Evolutionary Learning of Boolean Queries by Genetic Programming

Suhail S. J. Owais, Pavel Krömer, and Václav Snášel

Department of Computer Science, VŠB-Technical University of Ostrava,
17. listopadu 15, Ostrava - Poruba, Czech Republic
{vaclav.snasel}@vsb.cz

Abstract. 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 programming 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 Programming in Information retrieval specially in optimizing of a Boolean query.

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 consists of a list of words or terms (word and term have the same meaning in our work) 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 become a popular area of research for multiobjective problems. The use of evolutionary algorithms to solve problems with multiple objectives (known as Multi-objective Optimization Problems) has attracted much attention [6, 3, 18].

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"; 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 is the percentage of the retrieved documents that are relevant to the user query and recall is the percentage of the relevant documents that are retrieved for the requested query [1, 12, 8].

$$Recall = \frac{RelevantRetrieved}{Relevant} \qquad Precision = \frac{RelevantRetrieved}{Retrieved}$$

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 programming.

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 consists of set of terms and set of logical operators; especially **and**, **or**, **of**, and **not** operator see [8]. For this our motivation in our work is to do the evolution of the Boolean queries using genetic programming in the information retrieval [3, 2, 15].

Genetic Algorithm is an algorithm that used to find approximate solutions to problems that were 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, 16].

4 Genetic Programming

This section will present the implementation of information retrieval using genetic programming (for SQL we can see [17, 11, 7, 4]). The GA is generally used to solve optimization problems [12, 9, 5]. 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 is called allele. In information retrieval, query for relevant documents are representing for each individual or chromosome, and each document described by set of terms. For

the collection of documents D, the description for document D_i from l documents, 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 were mentioned 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**, **of** and **not**. The query set Q is 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].

Note: The **of** operator has the following general form:

$$N \,\text{of}(w_1, w_2, w_3, \dots, w_M);\ M \geq N$$

and works like this: the document will be retrieved when it contains at least N terms from the list of M terms. For an example,

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

In this work, we develop genetic programming for implementing GA operators 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.

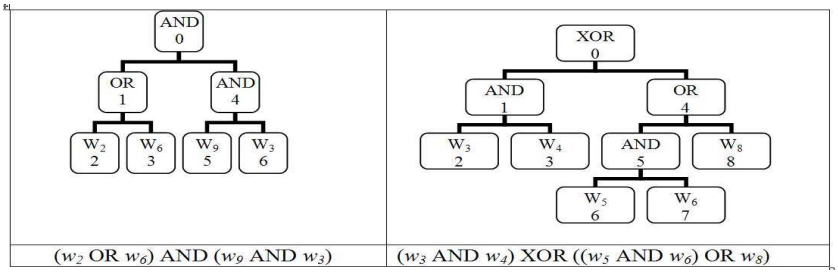

Fig. 1. Chromosome encoding form a query

5 Implement Genetic Operators to Evolve 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 $RecallFitness$ and $PrecisionFitness$ 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. Where α and β are added specially to precision fitness function [10].

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

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

Selection operator

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 parents and they will be used to produce two new offsprings.

Crossover operator

Offsprings must have some inheritance from the two 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.

Producing two new offsprings from implementation of a single point crossover was shown as an example in Figure 2.

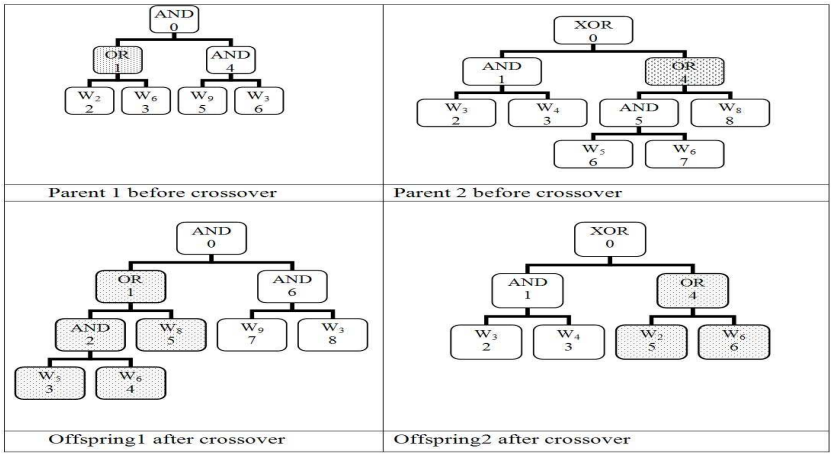

Fig. 2. Single point crossover, Randomly select nodes on parents

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 (by default 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 arity, such as any exchange in (**and**, **or**, **of** and **not**) 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 unary operator between two nodes in the offspring.

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

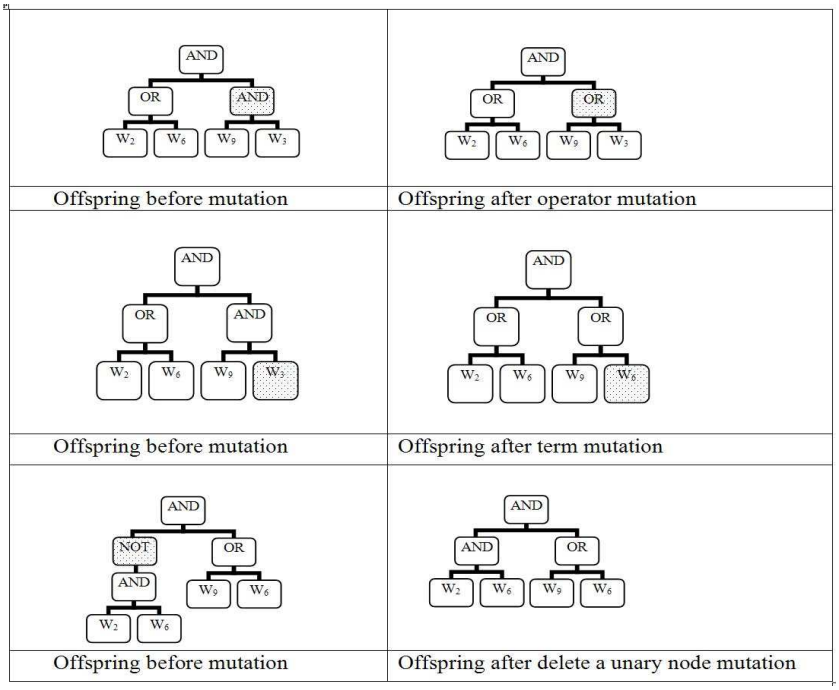

Fig. 3. Processing mutation over offsprings, where nodes are selected randomly

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 programming to process some experiments over a set of Boolean queries and various collections of documents; the documents are with various number of words or terms. All collections used in our experiments are described in Table 1, where collection 'Collection-1' consists of 10 different documents and 30 different words, where each document includes some of these words (one, two or more of them).

Table 1. Description of Document Collections

Collection Name	**Number of Documents**	**Maximum number of terms in each document**
Collection-1	10	30
Collection-2	50	200
Collection-3	5000	1000

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

$2\,\text{of}(w_2, w_8)$
$1\,\text{of}(w_1, w_2, w_8)$
$\text{not}(\text{not}\, w_{13}\, \text{and}\, \text{not}\, w_8)$
$(w_1\, \text{and}(w_2\, \text{and}\, w_8))\, \text{or}\, \text{not}(w_4\, \text{or}\, w_2)$
$\text{not}(w_1\, \text{or}\, w_2)\, \text{and}((w_5\, \text{or}\, w_4)\, \text{and}(w_3\, \text{and}\, w_6))$
$(w_9\, \text{and}\, w_{14})$
$(\text{not}\, w_{14})\, \text{and}\, w_1$
$(w_2\, \text{or}\, w_6)\, \text{or}(w_8\, \text{and}\, w_{13})$
$(w_3\, \text{and}\, w_4)\, \text{or}((w_1\, 2\, \text{xor}\, w_{15})\, \text{and}\, w_8)$
$(w_2\, \text{or}\, w_8)\, \text{or}(w_1\, \text{and}\, w_2)$

The Genetic programming execution was terminated when one or more chromosomes from the population reached the highest value of the selected fitness function, or when reaching a maximum number of generations, where the highest values for precision and recall are $\alpha + \beta$ and 1 respectively.

All experiments were done multiple times with the same options to see the differences in the results, because of probability used during genetic programming process. In all experiments were used following fixed options:

- the arbitrary weights for $\alpha = 0.25$, and $\beta = 1.0$
- crossover value = 0.8

6.2 Experiments on different types of mutation and different fitness functions

Mutation value is probability of applying mutation operator on offsprings. In these experiments we observed how the changes of mutation value affect the result of genetic programming process, where we used different types of mutation as described above and two different sets of options where implemented on our experiments.

First set of options are:

- User query is:- $(w_6\, \text{and}\, w_8)\, \text{and}\, \text{not}\, w_{10}$
- Collection name is:- Collection-1
- Used fitness measures are:- precision or recall
- Highest number of generations are: - 200 generations.

Experiments on using precision fitness function:

All terms from the initial population were used for mutation on leaves, the results were shown in table 2. Where in all experiments the chromosomes fitness values in the final population reached to be highest value, and the same for recall fitness value is the highest too, where the number of generations was variant.

Table 2. Precision, mutation over leaves using terms from all initial population

mutation value	number of generations	final precision	final recall
0.1	200	0.75	1.00
0.2	24	1.25	1.00
0.3	27	1.25	1.00
0.4	118	1.25	1.00
0.5	45	1.25	1.00

All terms from the user query were used for mutation on leaves, and the results were shown in Table 3. In this case, mostly maximum number of generations were reached to obtain the optimized query, because of not reaching the highest value on the selected fitness function, when the highest values for recall were reached.

Table 3. Precision, mutation over leaves using terms from user query only

mutation value	number of generations	final precision	final recall
0.1	200	0.75	1.00
0.2	200	0.75	1.00
0.3	200	0.75	1.00
0.4	200	0.75	1.00
0.5	113	1.25	1.00

All terms form the whole collection were used for mutation of leaves, and the results were shown in Table 4. Where in some experiments the termination of the program execution was done because of reaching the maximum number of generations or reaching the highest precision fitness value, where mostly in all experiments we reached the highest values for recall fitness values.

Table 4. Precision, mutation over leaves using terms from whole collection

mutation value	number of generations	final precision	final recall
0.1	200	0.75	1.00
0.2	58	1.25	1.00
0.3	11	1.25	1.00
0.4	187	1.25	1.00
0.5	42	1.25	1.00

Experiments on using recall fitness function:

All chromosomes in the final population had approximately the same highest value of recall, but mostly the values of precision are various, some of these results are shown in tables bellow. 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.*

Table 5. Recall, mutation over leaves using terms from user query

mutation value	number of generations	final precision	final recall
0.1	5	0.583	1.00
0.2	5	0.500	1.00
0.3	5	0.500	1.00
0.4	5	0.583	1.00
0.5	6	0.583	1.00

Table 6. Recall, mutation over leaves using terms from initial population

mutation value	number of generations	final precision	final recall
0.1	5	0.583	1.00
0.2	5	0.500	1.00
0.3	5	0.500	1.00
0.4	5	0.583	1.00
0.5	7	0.583	1.00

Table 7. Recall, mutation over leaves using terms from whole collection

mutation value	number of generations	final precision	final recall
0.1	5	0.583	1.00
0.2	5	0.500	1.00
0.3	6	0.500	1.00
0.4	8	0.583	1.00
0.5	5	0.583	1.00

From these results shown in tables 5, 6 and 7 the executions of program were terminated when the highest recall fitness function values of chromosomes were reached within few number of generations.

Some experiments were done using the second set of options with the following results shown in Table 8 and 9.

Second set of options are:

- User query is:- $((\text{not}\, w_{10})\, \text{and} (w_6\, \text{and}\, w_8))$
- Collection name is:- Collection-2
- Used fitness measures are:- precision or recall
- Highest number of generations are: - 1200 generations.

Table 8. Precision, mutation over leaves using terms from user query, Collection-2

mutation value	number of generations	final precision	final recall
0.1	1200	1.00	1.00
0.2	1200	0.75	1.00
0.3	197	1.25	1.00
0.4	462	1.25	1.00
0.5	1200	0.75	1.00

Table 9. Recall, mutation over leaves using terms from user query, Collection-2

mutation value	number of generations	final precision	final recall
0.1	16	0.3050	1.00
0.2	13	0.3050	1.00
0.3	23	0.3050	1.00
0.4	11	0.3050	1.00
0.5	17	0.3050	1.00

After increasing number of generations and experiments were done on Collection-2, there were a differences in the results because in some cases we reached the best solution depends on precision fitness function as shown in table 8 before, where the results in table 9 shown before mostly all experiments were reached the highest value of recall within a few number of generations.

Experiments over fitness functions:

The goal of optimization process of a Boolean query is to get a query with highest possible values of precision or recall depends on chosen fitness function. Results shown above demonstrate, that when we used recall as the fitness function, the program terminated within few number of generations because of reaching the highest value of recall as a fitness function, and when using precision as the fitness function recall has reached the highest value mostly in all experiments, where some of precision values were not high.

7 Conclusions

In this paper, an optimization of Boolean query over a collection of documents is presented. We focused especially on different types of mutation and on comparison of two different fitness measures, precision and recall. Experiments were done over various types of document collections and different types of mutation and two types of fitness functions. So we obtained the following conclusions:

First, when applying mutation operator on terms in the chromosomes from the initial population, it is necessary to have all the terms from the search space 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, recall seems to be more efficient than precision when chosen as a fitness function to reach an optimized query within less number of generations than when precision was chosen as a fitness function. So we retrieved all relevant documents with few number of non-relevant documents. But on choosing precision

as a fitness function, we reached mostly the highest values of recall before program termination when the highest values of precision were mostly not reached and the process terminated after reaching maximum number of generations.

Third, in some cases, especially when we used for mutation over leaves the terms from collection or from initial population on two different types of fitness functions an optimized query was reached within few number of generations, but on chosen recall as fitness function the results were reached within less number of generations than when precision was used as a fitness function. But for mutation over leaves the terms from user query only and the fitness function was precision there were worse results than in other cases.

We will focus in our *future work* on weighted terms and weighted Boolean operators for implementing the fuzzy logic over terms and Boolean operators weights for optimizing user query in information retrieval systems, and also on using different methods for evaluating the performance of information retrieval such as Harmonic mean measure (F-score). We also want to consider the number of Boolean operators and the number of terms as the objectives for query optimization.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, New York (1999)
2. 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) 693–705
3. 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, Springer-Verlag Berlin Heidelberg (2002) 710–719
4. Freytag, J. C.: A Rule-Based View of Query Optimization. Proceedings of ACM-SIGMOD (1987) 173–180
5. Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning. Reading, Massachusetts: Addison-Wesley (1989)
6. Abbass, H. A., Sarker, R., Newton, C.: PDE:A Pareto-frontier Differential Evolution Approach for Multi-objective Optimization Problems, Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001), Vol. 2, IEEE Service Center, Piscataway, New Jersey (2001) 971–978
7. Kim, W.: On Optimizing an SQL-like Nested Query. ACM Transactions on Database Systems 7 (1982) 443–469
8. Korfhage, R. R.: Information Storage and Retrieval. John Wiley & Sons, Inc. (1997)
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., 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 (1990) 38–49

12. Melanie, M.: An Introduction to Genetic Algorithms. A Bradford Book The MIT Press (1999)
13. Rijsbergen, C.J.: Information Retrieval (2nd edition), Butterworth (1979)
14. Salton, G., Buckley, C.: Terms-Weighting approach in automatic text retrieval. Information Processing and management (1988) 24(5):513–523
15. 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
16. Owais, S. S. J.: Timetabling of Lectures in the Information Technology College at Al al-Bayt University Using Genetic Algorithms. Master thesis, Al al-Bayt University, Jordan (2003) (in Arabic).
17. Yao, S. B.: Optimization of Query Algorithms. ACM Transactions on Database Systems 4 (1979) 133–155
18. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Swiss Federal Institute of Technology Zurich, Zurich (1999)