Evolutionary Improvement of Search Queries and Its Parameters

Pavel Krömer, Václav Snášel, Jan Platoš
Department of Computer Science,
VŠB - Technical University of Ostrava
17. listopadu 15, 708 33
Ostrava-Poruba, Czech Republic
Email:{pavel.kromer, vaclav.snasel, jan.platos}@vsb.cz

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs)
Scientific Network for Innovation and Research Excellence,
WA, USA
Email: ajith.abraham@ieee.org

***Abstract*—The formulation of user queries is an important part of the information retrieval process. In the complex environment of the World Wide Web and other large data collections, it is often not easy for the users to express their information needs in an optimal way. In this paper, we investigate evolutionary algorithms (in particular genetic programming) as a tool for the optimization of user queries and seek for its good settings.**

***Keywords*-Evolutionary algorithms, genetic programming, query optimization.**

I. Introduction

In this paper, we describe the usage of a powerful evolutionary optimization method, the genetic programming, for optimization of search queries. We investigate Web search as a fuzzy information retrieval task and create a test data set to evaluate and tune the algorithm. In a series of computational experiments, we seek for good parameters of the optimization algorithm that would provide above average results in terms of used fitness function.

The goal of evolutionary query optimization is to improve user queries on the basis of a user profile covering the information needs of a particular user. We focus on the implementation of genetic programming and its settings and emulate the user profile by a "model query" – a query that marks a fuzzy set of relevant documents in the collection.

Evolutionary algorithms, as well as their particular implementations, have a number of high- and low-level modifications and variants. They have been developed and tuned for different application areas; however, no ultimate general setup of evolutionary optimization has been found. No variant or particular setup of the evolutionary computation seems to be a silver bullet for optimization tasks. The situation has been paraphrased by the "no free lunch" theorem which states [16] that for any algorithm, any increased performance over one class of problems is paid for in performance over another class. The no free lunch theorem advocates the experimental search for good settings of deployed EA for different application domains.

II. Information retrieval

The area of information retrieval (IR) is a branch of computer science dealing with storage, maintenance and search in large amounts of data. The data could be in different formats, e.g. textual, visual, audio or multimedia documents [1].

An information retrieval system (IRS) is a software tool serving for data representation, storage and subsequent information search. The amount of documents contained in data collections managed by IRS is usually very large and the task of easy, efficient and accurate information search is specially challenging.

An IR model is a formal background defining internal document representation, query language and document–query matching mechanism. Consequently, the model determines document indexing procedure, result ordering and other aspects of particular information retrieval system. In this study, we have implemented extended Boolean IR model.

A. Extended Boolean IR model

Extended Boolean model of IR is based on fuzzy set theory and fuzzy logic. Documents are interpreted as fuzzy sets of indexed terms, assigning to every term contained in the document particular weight from the range $[0, 1]$ expressing the degree of significance of the term for document representation. Hence documents are modeled more accurate than in classic Boolean IR model. Formal description of a document $\boldsymbol{d}_i$ and full collection $\boldsymbol{D}$ in an extended Boolean IR model is shown in (1) and (2) respectively.

$$\boldsymbol{d}_i = (t_{i1}, t_{i2}, \ldots, t_{im}),\ \forall\ t_{ij} \in [0, 1] \tag{1}$$

$$\boldsymbol{D} = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{pmatrix} \tag{2}$$

Another feature of extended Boolean IR model is fuzzy extension of query language aiming at providing apparatus to express more flexible and accurate search requests. Two techniques are being used for query enhancement query term weighting using numeric weights or linguistic variables and Boolean conjunction parameterization for expressing relationships among the extremes of AND, OR, NOT etc. [2].

Choosing appropriate indexing procedure is essential for exploitation of extended Boolean IR model benefits. Internal

documentary collection model should be as accurate as possible snapshot of the collection of textual documents in natural language and at the same time a basis for efficient and practical search. Fuzzy indexing function is defined as shown in (3), where D stands for the set of all documents and T for set of all indexed terms.

$$F : D \times T \rightarrow [0, 1] \tag{3}$$

Donald Kraft proposed in [2] the usage of Gerard Saltons $tf \times idf_t$ indexing formula introduced for vector space IR model as document indexing mechanism in extended Boolean IR model.

Query language is in extended Boolean IR model enhanced with the possibility of weighting query terms in order to express different importance of those in search request and by weighting (parameterizing) aggregation operators to soften or blur their impact on query evaluation [1], [2]. Consider Q to be the set of user queries over a collection then the weight of term t in query q is denoted as $a(q, t)$ satisfying $a : Q \times T \rightarrow [0, 1]$. To evaluate atomic query of one term, stating therefore only one search criterion, will be used function $g : [0, 1] \times [0, 1] \rightarrow [0, 1]$. The value of $g(F(d, t), a)$ is called retrieval status value (RSV). For RSV enumeration is crucial the interpretation of query term weight a. The most used interpretations are to see query term weight as importance weight, threshold or ideal document description [1], [2].

In this study, we adopt the threshold interpretation defined in (4) and illustrated in Fig. 1. The functions $P(a)$ and $Q(a)$ are coefficients used for tuning the threshold curve. An example of $P(a)$ and $Q(a)$ could be as follows: $P(a) = \frac{1+a}{2}$ and $Q(a) = \frac{1-a^2}{4}$. Acording to the threshold interpretation, an atomic query containing term t of the weight a is a request to retrieve documents having $F(d, t)$ equal or greater to a. For documents satisfying this condition will be rated with high RSV and contrariwise documents having $F(d, t)$ smaller than a will be rated with small RSV.

$$g(F(d,t),a) = \begin{cases} P(a)\frac{F(d,t)}{a} & \text{for } F(d,t) < a \\ P(a) + Q(a)\frac{F(d,t)-a}{1-a} & \text{for } F(d,t) \geq a \end{cases} \tag{4}$$

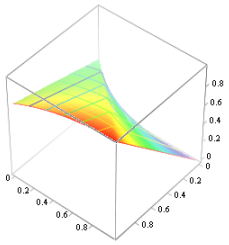


Fig. 1: $g(F(d, t), a)$ according to (4).

B. IR effectiveness evaluation

Precision P and recall R are among the most used IR effectiveness measures. They are defined in (5), where REL stands for the set of all relevant documents and RET for the set of all retrieved documents. Precision can be then understood as the probability of retrieved document to be relevant and recall can be seen as the probability of retrieving relevant document.

$$P = \frac{|REL \bigcap RET|}{|RET|} \qquad R = \frac{|REL \bigcap RET|}{|REL|} \tag{5}$$

Precision and recall in the extended Boolean IR model can be defined using sigma count $\|A\|$ [3]:

$$\rho(X|Y) = \begin{cases} \frac{\|X \cap Y\|}{\|Y\|} & \|Y\| \neq 0 \\ 1 & \|Y\| = 0 \end{cases} \tag{6}$$

$$P = \rho(REL|RET) \qquad R = \rho(RET|REL) \tag{7}$$

For easier IR effectiveness evaluation were developed measures combining precision and recall into one scalar value. F-score F [4] is among the most used scalar combinations of P and R.

$$F = \frac{2PR}{(P + R)} \tag{8}$$

III. Evolutionary algorithms

Evolutionary computation or evolutionary algorithms is a common label for a group of iterative stochastic search and optimization methods based on programmatical emulation of successful optimization strategies observed in nature [5], [6].

A. Genetic algorithms

Genetic algorithms are popular variant of evolutionary algorithms. They are based on programmatical implementation of genetic evolution and they emphasize selection and crossover as the most important operations in the whole evolutionary optimization process [7], [6].

Genetic algorithms evolve a population of chromosomes representing potential problem solutions encoded into suitable data structures. The evolution is implemented by iterative application of genetic operators modifying the chromosomes, i.e. the encoded form of problem solutions. Proper encoding is vital for the evolutionary search effectiveness. It defines the genotype, the space of all encoded problem solutions, which is different from phenotype, the space of all problem solutions. Genetic algorithms explore the genotype of investigated problem and the size and shape of problem genotype define its fitness landscape.

B. Genetic programming

Problem representation and chromosome structure is one of the key characteristics of genetic algorithms. Linear fixed length chromosomes consisting of 0s and 1s (or other discrete alphabets) are not very suitable for solving problems in artificial intelligence or for search queries optimization. This kind of problems requires rather hierarchical structures with unequal and unknown lengths [7], [8]. Genetic programming is an extension to GA allowing work with hierarchical, often tree-like, chromosomes with unlimited length [7], [8].

1 Define objective (fitness) function and problem encoding
2 Encode initial population P of possible solutions as fixed length strings
3 Evaluate chromosomes in initial population using objective function
4 **while** *Termination criteria not satisfied* **do**
5 Apply selection operator to select parent chromosomes for reproduction: $sel(P_i) \rightarrow parent_1$, $sel(P_i) \rightarrow parent_2$
6 Apply crossover operator on parents with respect to crossover probability P_C to produce new chromosomes: $cross(P_C, parent_1, parent_2) \rightarrow \{offspring_1, offspring_2\}$
7 Apply mutation operator on offspring chromosomes with respect to mutation probability P_M: $mut(P_M, offspring_1) \rightarrow offspring_1$, $mut(P_M, offspring_2) \rightarrow offspring_2$
8 Evaluate offspring chromosomes: $fit(offspring_1) \rightarrow offspring_1^{fit}$, $fit(offspring_2) \rightarrow offspring_2^{fit}$
9 Create new population from current population and offspring chromosomes: $migrate(offspring_1, offsprig_2, P_i) \rightarrow P_{i+1}$
10 **end**

Algorithm 1: A summary of genetic algorithm

Genetic programming has been introduced as a tool to evolve whole computer programs and represented a step towards adaptable computers that could solve problems without being programmed explicitely [9]. Moreover, genetic programming can be used to develop solutions in the field of machine learning, symbolic processing or any other domain that can formulate its solutions by the means of parseable symbolic expression. Genetic programming allows efficient evolution of such symbolic expressions with well defined syntax and grammar.

The chromosomes take in GP form of hierarchical variable size expressions, point labeled structure trees. The trees are constructed from nodes of two types, terminals and functions. More formally, a GP chromosome is a symbolic expression created from terminals t from the set of all terminals T and functions f from the set of all functions F satisfying the recursive definition [9]:

1) $\forall t \in T$: t is correct expression
2) $\forall f \in F : f(e_1, e_2, \ldots, e_n)$ is correct expression if $f \in F$ and $e+1, \ldots, e_n$ are correct expressions. The function $arity(f)$ represents the arity of f.
3) there are no other correct expressions

GP chromosomes are evaluated by recursive execution of instructions corresponding to tree nodes [9]. Terminal nodes are evaluated directly (e.g. by reading an input variable) and functions are evaluated after left-to-right dept-first evaluation of their paramaters.

Genetic operators are applied on the nodes in the tree-shaped chromosomes. Crossover operator is implemented as mutual exhchange of randomly selected subtrees of the parent chromosomes. For an example see Fig. 2. Mutation has to modify the chromosomes by pseudorandom arbitrary changes in order to prevent premature convergence and broaden the coverage of fitness landscape.

Genetic programming enables efficient evolution of symbolic expressions, even whole computer programs. However,

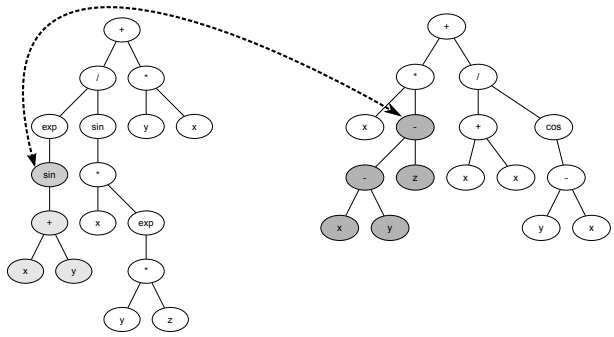


Fig. 2: Crossover in genetic programming.

the original design and the closure property are limiting factors to the algorithm.

IV. Evolutionary approaches to search optimization

Evolutionary optimizations were applied to document indexing, query representation and optimization, matching and ranking function optimization and user profile improvements.

The optimization of IR ranking function was described e.g. in [10], [11]. The studies describe in detail the importance of suitable ranking function and propose an automated GP based ranking function improvement mechanism for consensual and personalized IR systems. The optimization exploits document content and structure and it is demonstrated on vector space model of IRS.

There were several works dealing with query optimization by GA or GP. Donald Kraft et al. [2] used genetic programming to optimize Boolean search queries over documentary database with emphasis on comparison of several IR effectiveness measures as objective functions. Cordón et al. [12] introduced MOGA-P, an algorithm to deal with search query optimization as with a multi-objective optimization problem.

Gordon et al. [13] employed genetic programming to automatically evolve optimal term weighting function for retrieval algorithms based on a users evaluation of previously viewed documents.

Nyongesa and Maleki-dizaji [14] proposed reinforced interactive evolutionary learning for user modeling over a vector space mode of information retrieval. Cummins and O'Riordan [15] applied genetic programming for automatic query expansion technique adding terms to users initial query and observed improvement in mean average precision in several cases.

In this paper, we focus on the implementation and tuning of a GP for the optimization of the extended Boolean queries. The agorithm facilitates efficient evolution of flexible search queries in context of the powerfull extended Boolean IR model.

V. Experimental evolutionary query optimization

We have created an information retrieval system to evaluate genetic programming for search query optimization. The information retrieval system implements extended Boolean information retrieval model as described in Section II-A. The

$tf \cdot idf_t$ term statistics was used for document indexing and query weights (RSV) were evaluated using (4). Query language in implemented IRS supported standard Boolean operators AND, OR and NOT.

The information retrieval system served as a test bed for evolutionary query optimization. We have implemented genetic programming over extended Boolean queries. The GP evolved tree representations of search queries with Boolean operators as function nodes and terms leaves. Both, operator nodes and term nodes, were weighted. In order to generate random initial population for GP, the system was able to generate random queries. Particular settings of random query generator showing the probabilities of generating particular query node, are summarized in Table Ia. An example of three random queries generated by the system is shown in Fig. 3.

Crossover operator was is implemented as a mutual exchange of two randomly selected branches of parent tree chromosomes. Mutation operator in query GP aims to perturbe content and structure of chromosomes randomly. In our implementation, it selects a node from the processed chromosome at random and performs one of the mutation operations summarized in Table Ib.

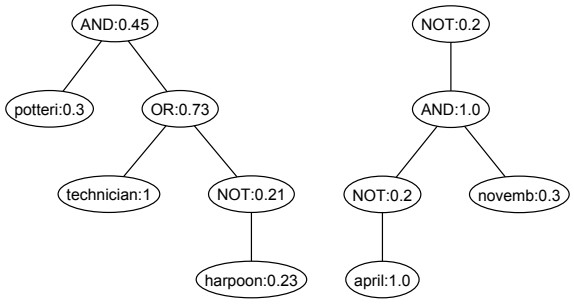
Event	Probability
Generate term	0.5
Generate AND	0.24
Generate OR	0.24
Generate NOT	0.02

(a) Probabilities of generating random query nodes.

Event	Probability
Mutate node weight	0.5
Insert or delete NOT node	0.1
Replace with another node or delete NOT node	0.32
Replace with random branch	0.08

(b) Probabilities of mutation operations.

TABLE I: Random query generation an mutation probabilities.



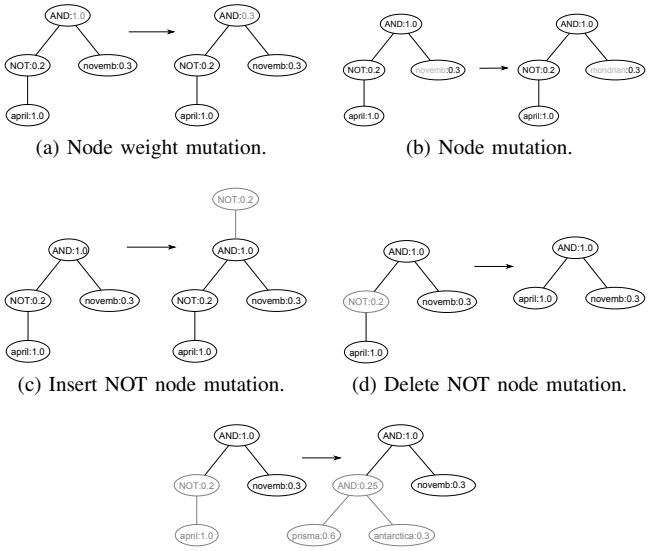
(a) Query potteri:0.3 AND:0.45 (technician OR:0.73 NOT:0.21 harpoon:0.23).

(b) Query NOT:0.2 ((NOT:0.2 april) AND novemb:0.3).

Fig. 3: Sample random queries.

Implemented query mutation types included:

- change of selected node weight. This mutation type is shown in Fig.4a
- replacement of selected node type with a compatible node type (i.e. operator OR replace by operator AND, term replaced by another term). This mutation type is shown in Fig. 4b.
- insertion of NOT operator before selected node (Fig. 4c).
- removal of NOT operator if selected (Fig. 4d).
- replacement of selected node with randomly generated branch (Fig. 4d).



(a) Node weight mutation.

(b) Node mutation.

(c) Insert NOT node mutation.

(d) Delete NOT node mutation.

(e) Replace node by branch mutation.

Fig. 4: Query mutation types.

F-Score has been used as fitness function in all experiments.

A. Experimental data set

A dump of one language mutation of the popular Web encyclopedia Wikipedia[1] was used as data collection for experiments. Simple English Wikipedia[2] is a language mutation of Wikipedia written in simple English with limited vocabulary and using only simple grammar. For our purposes, it is attractive because it contains only a relativelly small number of articles in English language. As in other language variants, articles are linked (within simple Wikipedia but also to other Web sites) and categorized.

We have selected small subset of Simplewiki collection to perform experiments in order to tune the genetic programming for evolutionary query optimization. We have used random node sampling method to select 110 documents (nodes) from Simplewiki at random with uniform probability. Selected nodes and links between them formed a small test collection called Rand110 with 5107 index terms.

VI. Evolutionary query optimization

This section presents computational experiments performed in order to verify the ability of genetic programming to evolve extended Boolean queries over a collection of relevance ranked documents. The goal of the algorithm is to evolve queries describing the fuzzy set of relevant documents marked by model query.

[1] http://wikipedia.org/
[2] http://simple.wikipedia.org/

The experiments with genetic programming are influenced by its stochastic nature. In order to obtain representative results, we have repeated every experiment in multiple independent runs and we present average values of obtained results.

A. Parameter tuning

A series of experiments to find suitable settings for query evolution GP has been conducted. Right settings can significantly affect performance and results of any evolutionary algorithm. The no free lunch theorem [16] states that there is no particular evolutionary algorithm performing optimally for all problems. In a similar manner, we can say that there is no general setting for any evolutionary algorithm, though there are some best practices for setting P_C, P_M, population size and so on. Nevertheless, it has been shown many times that the GP with commonly used parameters can be for some problems outperformed by GP with domain specific settings.

1) Crossover and mutation probability: We have sought for the best setting of mutation probability P_M and crossover probability P_C. To mark documents in Rand110 relevant and non-relevant, the model query "month or year or day" has been used. The use of model query guarantees that there is a search expression describing the fuzzy set of relevant documents perfectly and reaches F-Score 1. Indeed, this is a certainty we do not have in a general case. The model query assigned to each document in Rand110 a relevance score as illustrated in Fig. 5. Algorithm parameters for this experiment are summarized in Table II.

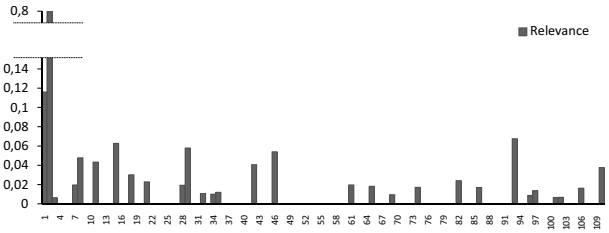


Fig. 5: Relevance used for P_C and P_M tuning in Rand110.

We have performed query evolution for every combination of P_C and P_M from the interval $[0.1, 1]$ with the step 0.1 and recorded the fitness values (best, average and worst) in final populations. The results of query evolution with different settings of P_C and P_M are visualized as contour plots in Fig. 6.

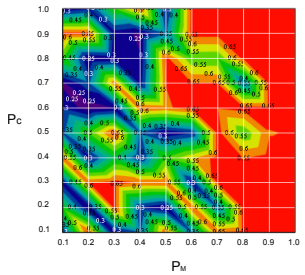
The best fitness values (larger than 0.67) have been achieved in optimization runs with large P_M. From the results can be seen that experimental runs with large P_M and large P_C at the same time achieved also good average final fitness and worst final fitness comparing to other test cases.

This is an interesting observation since the usual settings for genetic algorithms suggest large P_C but rather smaller P_M. The reason for the need for large P_M in search query evolution could be the size and complexity of query chromosomes. Rand110 contains more than 5000 terms, but the random initial population of 100 queries contains only a very limited sample of randomly generated term nodes. Because of this fact and thanks to chosen mutation implementation (during mutation, only one node is selected and transformed), the need for larger P_M arose. Based on the experience from this experiment, we are using probability of crossover $P_C = 0.8$ and probability of mutation $P_M = 0.8$ for the rest of experiments presented in this thesis.

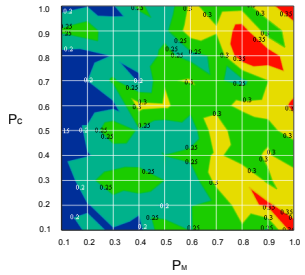
Despite the fact that a relevant query was used to create the fuzzy set of relevant documents, GP did not find a search expression with fitness close to 1 in the first 400 generations. It suggests that the number of generations should be larger than 400 and it also indicates that the GP might require an initial information to reach better fitness. After all, the space of all possible extended Boolean queries (even over Rand110 vocabulary only) is frankly immense.

Parameter	Value
Model query	"month or year or day"
Population size	100
Selection scheme	Elitary
Generations limit	400
Fitness	F-Score
Independent runs	10

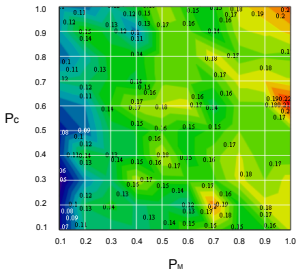
TABLE II: Parameters used for P_C and P_M tuning.



(a) Best fitness values for different P_C and P_M.



(b) Average fitness values for different P_C and P_M.



(c) Worst fitness values for different P_C and P_M.

Fig. 6: Mutation and crossover probability tuning results in contour plot.

2) Selection method: Suitable selection strategy for the query optimization task has been investigated. We have considered roulette wheel selection and elitary selection. In our genetic programming implementation, roulette wheel selection picks a chromosome with probability corresponding to chromosomes relative fitness. It means that a chromosome with higher fitness has better chance to become parent. Elitary selection was implemented so that first two best chromosomes were selected as parents. If there were either more than two

best or second best chromosomes, the parents were selected among them randomly with uniform probability.

The parameters of GP for the experiment with selection types are shown in Table III and the values of maximum, average and worst fitness in an average final population are summarized in Table IV.

Parameter	Value
Population size	100
Generations limit	1000
Fitness	F-Score
Mutation probability	0.8
Crossover probability	0.8
Repeat	10

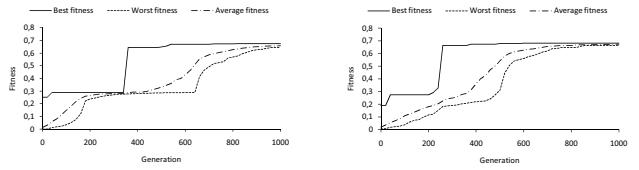
TABLE III: Algorithm parameters used for selection method comparison.

	Fitness in final population		
Selection	Best	Average	Worst
Elitary	0.388	0.303	0.300
Roulette wheel	0.388	0.372	0.365

TABLE IV: Comparison of query evolution results with different selections algorithm.

Table IV suggests that best fitness in an average final population is the same for both selection methods. However, the use of roulette wheel selection leads to better average fitness and better worst fitness in an average final population.

Examples of query evolution are shown in Fig. 7. There are no significant differencies between best runs with roulette wheel selection and elitary selection. We can note that the average fitness is with roulette wheel selection (Fig. 7b) growing more slowly but in a smoother manner. Average fitness and worst fitness in final population were better when using roulette wheel selection.



(a) Best run with elitary selection.

(b) Best run with roulette wheel selection.

Fig. 7: Exmples of query evolution.

VII. Conclusions

We have studied genetic programming for evolutionary query optimization. A test data set for the research of information retrieval in hyperlinked environments has been created. As the basis for testing data, one language variant of the Wikipedia was used.

We have designed, implemented and evaluated genetic programming for evolutionary query optimization. We have sought for good parameters for the algorithm and we have shown that the algorithm is able to find search expressions describing corresponding documents in extended Boolean information retrieval model. To achieve good performance and good results, the genetic programming for evolutionary query optimization required high values of P_C and P_M. The experiments have shown that the use of roulette wheel selection leads to better average fitness in the final population.

Acknowledgement

This work was supported by the Ministry of Industry and Trade of the Czech Republic, under the grant no. FR-TI1/420.

References

[1] F. Crestani and G. Pasi, "Soft information retrieval: Applications of fuzzy set theory and neural networks," in *Neuro-Fuzzy Techniques for Intelligent Information Systems* (N. Kasabov and R. Kozma, eds.), pp. 287–315, Heidelberg, DE: Springer Verlag, 1999.

[2] D. H. Kraft, F. E. Petry, B. P. Buckles, and T. Sadasivan, "Genetic Algorithms for Query Optimization in Information Retrieval: Relevance Feedback," in *Genetic Algorithms and Fuzzy Logic Systems* (E. Sanchez, T. Shibata, and L. Zadeh, eds.), (Singapore), World Scientific, 1997.

[3] H. L. Larsen, "Retrieval evaluation," in *Modern Information Retrieval course*, Aalborg University Esbjerg, 2004.

[4] R. M. Losee, "When information retrieval measures agree about the relative quality of document rankings," *Journal of the American Society of Information Science*, vol. 51, no. 9, pp. pp. 834–840, 2000.

[5] M. Dianati, I. Song, and M. Treiber, "An introduction to genetic algorithms and evolution strategies," technical report, University of Waterloo, Ontario, N2L 3G1, Canada, July 2002.

[6] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[7] J. Koza, "*Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*," Technical Report STAN-CS-90-1314, Dept. of Computer Science, Stanford University, 1990.

[8] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[9] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall/CRC, 2009.

[10] W. Fan, M. D. Gordon, and P. Pathak, "A generic ranking function discovery framework by genetic programming for information retrieval," *Inf. Process. Manage*, vol. 40, no. 4, pp. pp. 587–602, 2004.

[11] W. Fan, M. D. Gordon, P. Pathak, W. Xi, and E. A. Fox, "Ranking function optimization for effective web search by genetic programming: An empirical study," in *HICSS*, 2004.

[12] O. Cordón, F. de Moya, and C. Zarco, "Fuzzy logic and multiobjective evolutionary algorithms as soft computing tools for persistent query learning in text retrieval environments," in *IEEE International Conference on Fuzzy Systems 2004*, (Budapest, Hungary), pp. 571–576, 2004.

[13] M. Gordon, W. P. Fan, and P. Pathak, "Adaptive web search: Evolving a program that finds information," *IEEE Intelligent Systems*, vol. 21, pp. 72–77, 2006.

[14] H. O. Nyongesa and S. Maleki-Dizaji, "User modelling using evolutionary interactive reinforcement learning," *Inf. Retr.*, vol. 9, no. 3, pp. 343–355, 2006.

[15] R. Cummins and C. O'Riordan, "Using genetic programming for information retrieval: local and global query expansion," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, (New York, NY, USA), pp. 2255–2255, ACM, 2007.

[16] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, pp. 67–82, August 2002.