was difficult to solve it through set of methods or techniques inheritance or crossover, mutation, natural selection, and fitness function

that are principles of evolutionary biology in computer science. For more detail about Genetic Algorithms see [5, 15].

4 Genome query encoding

This section will present the implementation of information retrieval using genetic algorithms (for SQL we can see [17, 11, 8, 4, 12]). The GA is generally used to solve optimization problems [7, 9]. GA starts on an initial population with fixed size of chromosomes "*P-chromosomes*". Each individual are coded according to 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, query for relevant documents are representing for each individual or chromosome, and each document described by set of terms. The description d_i for document D_i, where $i = 1 \dots l$, the set of terms for D_i are T_j, where $j = 1 \dots n$, thus $d_i = (w_{1_i}, w_{2_i}, \dots, w_{n_i})$. The value for each term will be 1 if this term exists in the document or 0 if not (Note: about another weights for terms was mention in paper [14]), this indicate that the indexing function that 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 combination from set of terms and set of Boolean operators and, or, xor, not and of. The query set Q defined as set of queries for documents, define the query processing mechanism by which documents can be evaluated in terms of their relevance to a given query [10].

In this work, we develop 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 operated over individuals. Queries will be encoded as trees, where each chromosome contains set of genes, and each gene mention to be a node in a tree and the value for each node known as allele. An example that show query encoding for chromosome in the population shown in Figure 1.

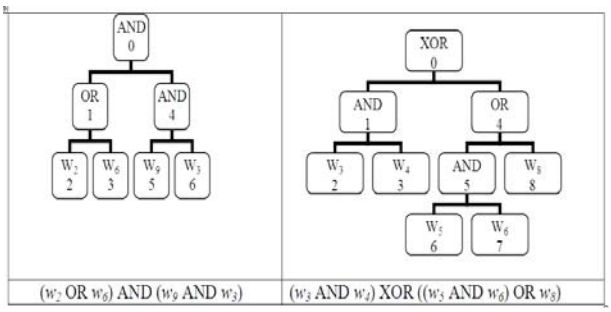

Figure 1. Chromosome encoding form a query

5 Implement Genetic Operators to Evolutes Boolean Queries

Genetic operators used in our work to evolve Boolean queries. Presenting for these operators Fitness, Selection, Crossover, and Mutation follows:

Fitness function operator

For each individual the value of precision and recall will be computed and known as fitness values see $RecallFitnessE_1$ and $PrecisionFitnessE_2$ respectively, this depends on the number of relevance documents r_d in the collection of documents to the user query, number of retrieved document f_d, and α and β are arbitrary weights. Here $PrecisionFitnessE_2$ function is composed from two parts first reflects recall quality and second precision quality. Influence of each part is given by α and β coefficients in precision fitness function [10].

$$ReallFitnessE_1 = \frac{\sum_d [r_d \times f_d]}{\sum_d [r_d]}$$

$$PrecisionFitnessE_2 = \frac{\alpha \sum_d [r_d \times f_d]}{\sum_d [r_d]} + \frac{\beta \sum_d [r_d \times f_d]}{\sum_d [f_d]}$$

Selection operator

Very simple implementation of this operator was sufficient. Two individuals with 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 to produce two new offsprings.

Crossover operator

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

1. The root node of the tree.
2. Each Boolean operator node.
3. Each leaf from the tree.

An example was shown in Figure 2.

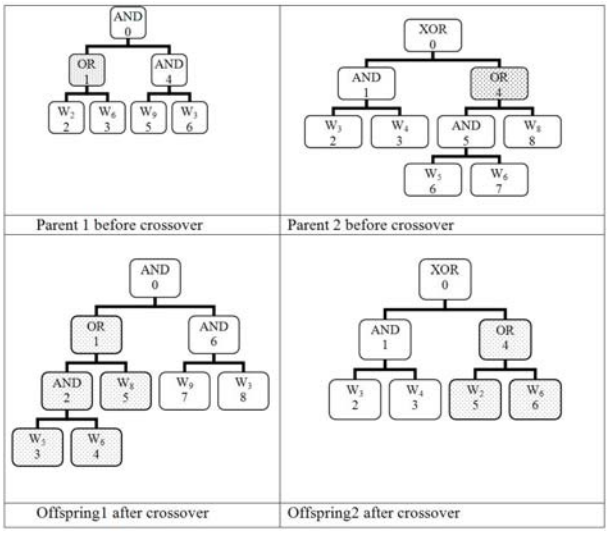

Figure 2. Single point crossover, Randomly select the nodes

Mutation operators

Mutation, random perturbation in the chromosome representation, is necessary to 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 stabilized level [10]. In our implementation, mutation operator works as the most important operator for the evolutionary learning of Boolean query.

Each node from the new offsprings may be mutated; that depends on mutation value (0.2). And we work with different type of mutations 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 (and, Or, Xor, and of) are allowed.
- Mutation on term node or leaf node: changing one term selected randomly from the offspring by any another one but the other one will be one from:
  - 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 query.
- Mutation by inserting or deleting operator between two nodes in the offsprings

Where mutation was implemented on this way: For given offspring select one node randomly and for this node we have two possibilities to mutate into another one or to apply insert a unary operator before it or delete it if and only of this node is a unary operator. Some examples were shown in Figure 3.

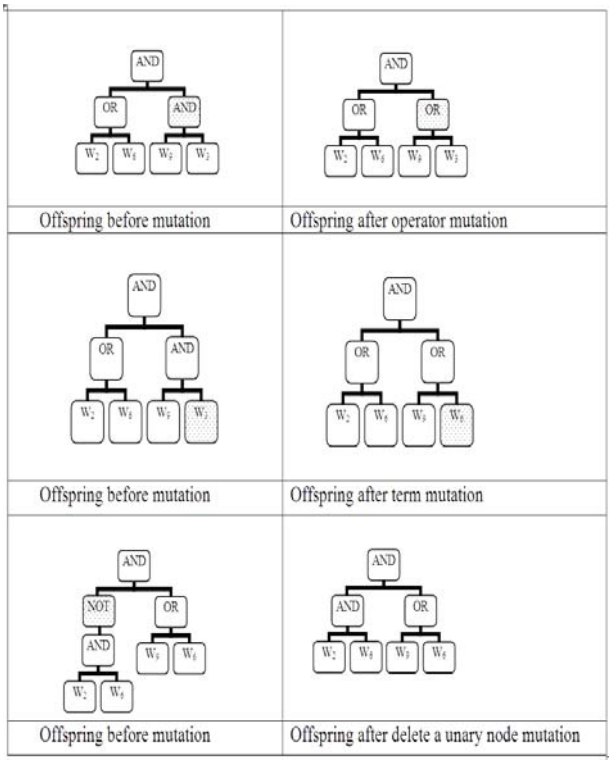

Figure 3. Single point crossover, Randomly select the nodes

6 Experiments

Presenting our work now to show how our research processed for Boolean queries evolutionary learning was done.

6.1 Introduction to experiments

We developed a genetic program to process some experiments over a set of Boolean queries and various collections of documents, and the documents are with various number of words; all collections used in our experiments are described in Table 1:

Collection Name	Number Words	Number of Documents
10x30	30	10
200x50	200	50
5000x1000	1000	5000

Table 1: Document Collections

For all of our experiments were used the following ten Boolean queries as an initial population for processing our genetic algorithm:

$2\,\mathrm{of}(w_2, w_8)$
$1\,\mathrm{of}(w_1, w_2, w_8)$
$\mathrm{not}(\mathrm{not}\, w_{13}\, \mathrm{and}\, \mathrm{not}\, w_8)$
$(w_1\, \mathrm{and}(w_2\, \mathrm{and}\, w_8))\, \mathrm{or}\, \mathrm{not}(w_4\, \mathrm{or}\, w_2)$

$\text{not}(w_1 \text{ or } w_2) \text{ and}((w_5 \text{ or } w_4) \text{ and}(w_3 \text{ and } w_6))$
$(w_9 \text{ and } w_{14})$
$(\text{not } w_{14}) \text{ and } w_1$
$(w_2 \text{ or } w_6) \text{ or}(w_8 \text{ and } w_{13})$
$(w_3 \text{ and } w_4) \text{ or}((w_1 2 \text{ xor } w_{15}) \text{ and } w_8)$
$(w_2 \text{ or } w_8) \text{ or}(w_1 \text{ and } w_2)$

Note: The of operator has the following general form:
$N \text{ of}(w_1, w_2, w_3, \ldots, w_M)$; $M \geq N$. and it Works as; terms the document will be retrieved when it contains at least N terms from the list of M terms specified in the query. For example,

$2 \text{ of}(w_1, w_2, w_3) =$
$((w_1 \text{ and } w_2) \text{ or}(w_1 \text{ and } w_3) \text{ or}(w_2 \text{ and } w_3))$

The Genetic programalgorithm ended when a given number of generations was reached; or when all chromosomes in the population had 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 few times with the same options to see the differences in the results, because results are affected by probability used during 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

6.2 Experiments Results over Mutation

Mutation value is probability of applying mutation operator on an offspring. In this experiment we observed how the change of mutation value affects the result of genetic program 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 is:- 10x30
- used fitness measure is:- precision
- Number of generations is: - 200 generations.

All terms from the initial population were used for mutation of leaves, the results obtained as shown in table 2:

mutation value	Number of generations	final precision	final 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

Table 2: Results when Mutation over leaves and terms from all initial population

Nearly in all experiments, the values of the fitness function for precision for all chromosomes in the final population reached to be as maximum of the precision value 1.25, and the same for 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 Table 3.

mutation value	Number of generations	final precision	final 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

Table 3: Results when Mutation over leaves and and terms from user query only

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

All terms form the whole collection was used for mutation of leaves, and the results were shown in Table 4. Where in some experiments the maximum number of generations was reached and in other maximum value of precision was reached.

mutation value	Number of generations	final precision	final 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

Table 4: Results when Mutation 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 mostly the values of precision are various; and the best of them are described in tables bellow, where Table 5 shows the results when the mutation over leaves used terms from user query only, Table 6 shows the results when the mutation over leaves used terms from initial population and Table 7 shows the results when the mutation over leaves used terms from whole population:

mutation value	Number of generations	final precision	final 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	

Table 5: Results when Mutation over leaves and terms from user query

mutation value	Number of generations	final precision	final 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

Table 6: Results when Mutation over leaves and terms from initial population

mutation value	Number of generations	final precision	final 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

Table 7: Results when Mutation over leaves and terms from whole collection

In some cases, especially when we used for mutation over leaves the terms from user query only and the fitness function was precision, there were worse results than in other cases as shown in tables 4, 5, and 6. We increased the maximal number of generations to be 1200 generations and did some experiments with following options. The results for these experiments are shown in Table 8.

- maximal number of generations is 1200
- user query is $((\text{not } w_{10}) \text{ and} (w_6 \text{ and } w_8))$
- mutation over leaves use terms from user query
- fitness function is precision

mutation value	Number of generations	final precision	final 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

Table 8: Results when Mutation over leaves and terms from user query

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

6.3 Fitness Function Experiments

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

Experiment options:

- collection name is 200x50
- user query is $((\text{not } w_{10}) \text{ and} (w_6 \text{ and } w_8))$
- maximum number of generations is 2000 generations
- all terms from initial population used for mutation over leaves.

When using precision as fitness function we reach the highest number of generations without reaching the best value of precision as shown in Table 9, and Table 10 shows the results when we used the recall as a fitness function.

mutation value	Number of generations	final precision	final 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

Table 9: Results when Precision was used as a fitness function

mutation value	Number of generations	final precision	final 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

Table 10: Results when Recall was used as a fitness function

* - in these cases the precision value of chromosomes in final generation was various. Number in table is lowest precision value in population.

7 Conclusions

In this paper, an optimization of 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 leaves.

After set of experiments we obtained the following conclusions. First, when applying mutation operator on terms in a query, it is necessary to have largest possible set of terms at disposal for mutation. If only terms from user query or initial population were used for mutation, the results were worse than when terms from whole collection were used. Only then there can come into existence new queries, describing the same documents as user query, but containing terms not included into user query or 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 final population. The query with fewer operators is better than query with more operators and the same values of precision and recall. This parameter can be important during whole genetic algorithms process.

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

Fourth, recall seems to be more efficient than precision. Recall as a fitness function returns quickly expressions describing all documents relevant to 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 number of generations needed to get these results is much bigger.

Acknowledgement

This work was supported by grant No 1ET100300419 awarded within Czech Republic government project Information Society.

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) 693705.

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

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

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

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

[8] Kim, Won: On Optimizing an SQL-like Nested Query. ACM Transactions on Database Systems 7, 3 (September 1982), pp. 443 - 469.

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

[10] Kraft, D.H., Bordogna, G., and 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, vol. 3, The Handbook of Fuzzy Sets Series, Norwell, MA: Kluwer Academic Publishers, 1999.

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

[12] Neruda R.: Genetic Algorithms and Neural Networks: Making Use of Parameter Space Symetries. In: IJCNN 2000. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. - Los Alamitos, IEEE Computer Society 2000

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

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

[15] Suhail S. J. Owais.: Timetabling of Lectures in the Information Technology College at Al al-Bayt University Using Genetic Algorithms. Master thesis, Al al-Bayt University 2003, Jordan. (in Arabic).

[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) 423 431.

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