Generation of simple structured IR functions by genetic algorithm without stagnation

Kulunchakov A. S.[a], Strijov V. V.[b]

[a] *Moscow Institute of Physics and Technology*
[b] *Computing Centre of the Russian Academy of Sciences*

Abstract

This paper investigates an approach to construct new ranking models for Information Retrieval. The IR ranking model depends on the document description. It includes the term frequency and document frequency. The model ranks documents upon a user request. The quality of the model is defined by the difference between the documents, which experts assess as relative to the request, and the ranked ones. To boost the model quality a modified genetic algorithm was developed. It generates models as superpositions of primitive functions and selects the best according to the quality criterion. The main impact of the research if the new technique to avoid stagnation and to control structural complexity of the consequently generated models. To solve problems of stagnation and complexity, a new criterion of model selection was introduced. It uses structural metric and penalty functions, which are defined in space of generated superpositions. To show that the newly discovered models outperform the other state-of-the-art IR scoring models the authors perform a computational experiment on TREC datasets. It shows that the resulted algorithm is significantly faster than the exhaustive one. It constructs better ranking models according to the MAP criterion. The obtained models are much simpler than the models, which were constructed with alternative approaches. The proposed technique is significant for developing the information retrieval systems based on expert assessments of the query-document relevance.

Keywords: information retrieval, evolutionary stagnation, ranking function, genetic programming, overfitting

Email addresses: kulu-andrej@yandex.com (Kulunchakov A. S.), strijov@gmail.com (Strijov V. V.)

1. Introduction

Information retrieval is finding relevant documents, which satisfy an information need, from within large collections [1]. To retrieve documents relevant to a query, one needs a rank estimation procedure called *ranking model*. It is defined on pairs *document-query*. For each pair it returns relevance of the *document* to the *query*. [2] defines IR ranking models as functions of two basic features of these pairs: term frequency (tf) and document frequency (idf). In this paper ranking models are constructed considered as mathematical functions defined on tf-idf features. Instead of enlarging the set of features to provide better performance [3], current paper use the same tf-idf features to make further comparison consistent.

The ranking models in [4, 5, 6, 7, 8] are derived on some theoretical assumptions. These assumptions This allow to build ranking models without an IR collection, but these assumptions are not often met. For example, the derived ranking models are not optimal according to mean average precision [1] on TREC collections [2]. Moreover, the quality of these models significantly differs on various the collections [2].

High-performing ranking models are also discovered by automatic procedures. The paper [2] exhaustively explores a set of IR ranking models represented as superpositions of expert-given grammar elements. The grammar is an expert-given set of primitive mathematical functions, where variables are *tf-idf* features [9]. The exhaustive algorithm explores the set of superpositions, which consists of at most 8 grammar elements. The best explored ranking functions in [2] are better in average on TREC collections than ones in [4, 5, 6, 7, 8]. Moreover, these functions are guaranteed to have simple structure. However, this algorithm has high computational complexity [2]. Therefore, an exploration of more complex superpositions is an intractable problem.

Another approaches to improve IR expert systems include various genetic algorithms: search for an optimal document indexing [10, 11], clustering documents according to their relevance to queries [12, 13], tuning parameters of queries [14, 15], facilitate automatic topic selections [16], search for key words in documents [17] and optimal coefficients of a linear superposition of ranking models [18, 19]. Genetic algorithms are applied to select features in image retrieval and classification [20]. Genetic algorithms are used to generate ranking functions represented as superpositions of grammar elements [21, 22, 23]. These procedures significantly extend the set of ranking superpositions considered in [2]. However, the *basic* algorithms in [21, 22] produce superpositions with significant structural complexity after 30-40 iterations of mutations and crossovers [23]. The basic

algorithms do not control the structural complexity of generated superpositions and do not solve a problem of evolutionary stagnation, when a population stops to change.

Strengths	Weaknesses
[Fan et al. 2004, 2000], [22, 21]	
Large feasible set of ranking functions Fast convergence to a local optimum	Complicated final superpositions Does not provide global optimum in the feasible set of functions Have not been tested on different datasets to show uniform improvement on them
[Goswami 2014], [2]	
Provides global optimum with respect to the feasible set Compact final ranking functions Have been tested on different datasets and uniform improvement over existing approaches was shown	Small feasible set of ranking functions
[BM25]	
Theoretically justified Simple and compact explicit expression	Is not uniformly good over different datasets
[The proposed model generation algorithm]	
Large feasible set of ranking functions Fast convergence to a local optimum Compact final ranking functions Have been tested on different datasets to show uniform improvement on them	Does not provide global optimum in the feasible set of functions

Table 1: Comparison of CPU time required by structural metrics

The problem of evolutionary stagnation appears when a majority of stored superpositions have similar structure and high quality. Next crossover operations constructs superpositions, which are similar to the stored ones. The mutation operation constructs a superposition, which is unlikely

to have as high quality as the stored superpositions. This superposition highly probably will be eliminated. Therefore the population will pass to the next iteration without changes. The genetic algorithm stops actual generation.

To outperform the ranking functions found in [2], one needs to extend the set of superpositions considered there. To perform it, a modified genetic algorithm is proposed. First, it detects evolutionary stagnation and replaces the worst stored superpositions with random ones. This detection is implemented with a structural metric on superpositions. Regularizers solve the problem of overfitting. They penalize the excessive structural complexity of superpositions. The paper analyzes various pairs regularizer-metric and chooses the pair providing a selection of better ranking superpositions. All strengths and weakness of compared approaches are summarized in Table 1.

The paper [2] uses TREC collections to test ranking functions. To make the comparison of approaches consistent, the present paper also use these collections. The collection TREC-7 (trec.nist.gov) is used as the train dataset to evaluate quality of generated superpositions. The collections TREC-5, TREC-6, TREC-8 are used as test datasets to test selected superpositions.

2. Problem statement

There given a collection C consisting of documents $\{d_i\}_{i=1}^{|C|}$ and queries $Q = \{q_j\}_{j=1}^{|Q|}$. For each query $q \in Q$ some documents C_q from C are ranked by experts. These ranks g are binary

$$g : Q \times C_q \to \mathbb{Y} = \{0, 1\},$$

where 1 corresponds to relevant documents and 0 to irrelevant.

To approximate g, superpositions of grammar elements are generated. The grammar $\mathfrak{G}$ is a set $\{g_1, \ldots, g_m, x_w^d, y_w\}$, where each g_i stands for an mathematical function and x_w^d, y_w stand for variables. These variables are tf-idf features of *document-query* pair (d, q). Feature x_w^d is a frequency of the word $w \in q$ in d, feature y_w is a frequency of w in C:

$$x_w^d = t_d^w \log\left(1 + \frac{l_a}{l_d}\right), \quad y_w = \frac{N_w}{|C|}, \tag{1}$$

where N_w is the number of documents from C containing w, t_d^w is the frequency of w in d, l_d is the number of words in d (the size of a document d), l_a is an average size of documents in C. Each superposition f of grammar elements is stored as a directed labeled tree T_f with vertices labeled by elements from $\mathfrak{G}$. The set of these superpositions is defined as $\mathfrak{F}$.

The value of f on a pair (d, q) is defined as a sum of its values on (d, w), where w is a word from q:

$$f(d,q) = \sum_{w \in q} f(x_w^d, y_w).$$

The superposition f ranks the documents for each q. The quality of f is the mean average precision [1]

$$\mathrm{MAP}(f, C, Q) = \frac{1}{|Q|} \sum_{q=1}^{Q} \mathrm{AveP}(f, q),$$

where

$$\mathrm{AveP}(f,q) = \frac{\sum_{k=1}^{|C_q|} \big(\mathrm{Prec}(k) \times g(k)\big)}{\sum_{k=1}^{|C_q|} \mathrm{Rel}(k)}, \quad \mathrm{Prec}(k) = \frac{\sum_{s=1}^{k} g(s)}{k},$$

where $g(k) \in \{0, 1\}$ is a relevance of the k-th document from C.

This paper aims at finding the superposition f, which maximizes the following quality function

$$f^* = \operatorname*{argmax}_{f \in \mathfrak{F}} \mathcal{S}(f, C, Q), \quad \mathcal{S}(f, C, Q) = \mathrm{MAP}(f, C, Q) - \mathrm{R}(f), \tag{2}$$

where R is a regularizer controlling the structural complexity of f.

The exhaustive algorithm in [2] generates random ranking superpositions consisting at most of 8 elements of the grammar $\mathfrak{G}$. Let $\mathfrak{F}_0$ be the set of the best superpositions selected in [2]. The solution f^* is compared with the superpositions from $\mathfrak{F}_0$ with respect to to MAP.

3. Generation of superpositions

IR ranking functions are superpositions of expert-given primitive functions. These superpositions are generated by the genetic algorithm. It uses an expertly given grammar $\mathfrak{G}$ and constructs superpositions of its elements. On each iteration it keeps stores a population of the best selected superpositions. To update them and pass to the next iteration, it generates new superpositions with use of the stored ones. Since the superpositions are represented as trees, the algorithm applies crossover $c(f, h)$ and mutation $m(f)$ operations to the stored trees

$$c(f,h) : \mathfrak{F} \times \mathfrak{F} \to \mathfrak{F}, \quad m(f) : \mathfrak{F} \to \mathfrak{F},$$

Definition 1. *Crossover operation $c(f, h) : \mathfrak{F} \times \mathfrak{F} \to \mathfrak{F}$ produces a new superpositions from given f and h. This operation represents f and h as trees, uniformly selected a subtree for each of them and swaps these subtrees.*

Algorithm 1 Basic genetic algorithm

Require: grammar $\mathfrak{G}$, required value α of MAP

Ensure: superposition f of elements from G with MAP $\leq \alpha$;

create a set of initial, random superpositions $\mathfrak{M}_0$,

repeat

- crossover random pairs of stored superpositions $\mathfrak{M}$,
- mutate random superpositions from the population $\mathfrak{M}$,
- consider these generated superpositions and the ones stored in $\mathfrak{M}$. Select the best of them according to MAP,
- store the best generated superpositions in the population $\mathfrak{M}$ and pass it to the next iteration,

until the required value of MAP is reached;

Here is an example of crossover on two superpositions, where randomly selected subtrees are in bold.

$$f(x,y) = \exp(x) + \mathbf{ln(xy)},\ h(x,y) = \sqrt{x} + \mathbf{(x+y)}$$

$$\downarrow$$

$$f'(x,y) = \exp(x) + \mathbf{(x+y)},\ h'(x,y) = \sqrt{x} + \mathbf{ln(x \cdot y)},$$

The new superpositions are formed by swapping of these subtrees.

Definition 2. *Mutation $m(f)$ uniformly selects a subtree from f and replace it with another random superposition. Mutation produces one new superposition.*

Here is an example of mutation on a superposition

$$f(x,y) = \sqrt{x} + \mathbf{ln(x \cdot y)} \to f'(x,y) = \sqrt{x} + \mathbf{exp(y)}.$$

Definition 3. *Size $|T|$ of a tree T is the number of its vertices.*

Restrict the size of substituting tree. If mutation replaces a subtree T with a tree T', then bound the size of T' by $c|T|$, where c is a constant. This restriction allows us to explore the set $\mathfrak{F}$ more

gradually. The reason is to prevent the algorithm from instantaneous moving toward complicated superpositions if the stored population consists mainly of simple structured superpositions. Now the genetic algorithm is described in Algorithm 1. It will be referred as *basic genetic algorithm*.

4. Metric properties of basic genetic algorithm

To analyze the genetic algorithm, introduce a structural metric $\mu(T, T')$. It is defined on pairs of directed labeled trees. Therefore, it is defined on pairs of elements from $\mathfrak{F}$ as well.

$$\mu(f, f') = \mu(T_f, T'_f).$$

This structural metric satisfies the following conditions

1) $\mu(f, f) = 0,\ \mu(f, f') > 0$ if $f \neq f'$ (non-negativity),

2) $\mu(f, f') = \mu(f', f)$ (symmetry),

3) $\mu(f, f') \leq \mu(f, f'') + \mu(f'', f')$ (triangle inequality). enumerate

For $r > 0$ define the r-neighborhood $U_r(f)$ of superposition f as a set of superpositions in $\mathfrak{F}$ that are at distance less than r from f

$$U_r(f) = \{f' \in \mathfrak{F} :\ \mu\big(f, f'\big)\ <\ r\}.$$

To associate the structural distance between superpositions with a distance on their values, introduce an extra condition. Claim that the functions, lying in one structural neighborhood, should rank the documents mainly similarly. Define a distance function η on the ranks of IR ranking functions:

$$\eta(f, f') = \frac{1}{|C|\big(|C| - 1\big)} \sum_{d_j, d_k \in C} [f(d_j) < f(d_k)][f'(d_j) > f'(d_k)],$$

where $[A]$ is the indicator of event A. It is related with Kendall rank correlation coefficient by the equation:

$$\tau(f, f') = 1 - 2\eta(f, f').$$

The function η is the normalized number of inversions necessary to transform one list with ranks to the other. Therefore $\eta(f, f')$ is a distance on the values of the superpositions. Call the neighborhood $V_r(f) = \{f' : \eta(f, f') < r\}$ the value-neighborhood.

Introduce a condition for μ to detect evolutionary stagnation of the genetic algorithm enumerate

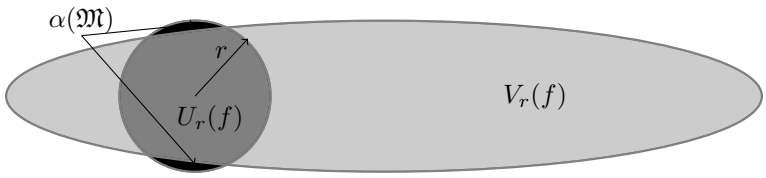

Figure 1: Illustration of supposed relation between $U_r(f)$ and $V_r(f)$.

4)

$$\alpha(\mathfrak{M}) = \nu\Big([\mu(f, f') \leq \alpha_1] \Rightarrow [\eta(f, f') \leq \alpha_2] \big| f, f' \in \mathfrak{M} \Big) \geq 1 - \varepsilon, \tag{3}$$

where $\alpha_1, \alpha_2, \varepsilon$ are some constants and $\nu\big(A\big)$ is the frequency of event A.

It claims that structurally similar functions rank documents mainly similarly. Figure 1 shows supposed relation between structural neighborhood $U_r(f)$ and value-neighborhood $V_r(f)$. Condition (3) states that the area of the black region on Figure 1 should be relatively small.

Let f_{opt} be a superposition of high quality according to $\mathcal{S}$. If μ satisfies condition (3), then the superpositions in the neighborhood $U_r\left(f_{\text{opt}}\right)$ will also have high quality. Suppose that $f_{\text{opt}} \neq f^*$ (2). It means that the optimal ranking superposition f^* is not found yet. If all superpositions of a stored population $\mathfrak{M}_i$ lye in $U_r\left(f_{\text{opt}}\right)$, then they will rarely leave $U_r\left(f_{\text{opt}}\right)$ on the next iterations, since crossovers produce superpositions mainly from $U_r\left(f_{\text{opt}}\right)$ and mutations produce superpositions mainly of lower quality. Therefore, the optimal function f^* will frequently become unreachable for the genetic algorithm, as consequence of this evolutionary stagnation.

Definition 4. *Evolutionary stagnation is a situation in a genetic algorithm, when stored superpositions are pairwise similar. The generated algorithm stops generation of principally new superpositions and the population mainly does not change from iteration to iteration.*

Definition 5. *Radius $r(\mathfrak{M})$ of a population $\mathfrak{M}$ is the minimum size of r-neighborhood with center in $f \in \mathfrak{M}$, which accommodates $\mathfrak{M}$. It shows how are the functions from $\mathfrak{M}$ scattered across the set $\mathfrak{F}$.*

$$r(\mathfrak{M}) = \underset{r>0}{\operatorname{argmin}}\{\exists f \in \mathfrak{F}\ \forall f' \in \mathfrak{M} : f' \in U_r(f)\} = \min_{f \in \mathfrak{M}} \max_{f' \in \mathfrak{M}} \{\mu f', f\}. \tag{4}$$

Detect evolutionary stagnation with structural metric μ. Lets consider a population $\mathfrak{M}$ stored by the genetic algorithm. If the genetic algorithm stagnates, then $r(\mathfrak{M})$ is relatively small. Oppositely, if the population is diverse, then the $r(\mathfrak{M})$ is big. Therefore evolutionary stagnation could be detected with the radius $r(\mathfrak{M})$. However, it is an intractable problem to find the exact value of $r(\mathfrak{M})$. Therefore, propose an empirical estimation of this radius.

Definition 6. *Structural complexity $|f|$ of superposition f is the number of grammar elements, which f consists of.*

Definition 7. *Empirical radius $r_e(\mathfrak{M})$ of is a normalized average distance between superpositions in $\mathfrak{M}$.*

$$r_e(\mathfrak{M}) = \frac{\sum\limits_{f,f' \in \mathfrak{M}} \mu_i(f, f')}{|\mathfrak{M}| \sum\limits_{f \in \mathfrak{M}} |f_j|}. \tag{5}$$

This estimation is used to detect evolutionary stagnation of the genetic algorithm. If $\mathrm{r}_e(\mathfrak{M})$ is less than a threshhold $\mathrm{r}(\mathfrak{M}) < \mathrm{Thresh}$, eliminate the worst superpositions from $\mathfrak{M}$ and replace them with random superpositions of the same structural complexity. This procedure increases the radius of $\mathfrak{M}$ and diversifies it. Therefore, the present aim of this paper is to select a proper structural metric μ, which satisfies all mentioned conditions.

5. Structural metrics

Each ranking superposition $f \in \mathfrak{F}$ is represented as directed tree T_f, which vertices are labeled by elements from grammar $\mathfrak{G}$. Structural metrics are defined on pairs of such trees. It automatically defines them on pairs of superpositions. This paper analyzes three metrics.

5.1. Similarity according to an isomorphism

The first structural metric μ_1 uses a definition of common subgraph of two graphs [24].

Definition 8. *Two graphs G_1 and G_2 are called isomorphic if there is an edge-preserving bijection between their vertex sets. The edge-preserving property states that two vertices are adjacent iff their images are adjacent.*

Definition 9. *Two trees T_i, T_j have a common subtree T if each of them has a subtree isomorphic to T.*

Definition 10. *A size $|T|$ of a tree T is the number of its vertices.*

Definition 11. *The largest common subtree T_{ij} of two directed labeled trees T_i and T_j is the tree of the largest size among all common subtrees of T_i and T_j.*

The distance between T_i and T_j is calculated by the following formula

$$\mu_1(T_i, T_j) = |T_i| + |T_j| - 2|T_{ij}|.$$

The paper [24] defines μ_1 likewise on pairs of graphs and proves that μ_1 satisfies 1-3 conditions if the graph size is defined as the number of its edges. For a tree the number of its vertices is equal to the number of its edges plus 1. Therefore, the results mentioned in [24] are applicable for our case and μ_1 satisfies 1-3 conditions. The last 4th condition is checked empirically.

5.2. Similarity according to edit distance

As before, a superposition is represented by a directed labeled tree. Represent a tree as a string of characters. This string is constructed as a sequence of labels of vertices written in pre-order [25].

Now define a structural metric μ_2 on pairs of character strings. It automatically defines the structural metric on pairs of superpositions. As the arities of functions from $\mathfrak{G}$ are known, each superposition could be reconstructed from its string representation. Therefore, there is no two character strings corresponding to one superposition of primitive functions. The structural metric μ_2 is called a Levenshtein distance.

Definition 12. *The Levenshtein distance between two character strings is the minimum number of single-character edits (insertions, deletions and rewritings) required to change one string into the other.*

Each edit distance satisfies the conditions 1-3. The metric μ_2 also satisfies them in the case when it is defined on pairs of superpositions, because the string representation is bijective. The last 4th condition is checked empirically.

The third structural metric μ_3 is a Levenshtein distance defined on pairs of directed labeled trees.

Definition 13. *The Levenshtein distance between two trees is the minimum number of edits (edge insertions, edge deletions and vertex relabeling) required to change one tree into the other.*

The structural metric μ_3 satisfies the metric axioms [26]. The last 4th condition is checked empirically.

6. Regularizers

To approximate noisy data accurately, the genetic algorithm generate complex superpositions after some iterations. To prevent this overfitting, it should control the structural complexity of superpositions by a regularizer. The regularizer restricts a set $\mathfrak{F}' \subset \mathfrak{F}$ of superpositions reachable by the genetic algorithm. Search for a regularizer, which makes the set $\mathfrak{F}'$ sufficiently rich to find there a proper approximating superposition and sufficiently small to avoid overfitting of the algorithm. Lets consider the structural parameters of directed labeled trees

1) The size of a tree, see Definition 3.
2) The number of leaves in a tree.
3) The height of a tree.

A restriction of these parameters makes complex superpositions unreachable for the genetic algorithm. This paper analyzes three regularizers built on these structural parameters. To penalize accurate superpositions less, all of these regularizers are proportional to MAP.

1) $\mathrm{R}_1(f) = p \cdot \mathrm{MAP}(f) \cdot I(|f| < \mathrm{CT})$,
   where CT is a threshhold for the structural complexity, p is a penalty parameter. The regularizer R_1 penalizes those superpositions, which have structural complexity larger than the threshhold CT.
2) $\mathrm{R}_2(f) = p \cdot \mathrm{MAP}(f) \cdot I(|f| \geq \mathrm{CT}) \cdot (|f| - \mathrm{CT})$,

where C is a positive parameter. The regularizer R_2 penalizes the superpositions having structural complexity larger than the threshhold CT. And the more complex a superposition, the higher the penalty.

3) $\mathrm{R}_3(f) = p \cdot \mathrm{MAP}(f) \cdot |f|^* \cdot \log(|f| + 1)$,
The regularizer R_3 treats a structural complexity of a superposition as the number of leaves $|f|^*$ of its tree multiplied by the estimation $\log(|f| + 1)$ of its height.

All parameters from the definitions should be set empirically. To set them one needs to follow the principle mentioned above: the set $\mathfrak{F}'$ should be sufficiently rich to find there a proper approximating superposition and sufficiently small to avoid overfitting of the genetic algorithm.

Select proper structural metric and regularizer to modify the basic genetic algorithm. The modified version solves the problems of overfitting and evolutionary stagnation. This version is described in Algorithm 2.

7. Computational experiment

The main goal of this paper is to generate superpositions outperforming the ones from $\mathfrak{F}_0$ selected in [2]. These functions, in turn, outperform known ranking models BM25, LGD, $\mathrm{LM}_{\mathrm{DIR}}$. Therefore, if the modified genetic algorithm succeeds in outperforming functions from $\mathfrak{F}_0$, it will also outperform BM25, LGD, $\mathrm{LM}_{\mathrm{DIR}}$ as well. Now describe the data used to estimate the quality of the generated superpositions.

Data. Authors in [2] estimate the quality ranking functions on TRECs. To make the comparison with $\mathcal{F}_0$ consistent, use TRECS as well. Perform the computational experiment on Trec-5, Trec-6, Trec-7, Trec-8 (trec.nist.gov). The Text REtrieval Conference (TREC), co-sponsored by the National Institute of Standards and Technology (NIST) and U.S. Department of Defense, was started in 1992 as part of the TIPSTER Text program. For each TREC, National Institute of Standards and Technology (NIST) provides a test set of documents and questions. Participants run their own retrieval systems on the data, and return to NIST a list of the retrieved top-ranked

Algorithm 2 Modified genetic algorithm

Require: grammar $\mathfrak{G}$, required value α of MAP

Ensure: superposition f of elements from G with MAP $\leq \alpha$;

create a set of initial, random superpositions $\mathfrak{M}_0$,

repeat

- crossover random pairs of stored superpositions $\mathfrak{M}$,
- mutate random superpositions from the population $\mathfrak{M}$,
- consider these generated superpositions and the ones stored in $\mathfrak{M}$. Select the best of them according to the quality function $\mathcal{S}$ (2),
- store the best superpositions in a population $\mathfrak{M}'$ and pass it to the next iteration,
- **if** $d_e(\mathfrak{M}') <$ Thresh **then**
  evolutionary stagnation is detected and we replace the worst superpositions from the population $\mathfrak{M}'$ by random superpositions,
- **end if**
- $\mathfrak{M} = \mathfrak{M}'$.

until the required value of MAP is reached;

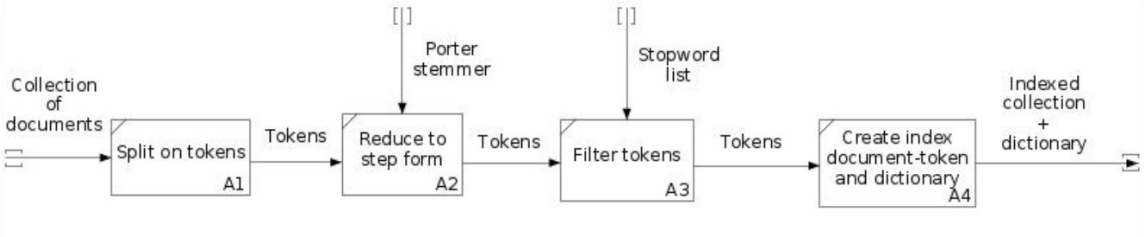

Figure 2: Scheme of data preprocessing steps.

documents. NIST pools the individual results, judges the retrieved documents for correctness, and evaluates the results. Thus each TREC consists of a collection of documents, user queries and judgments for a subset of a collection Each TREC is associated with this triplet. Each triplet has a collection of nearly 500 000 documents. 50 queries to the collection and 2000 judgments for each query in average. The number specified after the name ¡¡Trec¿¿ denotes the year of the creation of the TREC.

7.1. Data processing

As TREC collections are large, calculations of the variables x_w^d and y_w (1) are computationally expensive. To speed up the calculations, one should perform data preprocessing. Terrier IR Platform v3.6 (terrier.org) perform necessary steps for this preprocessing. It provides flexible processing of terms through a pipeline of components (stopwords removing, stemmers, etc.). The platform indexes a collection of documents. The preprocessing steps include stemming using Porter stemmer and removing stop-words using the stopword list. Second, Terrier performs a query expansion techniques and retrieves required documents efficiently. It processes the data stored in Trec5-8 and returns the matrices of features x_w^d and y_w for each word $w \in q$ and each document from the collection having this word.

The algorithm of primary data preprocessing makes the following steps, see Figure 2.

1. Split documents on tokens. Reduce each token to its stem form by Porter stemmer [4].
2. Filter the set of stemmed tokens is according to the stopwords list.

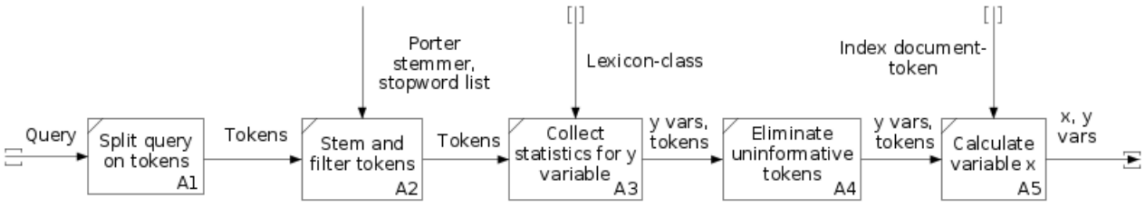

Figure 3: Scheme of query processing steps.

3. The collection is represented as an index *document-token*.
4. Create a *lexicon-class*, which represents the list of terms (dictionary) in the index.

After the preliminary steps are performed, one can calculate the variables x_w^d and y_w for each query q, see Figure 3.

1. Split q on tokens. Process each token by the stemmer and filter the resulted set by the stopword list.
2. *Lexicon-class* collects statistics about the tokens. It calculates the feature y_w.
3. Eliminate tokens with high value of y_w as uninformative.
4. For each token the platform retrieves the information about its second feature x_w^d from the index.

The described scheme is used by the modified genetic algorithm to estimate the quality of a superposition. Now describe the system performing this modified genetic algorithm. This system generates superpositions of primitive functions.

7.2. Generation system

Algorithm 2 gives the description of the modified genetic algorithm used for generation of ranking superpositions. These superpositions are constructed from the elements of $\mathfrak{G} = \{x_w^d, y_w, +, -, \times, \div, \log, \exp, \sqrt{\cdot}\}$.

On each iteration the algorithm stores 20 best generated superpositions. To create new superpositions, it performs 10 crossovers and 10 mutations on the stored ones. Then it selects 20 best according to (2) and pass to the next iteration. This paper terminates the generation after 300 iterations. The selected superpositions are compared with the ones from $\mathfrak{F}_0$ To use this algorithm, one must select proper regularizer and structural metric. The code for this system is found in https://github.com/KuluAndrej/Generation-of-simple-structured-IR-functions-by-genetic-algorithm-without-stagnation.

7.3. Selection of regularizer and structural metric

This paper analyzes three metrics and three regularizers defined above with respect to the genetic algorithm. There are 9 combinations of these metrics and regularizers. Selects the pair, which provides better generation of superpositions both in terms of structural diversity and prediction accuracy. The selected pair is used by the modified genetic algorithm to generate an optimal ranking superposition.

Table 2 shows a computational efficiency of calculation of different metrics with respect to different regularizers. There are 9 possible pairs metric-regularizer. The modified genetic algorithm is launched 100 times for each pair. The CPU time required to calculate all values of a metric is averaged over these 100 launches and 300 iterations for each launch. Table 2 shows that μ_2 is uniformly easiest to calculate. At the same time, μ_1 is uniformly hardest to calculate. This efficiency is considered in the selection. Now analyze the pairs with respect to the generation of superpositions.

First, analyze the modified genetic algorithm without regularizers. All measured values are averaged over 100 launches, see Figure 4. On the last 300-th iteration the average structural complexity of superpositions in the population is more than 40. Figure 4 shows slow trend to evolutionary stagnation. The reason is that structural complexity of the generated superpositions grows dramatically with the iteration number. It makes the stored superpositions sufficiently diverse. Therefore during the whole evolution the empirical diameter d_e of the stored population is large. However, the generated superpositions are significantly overfitted and should be penalized for the excessive structural

complexity.

Regularizer	μ_1	μ_2	μ_3
R_1	11.52	1.84	4.54
R_2	6.7876	0.9347	1.5666
R_3	7.63	1.05	1.87

Table 2: Comparison of CPU time required by structural metrics

Now let us analyze 3 metrics with presence of a regularizer. For each pair metric-regularizer plot the empirical diameter d_e depending on the number of iteration. Figures 5, 6, 7 also shows the average structural complexity l_a of stored superpositions. It allows to make inferences about the presence of overfitting.

Note that the empirical diameter $\mathrm{d}(\mathfrak{M})$ calculated with μ_1 remains approximately unchanged during the whole evolution, see Figures 5, 6, 7. This particular feature does not allow to detect evolutionary stagnation in proper time. The actual start of evolutionary stagnation can not be denoted with μ_1. Moreover, calculation of μ_1 is computationally inefficient comparing with μ_2 and μ_3, see Table 2. These reasons lead to elimination of μ_1 from the further analysis.

Two other metrics μ_2 and μ_3 provide almost equal values of $\mathrm{d}(\mathfrak{M})$, see Figures 5, 6, 7. The relative difference in these values is under 5% for all variants of used regularizer. Therefore, without loss of generality, select the structural metric μ_2 as more efficiently calculated, see Table 2.

The first regularizer R_1 is too strict, see Figure 5. The algorithm falls into evolutionary stagnation on the first iterations, because the set of reachable superpositions $\mathfrak{F}'$ is small. The similar situation is observed for the second regularizer R_2, see Figure 6. The algorithm does not immediately fall into evolutionary stagnation. The stored superpositions are updated up to the 300-th iteration. However, the empirical diameter $\mathrm{d}(\mathfrak{M})$ significantly decreases after 30-40 iterations, see Figure 6. It means that although the stored superpositions are being updated throughout the evolution, they have mainly similar structures. These reasons lead us to the use of the third regularizer R_3. The value of the empirical diameter $\mathrm{d}(\mathfrak{M})$ decreases smoothly with R_3, see Figure 7. It allows to have enough iterations to learn the structure of optimal superposition and detect evolutionary stagna-

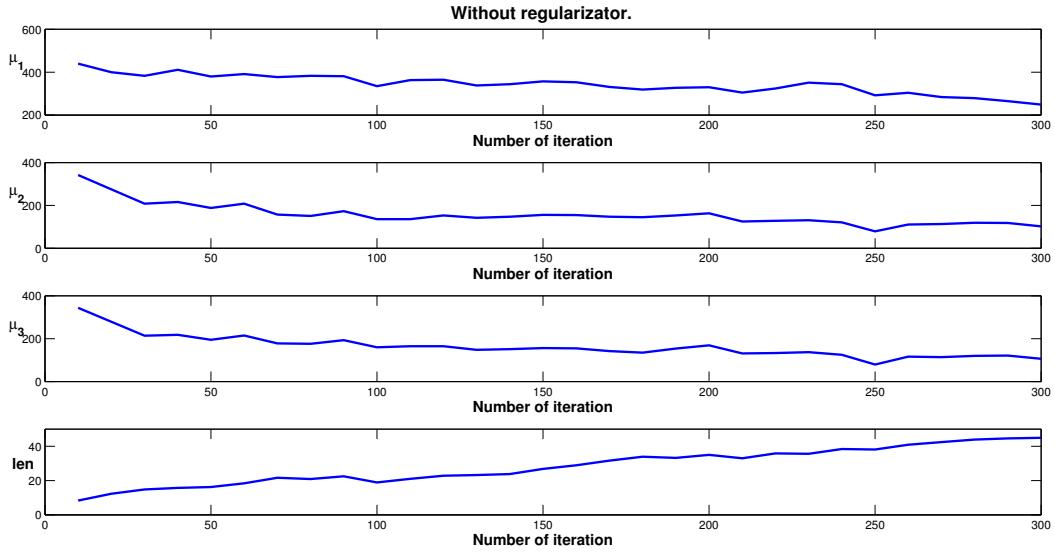

Figure 4: Dynamics of $d(\mathfrak{M})$ and l_a when no regularizer is used.

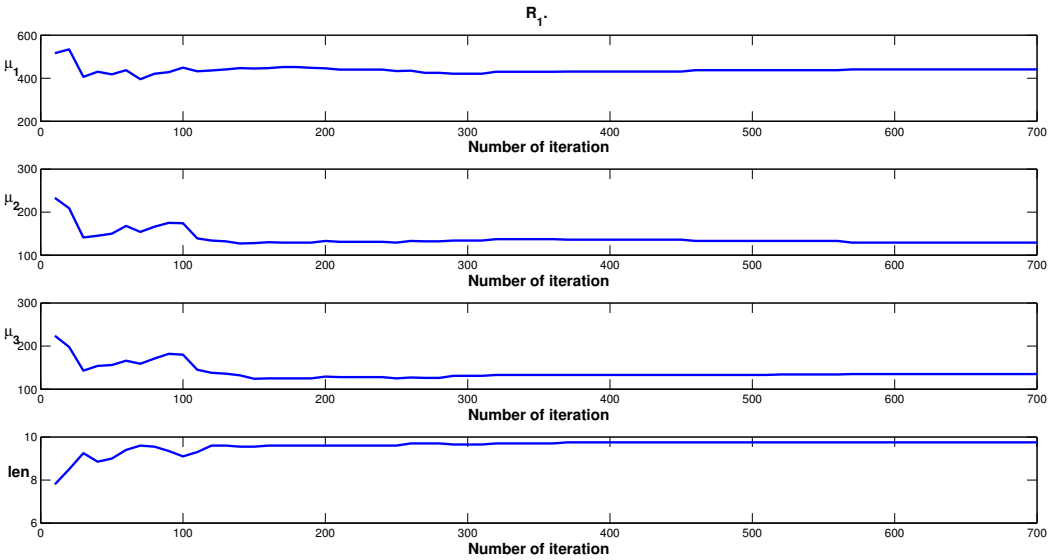

Figure 5: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_1 is used.

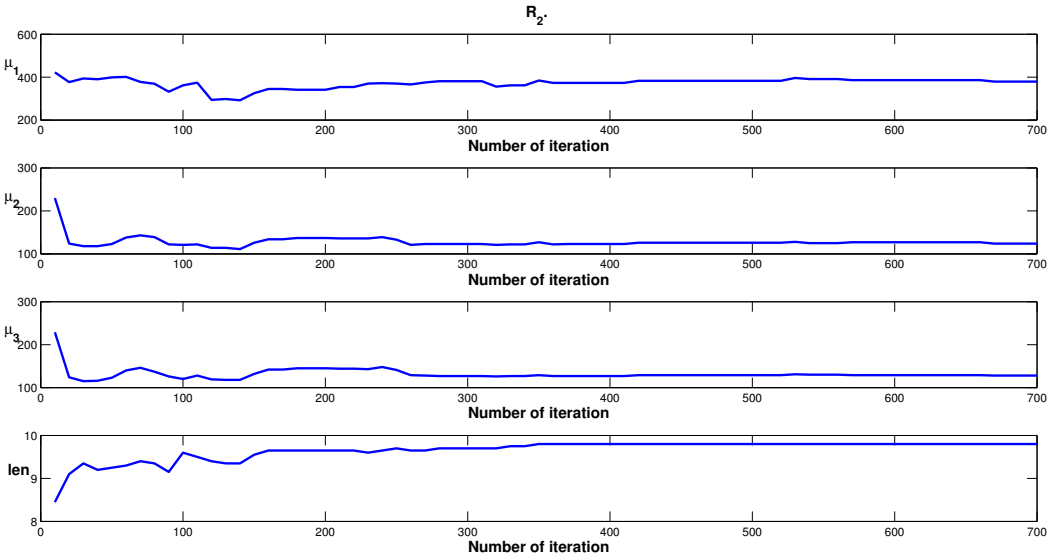

Figure 6: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_2 is used.

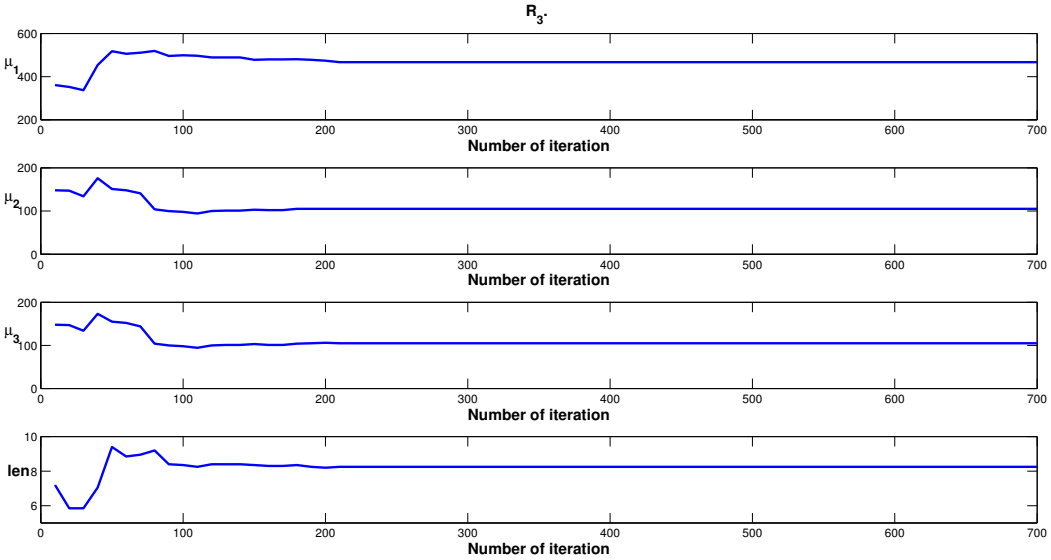

Figure 7: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_3 is used.

tion. Since the structural metric μ_2 and the regularizer R_3 are selected, the modification of the genetic algorithm is ready to generate ranking superpositions. s

Generation of ranking superpositions. Modified genetic algorithm is launched on TREC-7. The best selected superpositions are compared with ones from $\mathfrak{F}_0$. The superpositions in $\mathfrak{F}_0$ are of simple structure and have a high quality in average on analyzed collections. Besides, these superpositions are better in average than the traditionally used ranking models BM25, LGD, $\mathrm{LM}_{\mathrm{DIR}}$. Here is the list of the best superpositions from $\mathfrak{F}_0$

2

1. $f_1 = e^{\sqrt{\ln\left(\frac{x}{y}\right)}}$,
2. $f_2 = \sqrt{\frac{\ln(x)}{\sqrt{y}}}$,
3. $f_3 = \sqrt[4]{\frac{x}{y}}$,
4. $f_4 = \sqrt{y + \sqrt{\frac{x}{y}}}$,
5. $f_5 = \sqrt[4]{\frac{x}{y}} \cdot e^{-y/2}$,
6. $f_6 = \sqrt{\sqrt{x} + \sqrt{\frac{x}{y}}}$.

The selection of the best superpositions is performed by the modified genetic algorithm on TREC-7. The other datasets TREC-5, TREC-6, TREC-8 serve as test datasets. After 1000 iterations the modified genetic algorithm selects the following family of superpositions (for the convenience denote $\ln(x+1)$ as $\ln(x)$ and $g(x) = \ln\ln(x)$):

2

1. $h_1 = g\left(\frac{g(x)}{\sqrt{\ln(x) + x}}\right) - \ln(y)$,
2. $h_2 = g\left(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x) + x}}\right) - \ln(y)$,

3. $h_3 = g\left(\ln\left(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x)+x}}\right) - \ln(y)\right)$,

4. $h_4 = g\left(\frac{g(x)}{\sqrt{g\left(\sqrt{x}\right)+x}}\right) - \ln(y)$,

5. $h_5 = g\left(\frac{g(x)}{\sqrt{\ln(x)+\ln(y)}}\right) - \ln(y)$,

6. $h_6 = g\left(\frac{g\left(\ln(x)\right)}{\sqrt{\ln(x)+x}}\right) - \ln(y)$.

The values of MAP of the superpositions $\{h_j\}$ and $\{f_i\}$ are is presented in Table 3. The superpositions from $\mathfrak{F}_0$ are presented in the upper half of the table. The superpositions $\{h_j\}$ are presented in the lower half. The qualities of the best functions $\{f_i\}$ are bold in each column in the upper half. In the lower half we bold those values, which are higher than the bold values in the corresponding column in the upper half.

Superposition	TREC-5	TREC-6	TREC-7	TREC-8
Superpositions from $\mathfrak{F}_0$				
f_1	8.785	13.715	**10.038**	**13.902**
f_2	8.518	12.996	9.216	13.074
f_3	**8.908**	**13.615**	9.905	13.708
f_4	**8.908**	**13.615**	9.905	13.708
f_5	**8.908**	**13.615**	9.908	13.709
f_6	8.872	13.613	9.890	13.695
Family of selected superpositions				
h_1	**8.965**	**13.693**	**10.600**	**14.403**
h_2	**9.472**	**13.723**	**10.650**	**14.402**
h_3	**9.558**	**13.786**	**10.631**	**14.376**
h_4	**9.226**	**13.713**	**10.5**	**14.374**
h_5	8.862	13.388	**10.439**	**14.359**
h_6	8.104	13.483	**10.421**	**14.355**

Table 3: Comparison of the superpositions $\{h_j\}$ to $\{f_i\}$ according to the MAP criterion

Note that the superpositions $h_1, h_2,\ h_3,\ h_4$ are uniformly better than the functions from [2] on all 4 datasets. The other superpositions are better in average. The modified genetic algorithm is able to build effective yet simple structured superpositions, which outperform the known ones.

8. Conclusion

This paper investigates an Information Retrieval ranking function construction technique. It develops a genetic algorithm, which consequently generates ranking functions. The basic version of this algorithm is inclined to generate overfit ranking functions. To avoid overfitting, one must control their structural complexity and solve an evolutionary stagnation problem. It is solved by use of regularizers and structural metrics respectively. A regularizer, presented in the quality function, controls the structural complexity of the functions. Overfit functions are penalized and unlikely to pass to the next iterations in the genetic algorithm. A structural metric estimates the diversity of the generated functions. If all generated functions are similar to each other, some of them are replaced by random ones. It solves a problem of evolutionary stagnation. This paper analyzes different regularizers and structural metrics and chooses those, which provide a better generation. The modified genetic algorithm uses the selected pair metric-regularizer and generate effective yet simple structured functions. These functions outperform BM25, LGD, LMDIR. and the ones selected by are exhaustive approach.

Future research TODO

9. Acknowledgements

This research is supported by RFBR grant 16-37-00486.

References

[1] C. D. Manning, P. Raghavan, H. Schutze, Introduction to Information Retrieval, Cambridge University Press, Cambridge, UK, 2008.
URL http://nlp.stanford.edu/IR-book/information-retrieval-book.html

[2] P. Goswami, S. Moura, E. Gaussier, M.-R. Amini, F. Maes, Exploring the space of ir functions, in: ECIR'14, 2014, pp. 372–384.

[3] Z. Yea, J. X. Huangb, H. Lina, Incorporating rich features to boost information retrieval performance: A svm-regression based re-ranking approach, Expert Systems with Applications 38 (6) (2011) 75697574.

[4] M. F. Porter, Readings in information retrieval, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, Ch. An Algorithm for Suffix Stripping, pp. 313–316.

[5] D. Metzler, W. B. Croft, A markov random field model for term dependencies, in: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '05, ACM, New York, NY, USA, 2005, pp. 472–479.

[6] G. Amati, C. J. Van Rijsbergen, Probabilistic models of information retrieval based on measuring the divergence from randomness, ACM Trans. Inf. Syst. 20 (4) (2002) 357–389.

[7] S. Clinchant, E. Gaussier, Information-based models for ad hoc ir, in: Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '10, ACM, New York, NY, USA, 2010, pp. 234–241.

[8] J. M. Ponte, W. B. Croft, A language modeling approach to information retrieval, in: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '98, ACM, New York, NY, USA, 1998, pp. 275–281.

[9] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., New York, NY, USA, 1986.

[10] M. Gordon, Probabilistic and genetic algorithms in document retrieval, Commun. ACM 31 (10) (1988) 1208–1218.

[11] H. Valizadegan, R. Jin, R. Zhang, J. Mao, Learning to rank by optimizing ndcg measure, in: Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, A. Culotta (Eds.), Advances in Neural Information Processing Systems 22, Curran Associates, Inc., 2009, pp. 1883–1891.

[12] M. D. Gordon, User-based document clustering by redescribing subject descriptions with a genetic algorithm, JASIS 42 (5) (1991) 311–322.

[13] V. Raghavan, B. Agarwal, Optimal determination of user-oriented clusters: An application for the reproductive plan, in: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1987, pp. 241–246.

[14] J. Yang, R. Korfhage, E. M. Rasmussen, Query improvement in information retrieval using genetic algorithms - A report on the experiments of the TREC project, in: TREC, 1992, pp. 31–58.

[15] F. E. Petry, B. P. Buckles, T. Sadasivan, D. H. Kraft, The use of genetic programming to build queries for information retrieval., in: International Conference on Evolutionary Computation, 1994, pp. 468–473.

[16] D.-Y. Chiu, Y.-C. Pan, S.-Y. Yu, An automated knowledge structure construction approach: Applying information retrieval and genetic algorithm to journal of expert systems with applications, Expert Systems With Applications 36 (5) (2009) 9438–9447. doi:10.1016/j.eswa.2008.12.032.

[17] H. Chen, Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms, Journal of the American Society of Information Science 46 (3) (1995) 194–216.

[18] H. Billhardt, D. Borrajo, V. Maojo, Using genetic algorithms to find suboptimal retrieval expert combinations., in: SAC, ACM, 2002, pp. 657–662.

[19] P. Pathak, M. Gordon, W. Fan, Effective information retrieval using genetic algorithms based matching function adaptation, in: in Proceedings of the 33rd Hawaii International Conference on System Science (HICSS, 2000.

[20] C.-H. Lina, H.-Y. Chenb, Y.-S. Wua, Study of image retrieval and classification based on adaptive features using genetic algorithm feature selection, Expert Systems with Applications 41 (2014) 66116621.

[21] W. Fan, M. D. Gordon, P. Pathak, A generic ranking function discovery framework by genetic programming for information retrieval, Inf. Process. Manage. 40 (4) (2004) 587–602.

[22] W. Fan, M. D. Gordon, P. Pathak, Personalization of search engine services for effective retrieval and knowledge management, in: Proceedings of the Twenty First International Conference on Information Systems, ICIS '00, Association for Information Systems, Atlanta, GA, USA, 2000, pp. 20–34.

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

[24] Makarov, Metric properties of the functions of distances between molecular graphs, Journal of Structural Chemistry 2 (2007) 223–229.

[25] J. M. Morris, Traversing binary trees simply and cheaply., Inf. Process. Lett. 9 (5) (1979) 197–200.
URL http://dblp.uni-trier.de/db/journals/ipl/ipl9.html#Morris79a

[26] K. Zhang, D. Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems, SIAM J. Comput.