Isomorphism-Based Simplification of Evolutionary Generated Models in Symbolic Regression

Andrei Kulunchakov*, Vadim Strijov† *Moscow Institute of Physics and Technology, kulu-andrej@yandex.ru
†Dorodnicyn Computing Centre, CSC IC RAS, strijov@ccas.ru

***Abstract*—This paper investigates a genetic algorithm of the inductive model generation and model selection. A model is the superposition of primitive functions and represented as a directed labeled tree. Some of generated superpositions could be simplified so that their maps remain unchanged. A procedure performing this simplification is proposed. This procedure is based on a search algorithm that finds all isomorphic subtrees in a tree. As a result, some subtrees are substituted with another subtrees of lesser structural complexity. The proposed procedure reduces both structural complexity and dimensionality of the parameter space of a superposition. European stock options trading data is used to illustrate the procedure. The simplification is shown to improve the quality of superpositions with respect to a functional given in the paper. It also allows genetic programming to reach more accurate superpositions in the same number of iterations.**

***Index Terms*—isomorphic trees, graph rewriting system, dimensionality reduction, genetic programming.**

I. Introduction

THIS paper investigates symbolic regression [1]–[4] to generate forecasting models. The term symbolic regression represents a process in which measured data is fitted by a suitable mathematical formula [5]. There are a number of approaches to construct models for symbolic regression [5]. This paper develops genetic programming [6] approach. In genetic programming superpositions are constructed from expert-given primitive functions [7] to fit data. The superpositions are generated iteratively with crossing and mutation operations [6]–[8].

The problem specific for the genetic programming is overfitting [9]–[11]. Possible solutions to this problem are random sampling technique [12], interleaved sampling [13], minimising testing [14]. Another way to avoid overfitting is simplification of generated superpositions. This paper considers simplification, which preserves the mappings of superpositions [7]. Superpositions are represented as *directed labeled trees*, see Figure 1. A graph rewriting system [15] simplifies these trees.

To simplify a tree a rule-based graph rewriting system uses a set of expert-given graph rewriting rules. Define a graph rewriting rule (P,R) as a pair of directed labeled trees: a *pattern* tree P and a *replacement* tree R [16], see Figure 2. The representation of an input superposition as a *directed labeled tree* is called *host* tree H [16]. Graph rewriting substitutes the occurrence of P in H with the corresponding replacement tree R. To perform simplifications, assume that replacements trees have less cardinality of their vertex sets than the corresponding pattern trees. Imply that the total numbers of parameters written in the vertices in replacement trees are less than those in the corresponding pattern trees. Therefore the rule-based graph rewriting system reduces both structural complexity of a superposition and dimensionality of its parameter space. Consider a rule

$$2\cos(x)\sin(x) \to \sin(2x).$$

An example of a graph rewriting built on this rule is presented below

$$f_0 = 2\cos\left(f(w,x,y)\right)\sin\left(f(w,x,y)\right) \to \sin\left(2\cdot f(w,x,y)\right),$$

The possibility of such substitution of a variable x with another function $f(w,x,y)$ is verified by an algorithm detecting all isomorphic subtrees in a tree. It halves the dimensionality of the parameter space of f_0 and significantly reduces its structural complexity. It plays a crucial role in the improving of the superpositions fitting. There are less calculations and less parameters necessary to fit a superposition.

Superposition-tree for

$x_1 \tan(x_2)$.

Primitive functions:

$\times$, $\tan$.

Variables:

x_1, x_2.

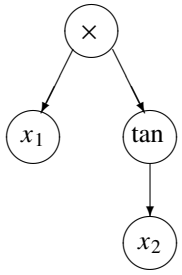
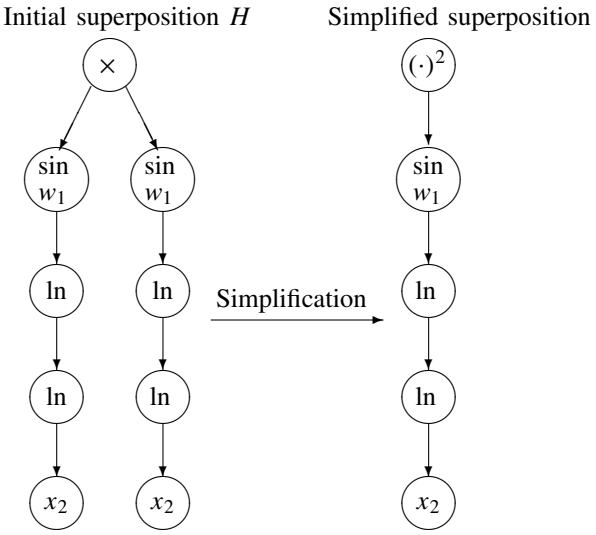


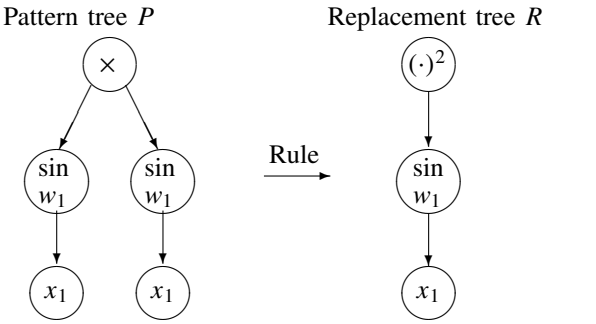
Fig. 1: Superposition represented as a directed labeled tree.

After the generation by genetic programming some of the superpositions have excessive structural complexity. To reduce it they should be simplified. They are compared with themselves after the simplification with respect to an error function and Akaike information criterion [17]. The final selected superpositions are compared for both initial and modified versions of genetic programming.

The detection of an isomorphism between two graphs is NP-hard problem [18]. It is shown to be solved in quasi-polynomial time [19]. However, there are several polynomial time algorithms [18], [20] for some special types of graphs. The $O(n)$-time algorithm [18] detects an isomorphism between two trees. An isomorphism between a tree and a subtree of another tree is recognized by the $O\left(\frac{k\sqrt{k}n}{\log k}\right)$-time algorithm, where k,n are the numbers of the vertices in the trees [20].

Supported by RFBR grant 16.07-01172. Manuscript received -; revised -.
Corresponding author: A. Kulunchakov (email: kulu-andrej@yandex.ru).





Simplification rule:

$$\sin(w_1 \cdot x_1) \cdot \sin(w_1 \cdot x_1) = (\sin(w_1 \cdot x_1))^2.$$

Substitutions:

$x_1 = \ln(\ln(x_2))$,

$x_2 = \mathrm{id}$.

Fig. 2: Simplification of $f(x_1, x_2) = \sin(w_1 \ln(\ln(x_2))) \cdot \sin(w_1 \ln(\ln(x_2)))$ by rule.

The substitution of expressions in a superposition mentioned above is provided by an algorithm detecting all pairs of isomorphic subtrees in a tree. This paper presents a polynomial time variant of this algorithm. The idea of this algorithm is to split the set of vertices into equivalence classes [18]. Each two vertices from one equivalence class are roots of isomorphic subtrees. The algorithm performs this splitting in two steps. First, split the set of vertices into groups according to their *height*. Second, make each equivalence class within one of these groups. This algorithm has computational complexity $O(n \log(n))$, where n stands for the number of vertices in a tree. This computational complexity is significantly lesser than one described in [21], which is above $O(n^2 \log(n))$. Small computational complexity is an important feature, because the model generation should not be slowed down by a simplification procedure.

Algorithm 1 Genetic programming implemented by MVR

Require: initial superpositions, a set of primitive functions $\mathfrak{G}$
Ensure: superposition f with MSE $\leq \alpha$
repeat

- apply the rule-based graph rewriting system to simplify current set of superpositions,
- estimate their parameters by the Levenberg-Marquardt algorithm,
- apply the cross-mutation algorithm [8] to generate new superpositions,
- estimate the optimal parameters of new superpositions,
- calculate the errors $\mathcal{S}(f)$ and MSE,
- select the best superpositions according to $\mathcal{S}$ and pass them to the next iteration

until required MSE is reached. =0

To generate superpositions we use genetic programming [8] implemented in Multivariate Regression Composer [22] software, see Algorithm 1. MVR uses expertly-given primitives $\mathfrak{G}$ and initial superpositions. Using crossing and mutation operations [6], [8] it iteratively generates new populations of superpositions. The optimal parameters of the generated superpositions are estimated by the Levenberg-Marquardt algorithm [23]. Evaluate the error function $\mathcal{S}$ on the generated superpositions. The best superpositions are passed to the next iteration. The iterations are terminated when a desired error of prediction is reached.

To test the proposed graph rewriting system the paper uses the European stock options trading data. An option is a contract giving the owner the right, but not the obligation, to sell a specified amount of an underlying asset at a set price within a specified time called expiration date [24]. A set price of an option is called a strike price [24].

Theoretical estimation of the fair market value of European-style options is given by Black-Scholes formula [24]. It has only one parameter that can not be observed in the market. This parameter is the average future volatility of the underlying asset. Volatility is the degree of variation of a trading price series over time as measured by the standard deviation of returns. In this paper, we investigate the volatility of a financial instrument over a specified period starting at the current time and ending at the expiration date of an option. It is estimated by the market price of the instrument in the assumption that the price is relevant to expected risks.

The Black-Sholses model relies on the assumptions which imply the independence of the volatility from the strike price and the expiration date. However, in practice the volatility depends on these two values. We assume that implied volatility value σ^{imp} is calculated as an argmin of the difference between the historical (stated on the trade) and the fair strike price in the Black-Scholes model:

$$\sigma^{\mathrm{imp}} = \mathrm{argmin}(C_{\mathrm{hist}} - C(\sigma, \mathrm{Pr}, B, K, t)), \tag{1}$$

where C_{hist} is the historical strike price; C is the fair strike price estimated by the Black-Scholes model; Pr is the price of the instrument; B is the bank rate; K is the strike price; t is the time left to the expiration date.

To estimate the fair strike price one must approximate the dependence $\sigma(K,t)$ between the strike price of the instrument, its volatility and the time left to the expiration date. The dataset is collected and described in [25].

In this paper we find a superposition which approximates this dependence. This superposition must be better than the one given by experts [25]

$$\sigma = \sigma(\mathbf{w}) = w_1 + w_2(1-\exp(-w_3x^2)) + \frac{w_4\arctan(w_5x)}{w_5}, \quad (2)$$

where

$$x = \frac{\ln K - \ln C(t)}{\sqrt{t}},$$

with the parameter vector $\mathbf{w} = [w_1,\dots,w_5]$.

II. Problem statement

There given a dataset $\mathfrak{D} = \{(\mathbf{x}_s, y_s)\}_{s=1}^N$ and a set of primitive functions

$$\mathfrak{G} = \{g_1,\dots,g_m,x_1,\dots,x_z\}, \quad (3)$$

where each g_k stands for an mathematical function and each x_j stands for a variable. Denote $\mathfrak{F}$ as a set of superpositions of the elements from $\mathfrak{G}$. The superpositions are represented as directed labeled trees. The vertices of these trees are labeled by elements from $\mathfrak{G}$.

The paper is aimed at finding the superposition $f(\mathbf{w},\mathbf{x})$ from $\mathfrak{F}$, which minimizes the error function $\mathcal{S}$

$$\mathcal{S} = \mathrm{MSE} + \lambda\|\hat{\mathbf{w}}\|_2^2 + C(f(\hat{\mathbf{w}},\mathbf{x})), \quad (4)$$

where λ is a positive regularization parameter and MSE is the mean squared error of approximation

$$\mathrm{MSE} = \frac{1}{N}\|y_s - f(\mathbf{w},\mathbf{x}_s)\|^2.$$

The vector $\hat{\mathbf{w}}$ is the optimal parameters according to MSE. To prevent the genetic algorithm from overfitting introduce $C(f(\hat{\mathbf{w}},\mathbf{x}))$ controlling structural complexity of superpositions.

Structural complexity $\phi(f)$ of a superposition f is the number of elements from $\mathfrak{G}$, which the superposition f is comprised of.

Introduce threshhold Φ determining the maximum acceptable structural complexity of a superposition. The superpositions with the structural complexity above the threshhold are penalized. The corresponding penalties are denoted by κ_1 and κ_2, $\kappa_1 < \kappa_2$,

$$C(f) = \begin{cases} \mathrm{MSE}\cdot\kappa_1\cdot\phi(f), \text{ if } \phi(f) < \Phi, \\ \mathrm{MSE}\cdot(\kappa_2(\phi(f)-\Phi)+\kappa_1\cdot\Phi), \text{else}. \end{cases} \quad (5)$$

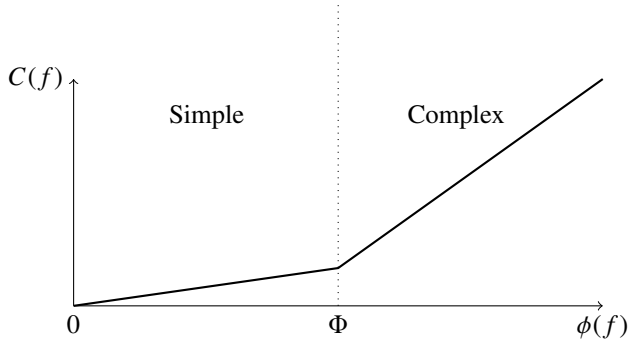


Fig. 3: Penalty on structure complexity.

The problem is to select the superposition $\hat{f} \in \mathfrak{F}$ which minimizes the error function $\mathcal{S}$:

$$\hat{f} = \underset{f\in\mathfrak{F}}{\operatorname{argmin}}\,\mathcal{S}(f|\hat{\mathbf{w}},\mathfrak{D}),$$

where the vector $\hat{\mathbf{w}}$ minimizes mean squared error:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\,S(\mathbf{w}|\ f,\mathfrak{D}).$$

III. Algorithm detecting isomorphic superposition-subtrees

Superpositions generated by genetic programming are represented as directed labeled trees. The set of labels is denoted as $\mathfrak{G}$ (3). Describe the algorithm, which detects all isomorphic subtrees in a superposition-tree. Superposition-tree $T = (V,E)$ is directed rooted tree with labeled vertices V. The label of a vertex v is denoted as $l(v) \in \mathfrak{G}$. Variables x from $\mathfrak{G}$ are assigned to the leaves of T. The other vertices are assigned with mathematical functions from $\mathfrak{G}$. If a vertex has a label corresponding to a commutative mathematical function, claim that its children are lexycographically ordered. Subtree $T' = (V',E')$ of a superposition-tree $T = (V,E)$ is a superposition-tree having $V' \subset V, E' \subset E$.

To detect all isomorphic subtrees in superposition-tree $T = (V,E)$ split the vertex set V into equivalence classes according to the following rule: each equivalence class C consists of those vertices, which are the roots of isomorphic subtrees of T. Two superposition-trees $T_1 = (V_1,E_1)$ and $T_2 = (V_2,E_2)$ are isomorphic iff there is a bijective function preserving labels and edges existence.

To perform it, first, split the vertex set V into levels

$$V = \bigcup_{i=0}^{h(r)} L_i. \quad (6)$$

A level L_i of superposition-tree T consists of vertices of T which have the heights equal to i. A height $h(v)$ of vertex v of a superposition-tree T is the length of the longest path from v to any of the leaves of T.

Each two vertices from one equivalence class have equal heights as well. Describe the algorithm of splitting

The heights h of vertices are found by the depth first search algorithm [18]. Since the heights are known, the levels $L_0,\dots,L_{h(r)}$ are found. Each equivalence class C entirely lies within one of these levels. The classes lying in one level

are ordered. Therefore, each class C has 2 indices $C = C_{i,j}$. The first index indicates the level L_i comprising the class and the second is the index of the class in L_i.

$$L_i = \bigcup_{j=0}^{\alpha_i} C_{i,j}. \tag{7}$$

Define an *index function* $c(v)$ on the vertices of T. If a vertex v lies in class $C_{i,j}$, the value of $c(v) = (i,j)$. In other words, $c(v)$ is the two indices of the class $C_{c(v)}$ comprising v.

$$c(v) = (i,j) : v \in C_{i,j}. \tag{8}$$

For example, the values of $c(v)$ for the graph from Figure 4 are the following (vertices are numbered from left to right)

TABLE I: Values of $c(v)$ for the graph in Figure 4.

	v_1	v_2	v_3	v_4	v_5	v_6
$c(v)$	(0,0)	(0,0)	(0,1)	(0,1)	(0,2)	(0,3)

Describe the iterative procedure of splitting of levels into equivalence classes (7). Start with the lowest level L_0 and split it according to the vertices labels $l(v)$, see Figure 4. Now inductively traverse the levels from L_1 to the root level $L_{h(r)}$. Suppose, the levels $L_0, L_1, \dots L_i$ are processed. Now proceed with the level L_{i+1}. Consider the directed graph $G_i = L_i \cup L_{i+1}$, see Figure 5.

The heads of all arcs lie in L_{i+1} and their tails lie in L_i. Consider each vertex v from L_{i+1} and the arcs $e_1 = (v \to v_1), e_2 = (v \to v_2), \dots$ going from v to L_i. Assign sorted tuple $\langle l(v), c(v_1), c(v_2), \dots \rangle$ to v. Required equivalence classes $C_{i+1,j}$ consist of the vertices having equal tuples. Therefore, sort the list of assigned tuples, traverse this list and create the required classes $C_{i+1,j}$. Figure 5 shows that the first two vertices of L_{i+1} make up class $C_{i+1,0}$ and the last one makes up the class $C_{i+1,1}$. This inductive step is described in the Algorithm 2.

Algorithm 2 Inductive step to split levels on classes

Require: Superposition-tree T with root r and levels $L_0, L_1, \dots, L_{h(r)}$;
Ensure: equivalence classes $\{C_{i,j}\}_{i,j=0}^{i=h(r),j=\alpha_i}$;
  split L_0 according to the vertices labels;
  for each level L_i from $L_1, \dots, L_{h(r)}$ **do**
    for each vertex v from L_i **do**
      Denote $(v, v_1'), (v, v_2')$ as the arcs going from v
      Assign sorted tuple $\langle l(v), c(v_1'), c(v_2'), \dots \rangle$ to the vertex $v \in L_i$;
    end for
    Traverse the list of assigned tuples and form the classes $\{C_{i,j}\}_{j=1}$ of equal tuples.
  end for
  return $\{C_{i,j}\}_{i,j=0}^{i=h(r),j=\alpha_i}$. =0

Finally, the vertex set is split into equivalence classes

$$V = \bigcup_{i,j=0}^{i=h(r),j=\alpha_i} C_{i,j}. \tag{9}$$

Computational complexity of the algorithm is expressed as follows

Computational complexity δ_1 of the algorithm detecting isomorphic subtrees in superposition-tree $T = (V, E)$ in the "big-O" [26] notation

$$\delta_1 = O(m|V| \log(|V|)),$$

where m is the maximum outgoing degree of the vertices from V.

The depth-first search algorithm calculates the values of heights h. Its computational complexity is $O(|V|)$. The leaves of T are split into classes $\{C_{0j}\}_{j=0}^{\alpha_0}$ through sorting their labels in $O\left(|L_0| \log\left(|L_0|\right)\right)$ time. Evaluate complexity of the inductive step of the algorithm.

Assume that vertices of L_{i+1} are ordered. Define E_k as a set of arcs outgoing from k-th vertex of L_{i+1}. Traverse sets $\{E_k\}_{k=1}^{|L_{i+1}|}$ to assign tuples to the vertices of L_{i+1} and sort these tuples. Its computational complexity is

$$\sum_{k=1}^{|L_{i+1}|} O\left(|E_k| \cdot \log(|E_k|)\right) \le \sum_{k=1}^{|L_{i+1}|} O\left(|E_k| \cdot \log(|V|)\right) = \log\left(|V|\right) \cdot \sum_{k=1}^{|L_{i+1}|} O\left(|E_k|\right).$$

Sort the list of tuples. To compare two tuples for equality one needs $O(\max_k(|E_k|))$ time. Computational complexity of the sorting is estimated

$$O\left(\max_k(|E_k| \cdot |L_{i+1}| \log(|L_{i+1}|)\right) \le O\left(m \cdot |L_{i+1}| \log(|V|)\right),$$

where $m = \max_k\left(|E_k|\right)$.

Finally, traverse the list of tuples and split L_{i+1} into equivalence classes. Computational complexity of this step is $O\left(m \cdot |L_{i+1}|\right)$.

Therefore, the algorithm has computational complexity δ_1 bounded from above by

$$O(|L_0| \log(|L_0|)) + \log(|V|) \sum_{i=1}^{h(r)} \left[O(m|L_k|) + \sum_{k=1}^{|L_k|} O\left(|E_k|\right) \right].$$

Note that

$$\sum_{k=1}^{|V_i|} O(|E_k|) \le O(|L_i| \cdot m),$$

and rewrite the previous expression:

$$\delta_1 \le O\left(|L_0| \log(|L_0|)\right) + m \cdot \log(|V|) \cdot \sum_{k=1}^{h(r)} O(|L_k|) \le O(m \cdot |V| \log(|V|)).$$

A superposition-tree defines a superposition of primitive functions. Generally these primitive functions has the arities bounded by 2. Then the estimation improves to

$$\delta_1 = O\left(|V| \log(\max_k |L_k|)\right).$$

Presented algorithm is noticably faster than the algorithm described in [18], which has computational complexity $O(n^3)$. Because of low computational complexity it is used in the rule-based graph rewriting system described below.

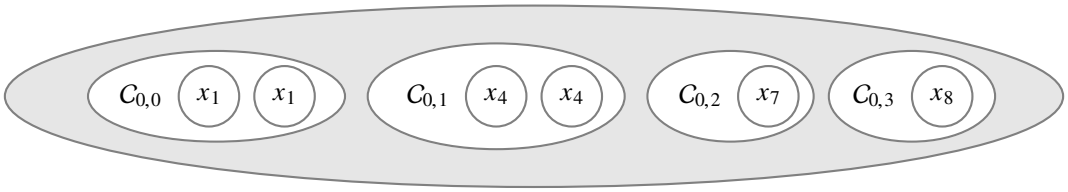


Fig. 4: Example of the lowest level being split $L_0 = \bigcup_{j=0}^{3} C_{0,j}$.

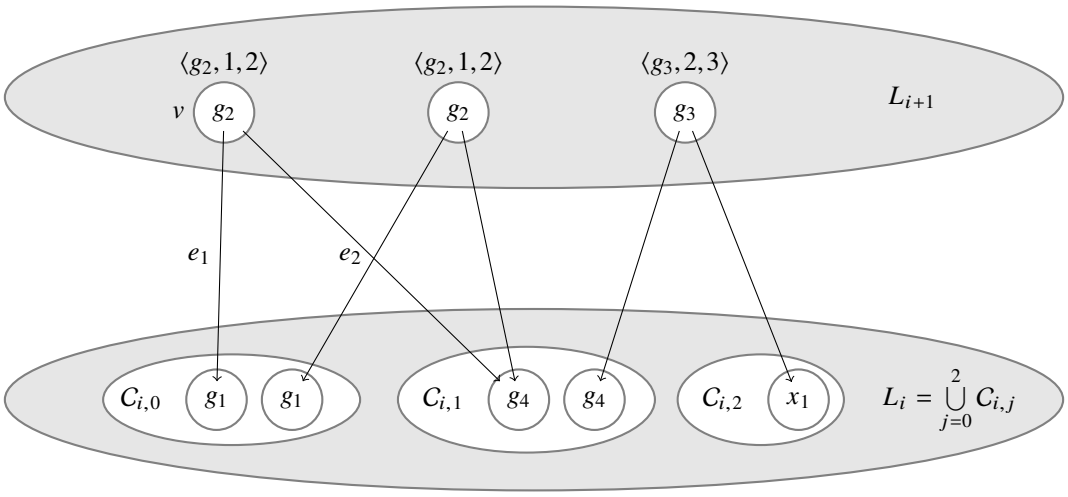


Fig. 5: Example of subgraph $G = L_i \bigcup L_{i+1}$.

IV. Rule-based graph rewriting system

We propose a graph rewriting system to simplify the superpositions generated by MVR. The system uses expert-given set of simplification rules.

A rule of rewriting is an ordered pair of superposition-trees (P, R) whose corresponding mathematical functions have the same mappings. They are called pattern and replacement trees respectively.

Consider a host tree H and a pattern tree P, see Figure 2 and Figure 2. All leaves of P are substituted by another superposition-trees. The leaves with the same labels are substituted with the same superposition-trees. If H is isomorphic to the resulted tree, H *matches* the pattern tree P. An example of matching and leaves substitution is shown in Figure 2.

The set of Z expert-given rules $((P, R))$ is given. Simplify superposition f in two steps. First, represent f as a superposition-tree H and split its vertex set into equivalence classes according to an isomorphism (9). Second, detect all subtrees in H matching any of the pattern trees from the rules. Replace them with the corresponding replacement tree from the rules. Describe the subtree detection procedure.

Consider a pattern tree P with root q. Search for the vertices of H having the same label with q. Denote v as one of such vertices and T as the corresponding subtree with root in v. For example, in Figure 2 such vertex v is the root of the host tree. Launch breadth-first search algorithm [18] to check if T matches P. At each step consider a pair (v, p) of vertices from T and P respectively. If p is a leaf of P, substitute it with the subtree rooted in v, see Figure 2. To check the validity of the final substitution, verify that the leaves with equal labels are substituted by isomorphic subtrees. For example, in Figure 2 both leaves of P are substituted by the same subtrees. To perform it mark p with a pair of indices $c(p)$ (8)

$$c(p) = c(v).$$

It means that the leaves with equal pairs of indices were substituted by isomorphic subtrees. If the breadth-first search is terminated and all leaves of L are marked with new labels, one must verify if the substitution is correct: the leaves with equal labels must be substitited by isomorphic superpositions. If there exists such pair of leaves (p, p') of P that $l(p) = l(p')$ and $c(p) \neq c(p')$, the subtree T does not match P. Otherwise, it matches P. The described algorithm is presented in Algorithm 3.

Computational complexity of the implementation of the rule-based graph rewriting system is expressed as follows

Computational complexity δ_2 of the Algorithm 3 searching for a subtree of $T = (V, E)$ matching to a rule (P, R) from the expert-given set of rules in the "big-O" [26] notation

$$\delta_2 = O\left(Z \cdot |V|\right) + O(|V| \log(|V|)),$$

where Z is the number of rules.

Consider a pattern tree P with root q. Searching for the vertices of T having the same label with q requires $O(|V|)$ time. For each candidate v we launch the verification Algorithm 3 in it. The Algorithm requires $O(|P| \log(|P|)$ time to check if

Algorithm 3 Check if a subtree matches a rule

Require: subtree T and pattern tree P with the roots r and q respectively,
index function $c(v)$,
leaves of P are indexed by integers from 1 to l_0.
Ensure: Does T match to P?
Define an empty *array* of size l_0 and an empty *queue*,
push the pair (r,q) to *queue*.
while *queue* is not empty **do**
pop the top pair (v,p) from *queue*.
if p is a leaf of P with index i **then**
$array[i] = c(v)$,
else
if $l(p) = l(v)$ **and** outgoing degrees are equal **then**
Push the tail of each outgoing arc (p,p_γ) and (v,v_γ) to *queue*,
else
break **while**-loop.
end if
end if
end while
if *array* has unassigned values **then**
return *no*.
end if
Sort *array* with respect to the labels $l(p)$ of the leaves of P, search for indeces i,j such that $(l(v_i) == l(v_j))$ **and** $(array[i] \neq array[j])$.
if such indeces exist **then**
return *no*,
else
return *yes*.
end if=0

the subtree with root in v matches P. This procedure is done at most $|V|$ times for each rule. The found occurence of P is replaced with R. To perform this replacement correctly it is necessary to apply the substitutions found in Algorithm 3 to R, see Figure 2. This replacement requires $O(|V|)$ operations. Therefore, the complexity δ_2 of the Algorithm 3 is

$$\delta_2 = O\left(Z \cdot |V| \max(|P| \log(|P|))\right) + O(|V| \log(|V|)).$$

Generally the pattern trees have rather small size and are bounded by some constant. This notion improves the expression above.

$$\delta_2 = O\left(Z \cdot |V|\right) + O(|V| \log(|V|)).$$

This outperforms the rule-based graph rewriting system described in [21]. It allows to use this rewriting system in the superpositions generation without it being slowed down.

V. Computational experiment

A. Data

To test the proposed graph rewriting system generate superpositions fitting the European stock put options [25]. The primitive fucntions used in the generation are listed in Table II.

TABLE II: Primitive functions used in MVR.

Name	Function	#param-s
expl	$\exp(w1 \cdot x)$	1
inv	$1/x_1$	0
frac2	x_1/x_2	0
hyperbola	w_1/x_1	1
linear	$w_1 \cdot x_1 + w_2$	2
ln	$\ln x_1$	0
minus2	$x_1 - x_2$	0
mult	$w_1 \cdot x_1$	1
normal	$w_1 \cdot \exp(w_1 \cdot (x_1 - w_2)^2)$	3
parabola	$w_1 \cdot x_1^2 + w_2 \cdot x_1 + w_3$	3
omexpl	$1 - exp(w_1)$	1
plus	$x_1 + w_1$	1
plus2	$x_1 + x_2$	0
sin	$\sin x_1$	0
sina	$\sin(w_1 \cdot x_1 + w_2)$	2
sqrt	$\sqrt{x_1}$	0
sqrta	$\sqrt{w_1 \cdot x_1 + w_2}$	2
tansig	$\text{tansig}(x_1)$	0
times2	$x_1 \cdot x_2$	0

TABLE III: Rules of simplification

inv(expl(x)) → expl(x)
normal(linear(x)) → normal(x)
arctanl(mult(x)) → arctanl(x)
times2(linear(x),linear(x)) → parabola(x)
times2(linear(x),plus(x)) → parabola(x)
plus2(linear(x),linear(x)) → linear(x)
plus2(linear(x),plus(x)) → linear(x)
plus2(parabola(x),mult(x)) → parabola(x)
minus2(parabola(x),parabola(x)) → parabola(x)
minus2(linear(x),mult(x)) → linear(x)
minus2(parabola(x),linear(x)) → parabola(x)
minus2(parabola(x),plus(x)) → parabola(x)
minus2(linear(x),x) → linear(x)
plus2(linear(x),x) → linear(x)
plus2(x,parabola(x)) → parabola(x)

The trading data is related to Brent Crude Oil, symbol NYM. Use daily closing prices of an option and underlying. The set of the striking prices $\mathcal{K} = [18.0, 19.0 : 0.5 : 23.0, 24.0 : 0.5 : 28.5]$. Use historical prices of options $C_{K,t}$ and underlying Pr_t, where $K \in \mathcal{K}$, $t \in T$. The sample $\{(\mathbf{x}_s, y_s)\} = \{(\langle K_s, t_s\rangle, \sigma_s)\}$ is built with use of these data as follows. For each values K and t calculate the volatility σ by (1), where the bank rate $B = 0.075$. The time t is expressed in years left to the expiration. There are 112 timestamps in the data.

In the computational experiment 72 rules are used. The first 15 of them are listed in Table III. These rules reduce the dimensionality of the parameter space of a superposition. The initial superpositions for MVR include the etalon function (2).

1) $f_1 = \left(w_1 + w_2 K + w_3 K^2 + w_4 \exp(-w_5 K^2)\right) \sqrt{w_6 t}$,
2) $f_2 = w_1 + w_2 K + w_3 t + w_4 K t + w_5 K^2 + w_6 t^2$,
3) $f_3 = (w_1 + w_2 K + w_3 \exp(-w_4 K^2))(w_5 t^3 + w_6 t^2 + w_7 t + w_8)$,
4) $f_4 = w_1 + w_2(1 - \exp(-w_3 x^2)) + \dfrac{w_4 \arctan(w_5 x)}{w_5}$.

B. Plan of experiment

To analyze the proposed rule-based graph rewriting system, the following steps are taken. 1.

1) Analyze a performance of the system on the whole set $\mathfrak{F}$. Generate random superpositions from $\mathfrak{F}$ with fixed structural complexity $\phi(f)$ and simplify them by the system. Analyze the fraction of simplifiable superpositions in the generated set. The performance of the system is the average structural improvement after the simplifications.

$$\text{improvement} = \frac{\text{init}(\#\text{primitives} + \#\text{param.})}{\text{new}(\#\text{primitives} + \#\text{param.})}. \quad (10)$$

2) Analyze a performance of the system on superpositions generated by MVR. As the aim is to fit the dataset, the performance is the improvement of the error function $\mathcal{S}$. As $\mathcal{S}$ is heuristic quality estimation, Akaike information criterion is also analyzed. AIC stands for a theoretically justified estimation of the superposition quality with respect to both approximation and dimensionality of the parameter space.

$$\text{AIC} = 2k + N\left(\ln(N \cdot \text{MSE})\right),$$

where k is the dimensionality. The fraction of simplifiable superpositions is plotted and analyzed with one recieved from the previous item.

3) Analyze the impact of the system on the quality of the final superpositions selected by MVR. Launch MVR with and without the proposed graph rewriting system. Compare the final selected superpositions for both cases. As the aim of MVR is selection structurally simple and well fitted superposition, the comparison is conducted with respect to $\mathcal{S}$ and MSE on a test sample.

C. Experiment

Start with an analysis of the system on $\mathfrak{F}$. Generate random models from $\mathfrak{F}$ with fixed structural complexity and find the percentage of simplifiable models in the generated sets. For each structural complexity 10000 random superpositions are generated. The percentages strongly depend on the chosen structural complexities, see Table IV and Figure 6.

Table IV shows that complex superpositions are more inclined to be simplified. It follows from the notion that the probability of a rule presence increases as a superposition becomes more complex. However, the simplification is equally efficient for superpositions of all structural complexities, as the relative improvement (10) does not change. The values presented in Table IV depend strongly on the used sets of primitives and rules. For example, if one uses diverse rules, MVR will frequently generate simplifiable superpositions. On the other hand, the use of small number of rules does not afford to simplify generated superpositions frequently.

Now analyze the properties of the graph rewriting system applied to MVR. After the termination MVR selects 20 best superpositions with respect to $\mathcal{S}$. The parameters of $\mathcal{S}$ are set empirically, see Table VI. Here are the reasons followed in the parameters setting: the generated superpositions should be sufficiently simple to be analyzed and interpreted by experts, at the same time the superpositions should be sufficiently complex to fit the dependencies in the dataset. Varying these parameters, a desired trade-off was achieved.

TABLE V: Relative quality changes for simplified superpositions, %.

Criterion	unchanged	worsen	improved
$\mathcal{S}$	0	13	87
AIC	16	8	75

TABLE VI: Parameters of $\mathcal{S}$.

Parameter	λ	Φ	κ_1	κ_2
Value	0.000035	20	0.1	1

Launch the generation by MVR 30 times and search for the best superpositions to fit the dataset. Compare the simplified superpositions with the initial ones according to the error function $\mathcal{S}$ and Akaike information criterion.

Figure 7 shows the fraction of simplifiable superpositions among the generated till i-th iteration. All measured values are averaged over 50 launches. After 15-20 iterations MVR stagnates, stored superpositions are similar and the population almost stop changing. As selected superpositions are already simplified and not eliminated from the population, crossover and mutation are unlikely to construct many new simplifiable superpositions from them. Therefore the fraction value remains relatively low (10-13%) with respect to the random populations analyzed above. It remains constant in average, when MVR stagnates, see Figure 7. This fraction also significantly depends on the parameters of $\mathcal{S}$. For example if the penalty λ is chosen too big, the generated superpositions are mostly simple structured and not simplifiable. Otherwise, if complex superpositions are not penalyzed with λ, the population will consist of overfitted superpositions, which are likely simplifiable. The same logic is true for the threshhold Φ and penalties κ_1, κ_2.

Simplified superpositions are compared with the initial ones according to $\mathcal{S}$ and Akaike information criterion. Figure 10 shows the results. Outliners are deleted from the hists by 95% quantile. Most superpositions are improved by graph rewriting system according to the error function $\mathcal{S}$ and AIC information criterion. Nearly 64% of superpositions were improved with respect to both criteria. In average, the AIC was better on 0.2% and the error function $\mathcal{S}$ was better on 3% after the simplification. The table below lists the percentages of unchanged, improved and worsen superpositions with respect to $\mathcal{S}$ and AIC criterion.

Finally, compare the superpositions selected by the initial and modified versions of MVR. The modified version uses the proposed graph rewriting system and simplify generated superpositions. Two versions are compared based on the error function $\mathcal{S}$ and MSE on a test sample. These criteria are applied separetely to the top-1 and top-20 of selected superpositions. Launch the genetic algorithm 200 times and average the results over the launches, see Table VII. The use of graph rewriting system improves the genetic algorithm according to both MSE and $\mathcal{S}$.

TABLE IV: Percentage of simplifiable superpositions and average relative structural improvement.

Structural complexity	6	8	10	12	14	16	18	20
Percentage	0.21	0.28	0.34	0.42	0.46	0.51	0.54	0.58
Relative improvement (10)	1.04	1.04	1.04	1.04	1.04	1.04	1.04	1.04

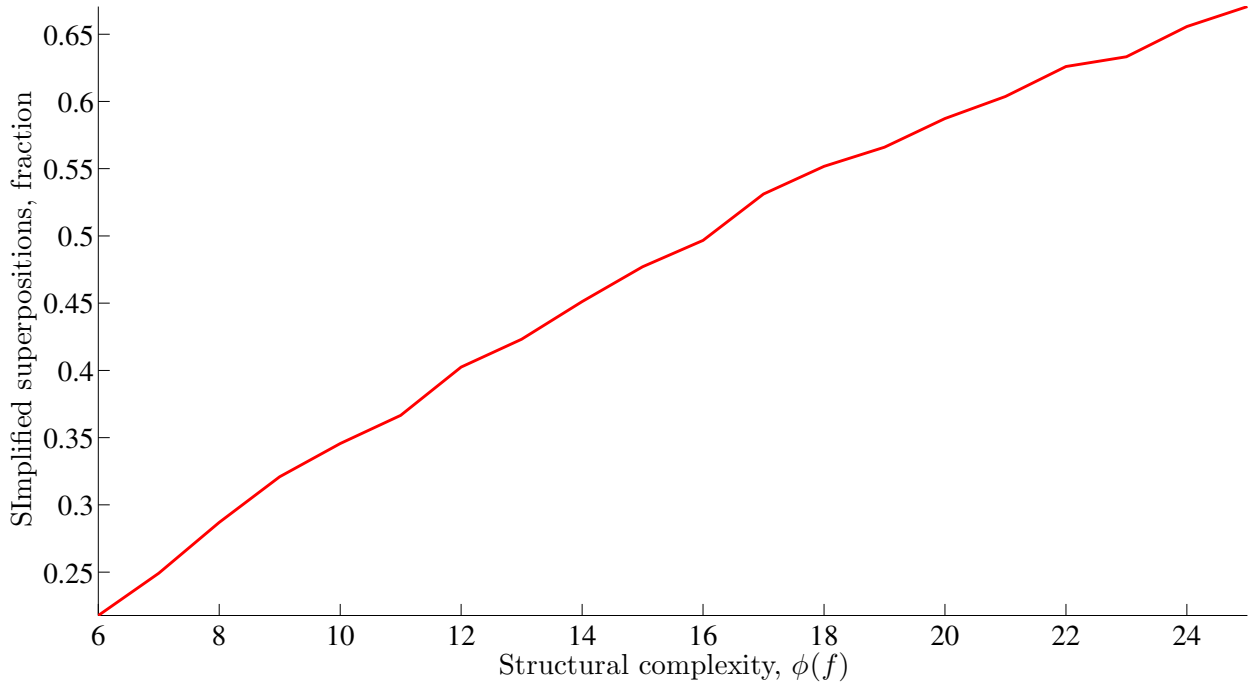


Fig. 6: Fraction of simplifiable random superpositions of fixed structural complexity.

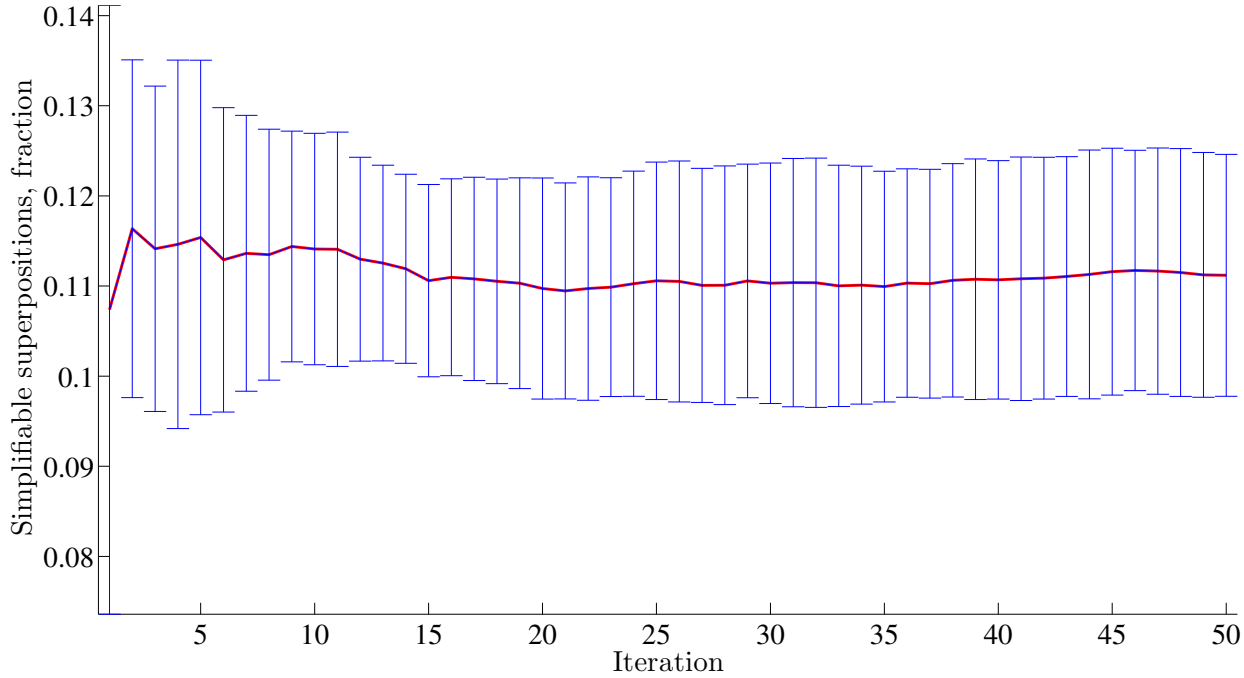


Fig. 7: Fraction of simplifiable superpositions in population generated by MVR.

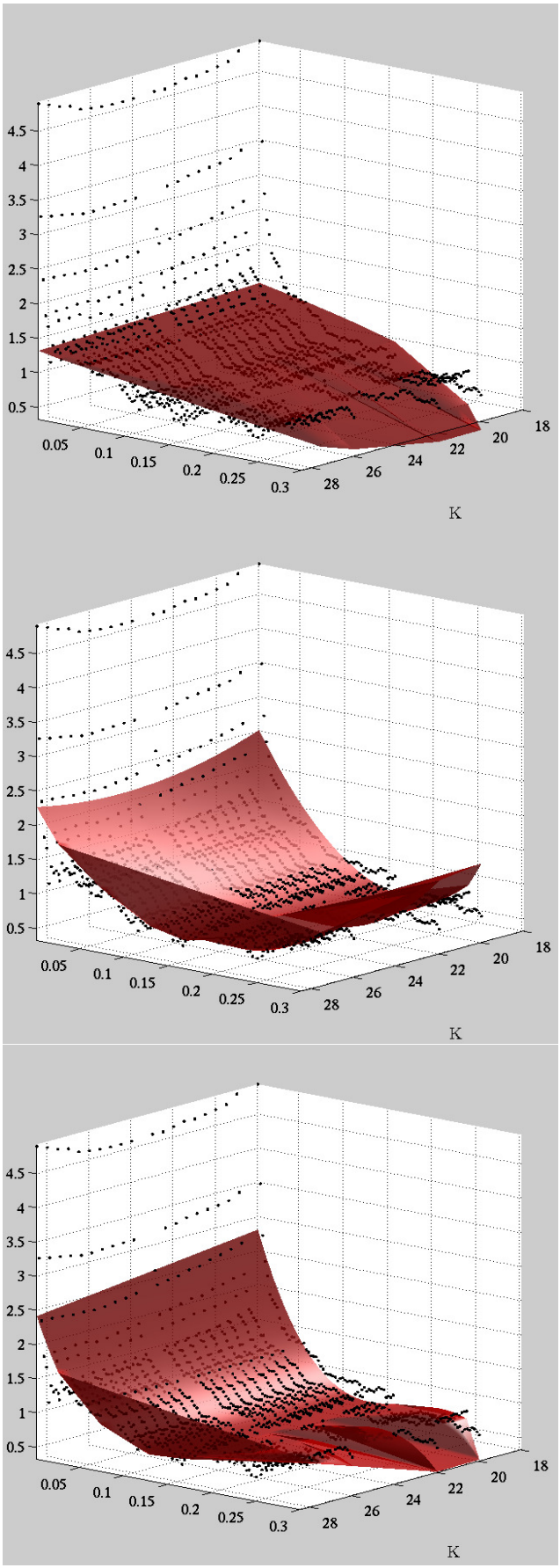


Fig. 8: Initial superpositions f_{1-3}.

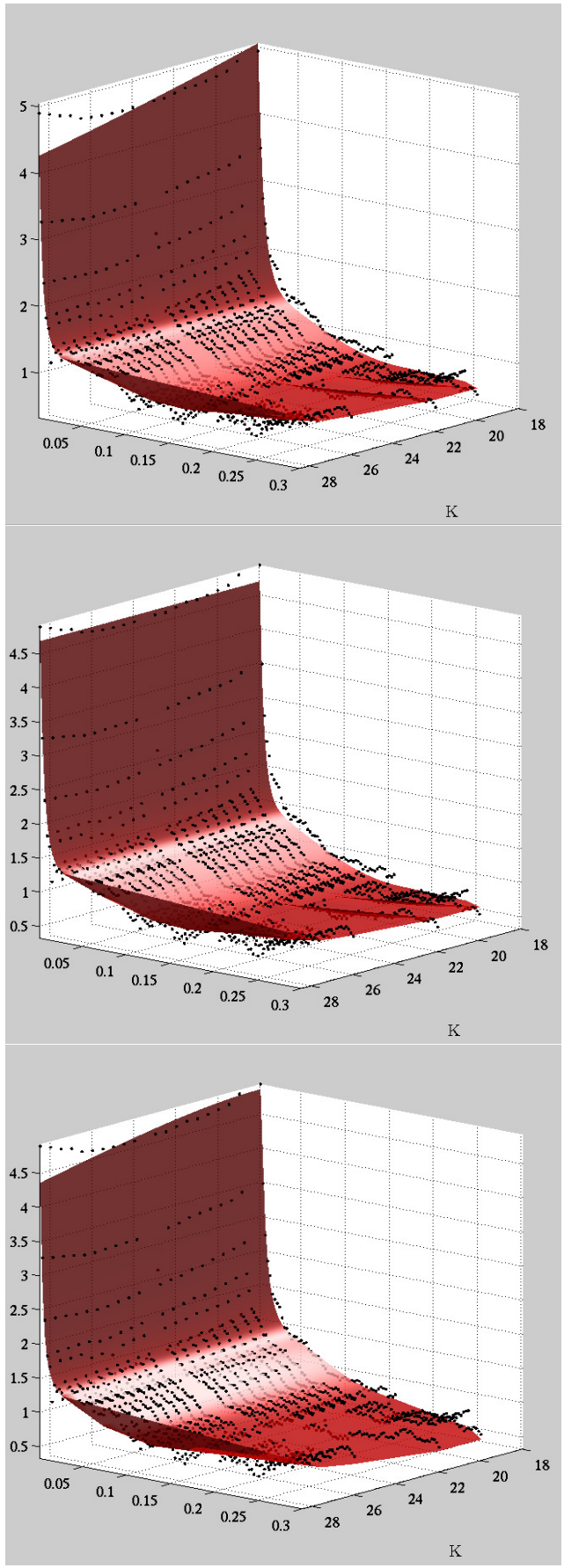


Fig. 9: Final selected superpositions f_{4-6}.

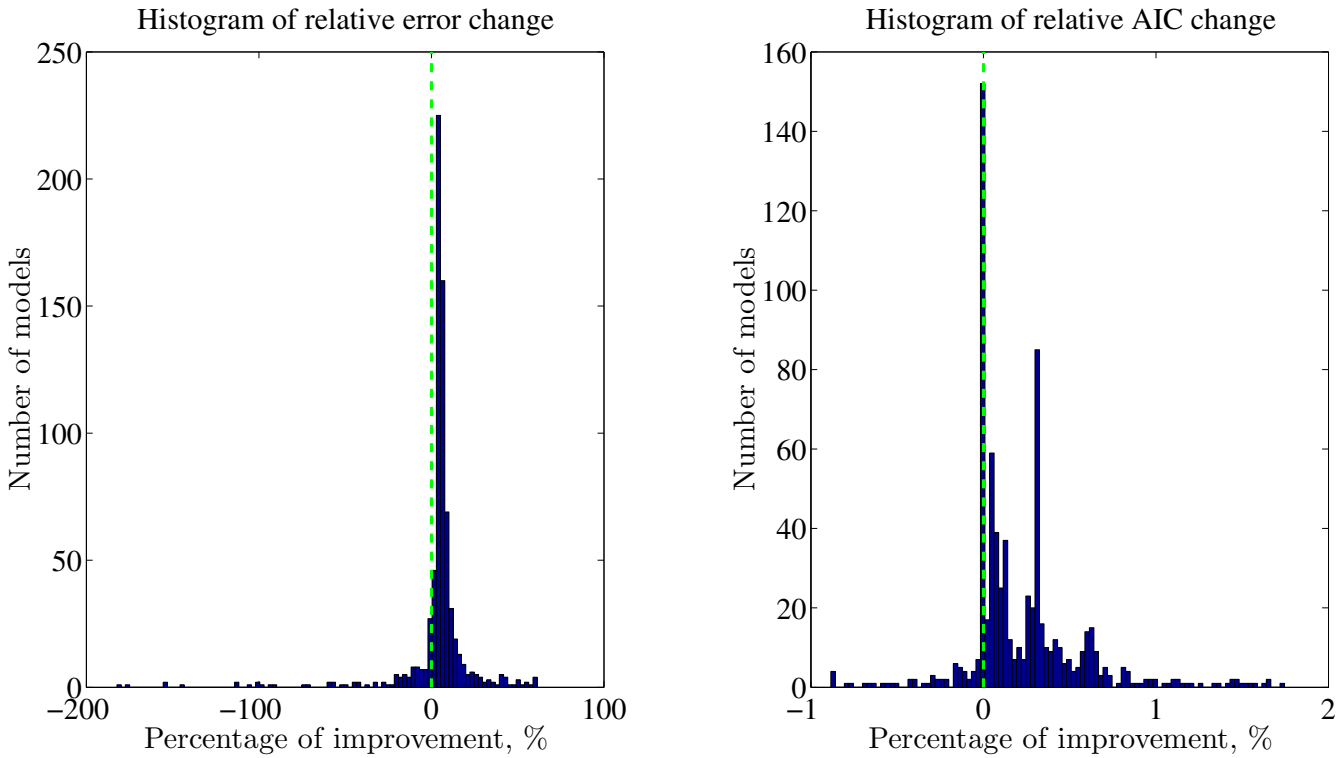


Fig. 10: Histograms of relative changes of $\mathcal{S}$ and AIC criterion after the simplification.

TABLE VIII: MSE on the train dataset, %.

Superposition	f_1	f_2	f_3	$\sigma(\mathbf{w})$ (2)	f_4	f_5	f_6
MSE	0.32	0.15	0.117	0.0173	0.0068	0.0072	0.0086

TABLE VII: Relative quality changes for the best selected superpositions, %.

MVR version	MSE (top-1)	$\mathcal{S}$ (top-1)	MSE (top-20)	$\mathcal{S}$ (top-20)
Initial	0.6	13.7	0.76	1.61
Modified	0.5	12.8	0.68	1.54

The initial superpositions of MVR evolves to better fitting superpositions. One of the launches of MVR produces the following functions, where P is a parabola.

$$f_4(\mathbf{w},\mathbf{x}) = P\left(\left(w_1 + w_2 e^{tK} + t\right) \cdot \sqrt{w_3 t + w_4}\right),$$

$$f_5(\mathbf{w},\mathbf{x}) = P\left(\left(w_1 + e^{w_1 t + t^2} + t\right) \cdot \sqrt{w_2 t + w_3}\right),$$

$$f_6(\mathbf{w},\mathbf{x}) = P\left(e^{tK}\right) \cdot P\left(\text{normal}\left(e^{tK}\right)\right).$$

In the plots above the left column stands for the initial superpositions f_{1-3} and the right for the final selected f_{4-6}. 50 iterations are sufficient for MVR to build a well fitted function. Table VIII shows the improvement in the approximation after the evolving of f_{1-3}. The selected superpositions outperforms the expert-given function $\sigma(\mathbf{w})$ (2) as well.

To reproduce the represented results, download the project code from the reference.

VI. Conclusion

A procedure of a superposition simplification is proposed. This procedure is implemented as a rule-based graph rewriting system, which uses an algorithm of detection isomorphic subtrees in a tree. The procedure reduces both structural complexity of a superposition and dimensionality of its parameter space. Computational complexity of the procedure is small enough to use it in the genetic programming to generate superpositions. After the simplification the quality of superpositions mostly increases according to the AIC criterion and the error function. The modified version of MVR generates better superpositions according to the MSE on the test sample and the error function.

References

[1] D. John and J. Engle-Warnick, "Using symbolic segression to infer strategies from experimental data," *Evolutionary Computation in Economics and Finance*, vol. 100, pp. 61–84, 2002.

[2] C. Sammut and G. I. Geoffrey, "Symbolic regression." in *Encyclopedia of Machine Learning*. Springer, 2010, p. 954.

[3] V. V. Strijov and G. W. Weber, "Nonlinear regression model generation using hyperparameter optimization," *Computers and Mathematics with Applications*, vol. 60, no. 4, pp. 981–988, 2010, pCO' 2010 - Gold Coast, Australia 2-4th December 2010, 3rd Global Conference on Power Control Optimization.

[4] V. V. Strijov, *Methods of Inductive generation of regression models*. .: , 2008.

[5] I. Zelinka, Z. Oplatkova, and L. Nolle, "Analytic programming - symbolic regression by means of arbitrary evolutionary algorithms," *International Journal of Simulation Systems, Science & Technology*, vol. 6, no. 9, pp. 44–56, Aug. 2005, special Issue on: Intelligent Systems.
[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
[7] G. Rudoy and V. Strijov, "Algorithms for inductive generation of superpositions for approximation of experimental data," *Informatics and applications*, vol. 7(1), pp. 17–26., 2013.
[8] 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. 587–602, May 2004.
[9] G. Paris, D. Robilliard, and C. Fonlupt, "Exploring overfitting in genetic programming," in *Artificial Evolution, 6th International Conference, Evolution Artificielle, EA 2003, Marseilles, France, October 27-30, 2003*, 2003, pp. 267–277.
[10] L. Vanneschi, M. Castelli, and S. Silva, "Measuring bloat, overfitting and functional complexity in genetic programming," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 877–884.
[11] L. A. Becker and M. Seshadri, "Comprehensibility and overfitting avoidance in genetic programming for technical trading rules," Worcester Polytechnic Institute, Tech. Rep., May 2003.
[12] I. Gonçalves, S. Silva, J. Melo, and J. Carreiras, "Random sampling technique for overfitting control in genetic programming," in *15th European Conference on Genetic Programming (EuroGP 2012)*, n/a, 2012.
[13] I. Gonçalves and S. Silva, "Balancing learning and overfitting in genetic programming with interleaved sampling of training data," in *16th European Conference on Genetic Programming (EuroGP 2013)*, n/a, 2013.
[14] W. B. Langdon and W. B. Langdon, "Minimising testing in genetic programming," 2011.
[15] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation*. Springer, Berlin, 2006.
[16] P. P. Pérez, "Matrix graph grammars as a model of computation," *CoRR*, vol. abs/0905.1202, 2009.
[17] J. B. Kadane and N. A. Lazar, "Methods and criteria for model selection," *Journal of the American Statistical Association*, vol. 99, pp. 279–290, 2004.
[18] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1974.
[19] B. Laszlo, "Graph isomorphism in quasipolynomial time," *CoRR*, vol. abs/1512.03547, 2015.
[20] R. Shamir and D. Tsur, "Faster subtree isomorphism." *J. Algorithms*, vol. 33, no. 2, pp. 267–280, 1999.
[21] G. I. Rudoy and V. V. Strijov, "Simplification of superpositions of primitive functions with graph rule-rewriting," in *Intellectual Information Processing. Conference proceedings.*, 2012, pp. 140–143.
[22] M. Composer. https://sourceforge.net/projects/mvr/.
[23] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. J. Appl. Maths.*, vol. II, no. 2, pp. 164–168, 1944.
[24] J. C. Hull, *Options, Futures, and Other Derivatives with Derivagem CD (7th Edition)*, 7th ed. Prentice Hall, May 2008.
[25] V. V. Strijov and R. A. Sologub, "The inductive generation of the volatility smile models," Ph.D. dissertation, Journal of Computational Technologies, 2009.
[26] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*. Springer, Berlin, 2008.

Kulunchakov Andrei received the B.S. degree in applied mathematics and physics from Moscow Institute of Physics and Technology and Department of Intelligent Systems, where he is currently pursuing the M.S. degree. His research interests include genetic programming, structural simplification, ranking functions and time series classification/clusterization.

Strijov Vadim obtained Ph.D. in Physics and Mathematics in 2002 with a thesis on Mathematical Modelling and Data Analysis and D.Sc. in Theoretical Foundations of Computer Sciences in 2014 from the Computing Centre of the Russian Academy of Sciences. His research interests include machine learning, data analysis, model selection, structure learning, preference learning.

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 improves a genetic algorithm for generating simple structured IR ranking functions. These functions are constructed as superpositions of expert-given primitive functions. This paper solves the problems of overfitting and evolutionary stagnation, specific for genetic algorithms. To solve the problem of overfitting it uses a regularizer controlling the structural complexity of generated superpositions. Evolutionary stagnation is detected with a structural metric on superpositions. Adding new random superpositions devirsifies a stagnating population. The quality of a ranking function is evaluated by mean average precision (MAP). The computational experiment is conducted on TREC5-8. The best selected superpositions have uniformly higher quality on TRECs than classical IR models and the ones selected by an exhaustive algorithm.

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

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* and for each pair returns relevance of the *doc-*

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

ument to the *query*. IR ranking models in [2] use two basic features of such pairs: term frequency (tf) and document frequency (idf). In this paper ranking models are 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. This approach allows 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 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 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.

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.

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 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

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

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.

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.

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

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

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;

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

4)

$$\alpha(\mathfrak{M})=\nu\bigg([\mu(f,f')\leq\alpha_1]\Rightarrow[\eta(f,f')\leq\alpha_2]\big|f,f'\in\mathfrak{M}\bigg)\geq1-\varepsilon,\qquad(3)$$

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

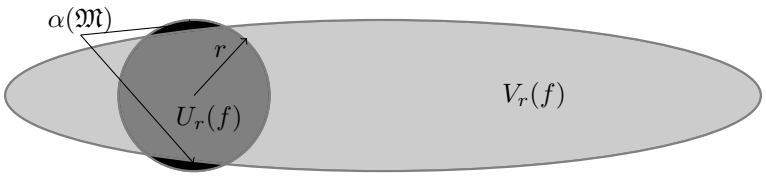


Figure 1: Illustration of supposed relation between $U_r(f)$ and $V_r(f)$.

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 $U_r(f_{\text{opt}})$ 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 $U_r(f_{\text{opt}})$, then they will rarely leave $U_r(f_{\text{opt}})$ on the next iterations, since crossovers produce superpositions mainly from $U_r(f_{\text{opt}})$ 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.

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.

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 $\mathfrak{F}$.

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

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.

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

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

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

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]. 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.

Two trees $T_i,\ T_j$ have a common subtree T if each of them has a subtree isomorphic to T.

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

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.

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.

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.

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;

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, $\mathrm{LM_{DIR}}$. Therefore, if the modified genetic algorithm succeeds in outperforming functions from $\mathfrak{F}_0$, it will also outperform BM25, LGD, $\mathrm{LM_{DIR}}$ 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 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

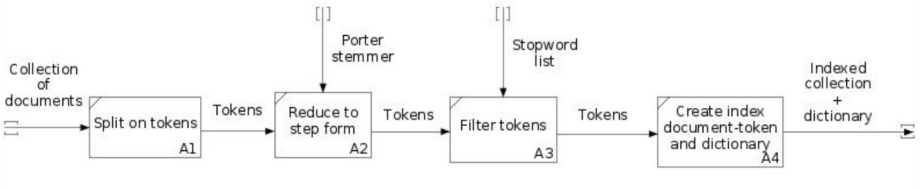


Figure 2: Scheme of data preprocessing steps.

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.
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.

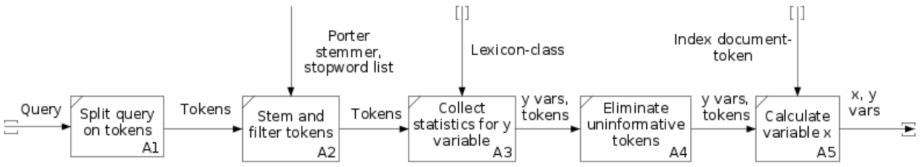


Figure 3: Scheme of query processing steps.

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 1 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 1 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.

But 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.

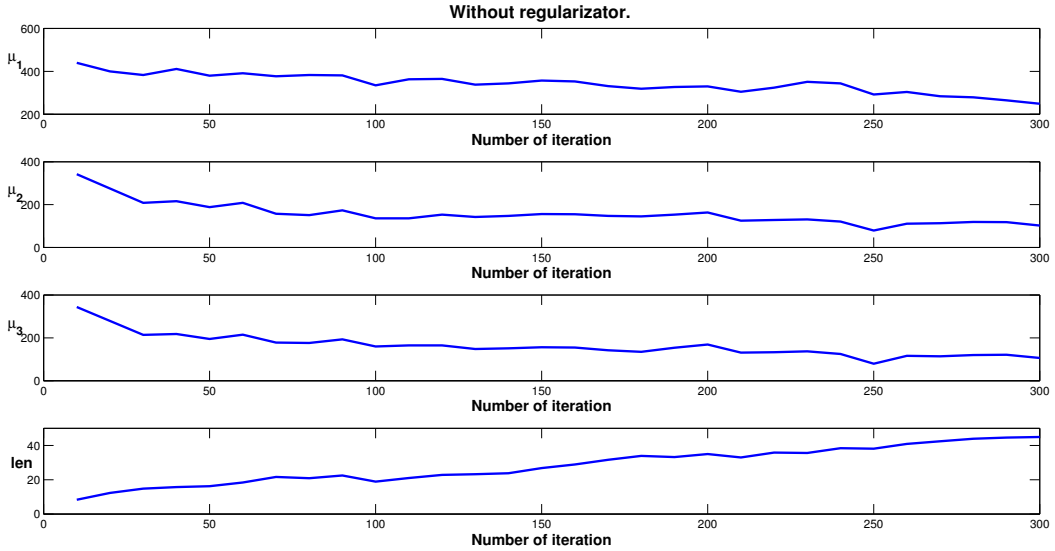


Figure 4: Dynamics of $d(\mathfrak{M})$ and l_a when no regularizer is used.

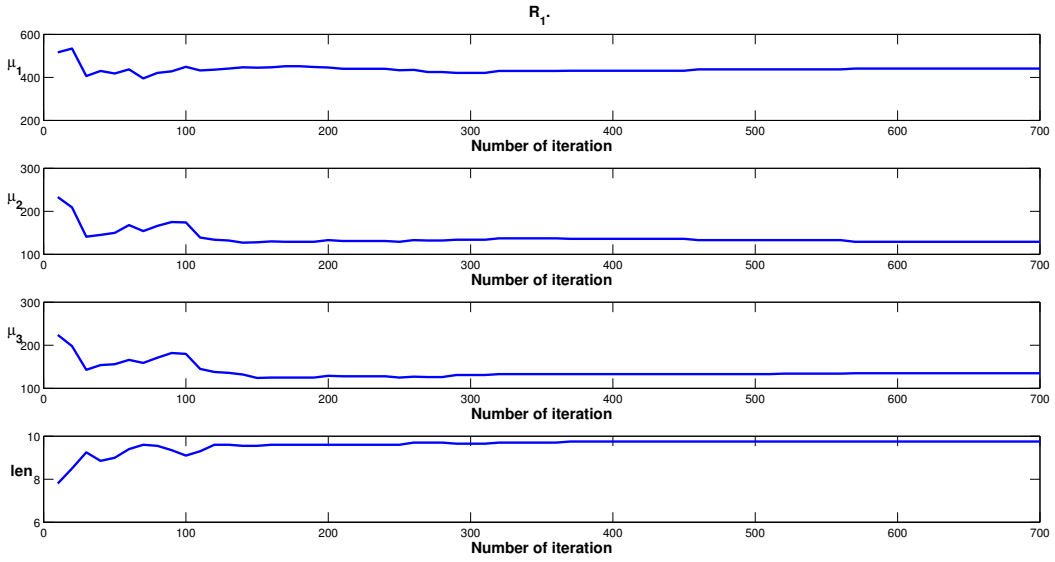


Figure 5: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_1 is used.

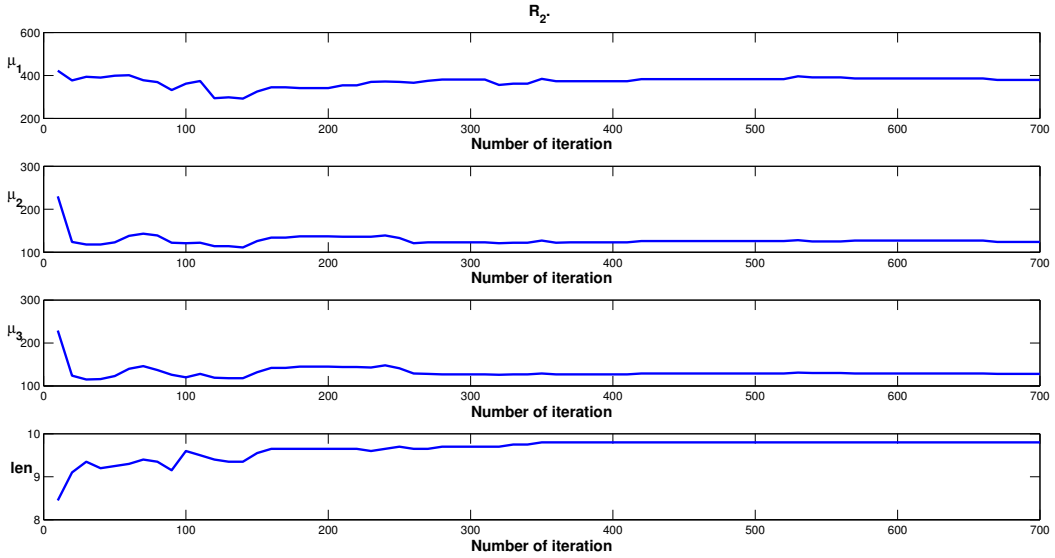


Figure 6: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_2 is used.

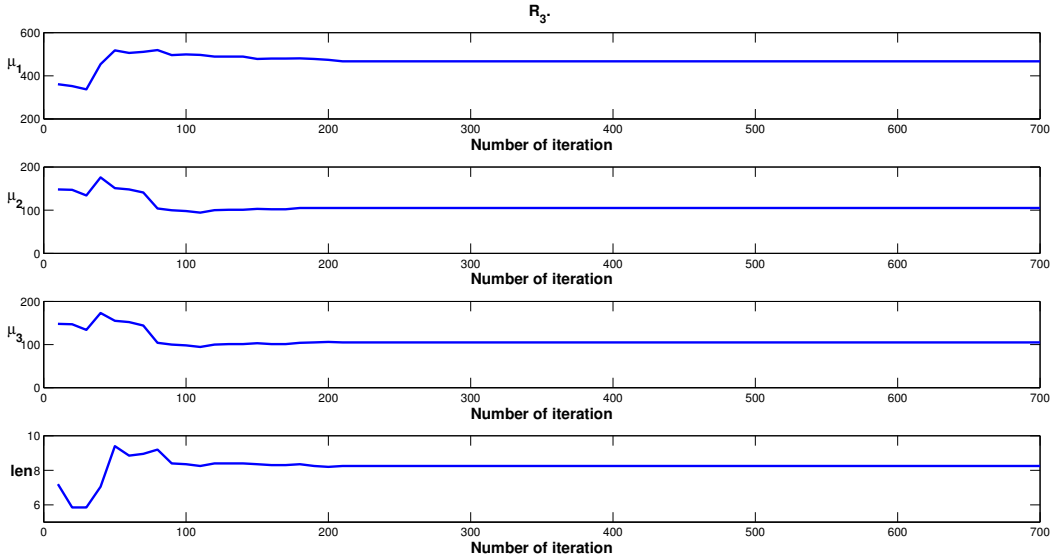


Figure 7: Dynamics of $d(\mathfrak{M})$ and l_a when the regularizer R_3 is used.

regularizer	Metric μ_1	Metric μ_2	Metric 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 1: Comparison of CPU time required by structural metrics

Now 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 1. These reasons lead to elimination of μ_1 from the further analysis.

The 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 1.

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 stagnation. 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 $g(x) = \ln\ln(x)$):

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 on the superpositions $\{h_j\}$ and $\{f_i\}$ are presented in Table 2. 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.

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 solves IR problem by generating a proper ranking function, which estimates relevance of documents to queries. This function is generated by a

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

genetic algorithm. However, its basic version is inclined to generate overfit functions. To avoid overfitting, one must control their structural complexity and solve an evolutionary stagnation problem. Its 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, $\mathrm{LM_{DIR}}$. and the ones selected by an

exhaustive approach.

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.