Optimization of regular expressions using competitive coevolutionary algorithm based on symbolic regression

Liliya Demidova*, and *Nikita* Moroshkin

Institute for Information Technologies, Federal State Budget Educational Institution of Higher Education «MIREA – Russian Technological University», Vernadsky avenue, 78, Moscow, 119454, Russia

Abstract. The paper presents an algorithm for optimizing the structure of regular expressions of the Python programming language dialect of the re module. The optimization algorithm is implemented as a competitive coevolution algorithm based on the symbolic regression algorithm (the Gene Expression Programming algorithm will be used as an implementation of the symbolic regression algorithm). The paper proposes a pseudocode for the regular expression optimization algorithm as an abstract "black box" model, provides hyperparameters of competitive coevolution, as well as a function for assessing the suitability and reliability of individual algorithms within the coevolution. A comparative analysis of the results of running the GEP algorithms as part of coevolution and separately demonstrates the effectiveness of using coevolution as a method for optimizing regular expressions.

1 Introduction

Currently, almost all areas of human activity are somehow related to the field of data analysis. One way or another, but the development of machine learning models, including large language models, in the areas of text information analysis does not lead to a decrease in the interest of specialists in using classical algorithms for processing text data.

One of the most common and important tasks of text information analysis is the problem of finding a substring in the original string. One of the variations of this problem is the problem of finding a substring by a certain pattern. To find a substring by a certain pattern, regular expressions (RE) are most often used [1]. RE is a tool for searching words in a text corpus by a certain pattern. Each RE instance defines a finite automaton (FA), most often this automaton is a deterministic or non-deterministic finite automaton (DFA and NFA, respectively) [2]. In the case of working with DFA, RE have limited functionality [2], but such implementations [3] are the most effective and are used in processing a large text corpus to search for patterns that are simple in structure [4]. In the case of working with NFA, extended functionality is added to the implementation of the RE mechanism, for example,

* Corresponding author: demidova.liliya@gmail.com

search by pattern with the resolution of the error tolerance in the FA [4] or unlimited boundaries.

Optimization of the RE structure in order to increase the efficiency of the implemented FA is the process of minimizing vertices in a given FA. If the original RE has an implementation in the NFA, the Thompson algorithm [3] is applied. This algorithm implements the transformation of the NFA into an equivalent DFA. Despite the existing methods of RE optimization, not all syntaxes similar to RE can be optimized by the standard method of FA minimization. The spread of low-code platforms and the development of the text analysis field influenced the emergence of new syntaxes of RE and similar ones, which are not always implemented in some FA [4, 5]. In this case, the known algorithms for RE optimization cannot be applied due to the unknown implementation of the RE mechanism. In fact, under these conditions, RE is an abstract model of a "black box" [6].

The algorithm proposed in this paper is proposed to be used to optimize regular expressions for stream processing tasks. Since PEs are often used as a simple filter to reduce data flow (often done to reduce the load on subsequent more complex algorithms in terms of resource consumption), RE optimization can yield significant resource savings. Running this algorithm to optimize REs for end-to-end tasks (e.g., editing some file) does not make sense, since it is likely to take less time to process a PE than it does to run the optimization algorithm. It is recommended to run the optimization algorithm before the start of the stream handler.

In this case, researchers propose to use stochastic optimization algorithms, namely population algorithms. In [7], the efficiency of using the differential evolution algorithm for RE optimization was demonstrated, however, when representing RE as an incidence list of some abstract syntax tree, it is necessary to develop some translator from the source dialect to the target one. Also, the results of the PE optimization require improvement, since the approach of representing some PE as an incidence list provokes the algorithm to create a large number of invalid individuals. The considered conditions of the "black box" model as a mathematical model of RE do not allow building a translator to some target dialect. In this paper, it is proposed to use an expression tree based on the specified functions and terminals of the source RE as an intermediate representation of RE. It is proposed to use the symbolic regression algorithm, designed to optimize the structure of tree expressions, as an optimization algorithm. In the case of solving the RE optimization problem using symbolic regression, it is necessary to solve the problem of selecting the symbolic regression parameters. In some cases, multiple runs are impossible due to the high cost of calculating the fitness function for all the algorithms under consideration, the lack of time to restart the algorithms, and so on. In this case, coevolutionary optimization algorithms can be recommended for use [8]. In this paper, a coevolutionary optimization algorithm for RE be presented using the symbolic regression algorithm as individual optimization algorithms.

2 Materials and methods

In general, the problem solved by population algorithms can be represented as:

$$x^* = \underset{x \in \mathbb{R}^z}{arg\ min}\big(Q(x)\big), \tag{1}$$

where x is a vector belonging to the search space $\mathbb{R}^z$ and describing an individual; z is a dimension of vector x; $\mathbb{R}^z$ is a range of admissible values of a vector; $Q(x)$ is a objective function that determines the value of the individual's quality indicator; x^* is a vector of optimal solution.

In the context of solving a regular expression optimization problem, a solution vector or an individual in a population is represented as an expression tree (ET). Symbolic regression

is a type of regression analysis [9] that considers the ET space to find a specific ET that best fits a given data set, i.e., has the best fitness score under given conditions. Symbolic regression is most often used to optimize mathematical expressions that are presented in a hierarchical structure, i.e., as some ET. This paper will consider the genetic expression programming (GEP) algorithm [9], which is a representative of a family of population algorithms that solve the symbolic regression problem.

Let reg_0 be some initial RE. Let t be some set of terminals $t_1, t_2 \dots, t_r$, where r is the number of terminals used in the original RE reg_0. Let f be set of functions that accept a set of terminals t as parameters, $f_1(t), f_2(t) \dots, f_k(t)$, where k is the number of functions used in the original RE reg_0. For example, for the RE «a(b|c)?|d*», the set of terminals is $t = (a, b, c, d)$, the set of functions is $f = (|, ?, *)$. Also, each function $f_i \subset f$ is characterized by the arity parameter ε, which determines the maximum possible number of terminals as parameters for the function under consideration.

Then the ET for the original RE reg_0 will be of the form $tree_0(f, t \mid reg_0)$. When working with the symbolic regression problem and the genetic expression programming algorithm, it is necessary to determine the phenotype and genotype of the individual. The original form of the RE is shown in Figure 1a. The genotype of the individual is the K-expression [9] (shown in Figure 1b). The phenotype is the ET described above (shown in Figure 1c).

The gene expression programming (GEP) algorithm is specified by several hyperparameters [9]: the population size N^{pop}, the maximum tree depth d, the length of genes in the head part of the genotype h and the length of genes in the tail part of the genotype t. The length of the tail part of the genotype can be obtained using the following formula:

$$t = h \cdot (\varepsilon_{max} - 1) + 1, \tag{2}$$

where h is the length of gene in the head part of the genotype, ε_{max} is a maximum arity of a set of functions f. To evaluate the fitness of an individual in the optimization algorithm, it is proposed to define two quality metrics, such as the accuracy of the RE Acc and the efficiency of the RE $Perf$.

To calculate the values of the described metrics, it is necessary to generate a certain number of test strings. Let S be the set of generated strings that are used to check the RE for correctness and to calculate quality metrics. Let A be the alphabet whose symbols make up the set of generated strings S, and S_g be the gth string from the set of test strings S ($g = \overline{1, G}$).

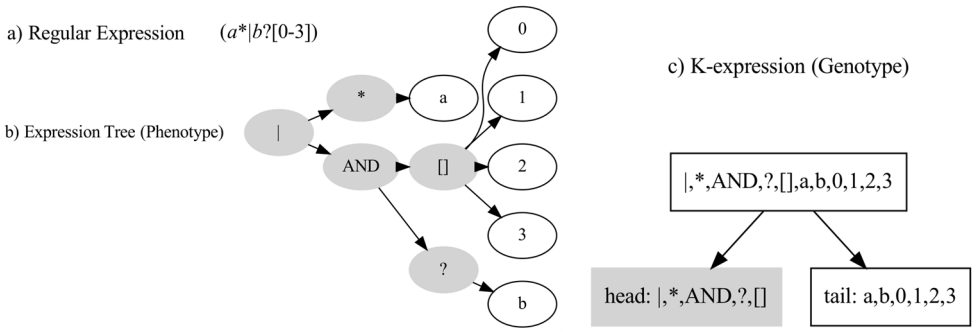

Fig. 1. Example of intermediate representations: a) type of the original regular expression; b) expression tree (phenotype); c) K-expression (genotype).

Then the accuracy of some RE reg can be defined as:

$$Acc(reg, S) = \frac{1}{G} \sum_{g=1}^{G} m(reg, S_g), \tag{3}$$

where S_g is the gth string from a set of generated strings S ($g = \overline{1,G}$); G is the number of generated strings on which the correctness of the regular expression is checked; m is the processing function for string S_g through an RE automaton reg, returns 1 if a RE reg completely matches some section in the string S_g and returns 0 otherwise.

It is proposed to define the metric of the efficiency of the RE reg as:

$$Perf(reg,S) = \frac{1}{N^{iter}}\sum_{n=1}^{N^{iter}}\frac{1}{G}\sum_{g=1}^{G} p(m(reg,S_g)), \quad (4)$$

where S_g is the gth string from a set of generated strings S ($g = \overline{1,G}$); G is the number of generated strings on which the correctness of the regular expression is checked; m is the processing function for string S_g through an RE automaton reg, returns 1 if a RE reg completely matches some section in the string S_g and returns 0 otherwise, p is the function of measuring the time of the transmitted operation; N^{iter} is the number of specified iterations.

The constant N^{iter} is specified when designing the program. This constant is necessary for accurate measurement of the regular expression execution time, since the time spent may differ with different launches of the regular expression. The reasons for such differences are usually in the hardware of the computer under consideration. Let the area of reasons for the current example be limited to the operation of programming languages and the use of available hardware resources, namely RAM. In some cases, garbage collectors of the programming languages used can affect the execution time of the RE under consideration. To minimize the influence of third-party processes, the measurement should be performed a specified number of times, calculating the arithmetic mean execution time of some RE or the median, to avoid taking into account outliers in the execution time measurements. The time measurement function $p(\partial)$ measures the execution time for some operation. It is necessary to choose the $p(\partial)$ in such a way that it is least dependent on third-party processes and takes into account the specifics of the program operation in the programming language used.

The objective function $Q(x)$ of population RE optimizing algorithm, which is used to solve problem (1) can be represented as:

$$Q(x) = \begin{cases} \frac{(Perf(greg(x),S))}{Acc(greg,S)}, & Acc(greg,S) > 0 \\ stub, & Acc(greg,S) = 0 \end{cases}, \quad (5)$$

where x is a list describing an individual, i.e. some ET; $Perf$ is a function for calculating the speed of the regular expression (4); $greg$ is the function that returns the RE for the input ET x; S s the set of generated strings; Acc is the metric of the accuracy of the RE (2) when calculating the objective function; $stub$ is the constant set by the researcher for cases when the obtained RE has $Acc = 0$, this value is not taken into account when calculating statistics.

The coevolutionary algorithm is proposed to be implemented as a competitive coevolution based on the comparison of individual algorithms by the quality of each iteration (generation). The main parameters of the coevolutionary algorithm [8] can be considered the common resource $R \subset \mathbb{N}$, the length of the adaptation interval $T \subset \mathbb{N}$, the size of the "social card" M (the percentage of the initial population size, which allows determining the minimum population size that must be preserved during its correction in the process of coevolution) and the size of the penalty for the losing population $P \subset \mathbb{R}$ (the percentage of individuals that must be removed from the population).

The quality of the i −th population can be determined by the following formula:

$$q_i = \sum_{j=0}^{T-1}\left(\frac{T-j}{j+1}\right) \cdot b_i(j), \quad (6)$$

where T is the length of adaptation interval; $b_i = 1$, if at the jth iteration (in the j-м generation) the ith population has the best individual, otherwise $b_i = 0$.

The quality of the population is determined only if the iteration (generation) number $j > T$.

It should be noted that the launch of both individual algorithms and the coevolutionary algorithm is stopped only when the total available resource R. The pseudocode of the proposed coevolutionary optimization algorithm is shown in Figure 2.

Input: reg_0 – original RE, S – set of test strings for the original RE, $R \subset \mathbb{N}$ – shared resource, V_i – populations, $T \subset \mathbb{N}$ – adaptation interval, $I \subset \mathbb{N}$ – number of individual algorithms, $P \subset \mathbb{R}$ – percent of penalty for population, $M \subset \mathbb{R}$ – percent for "social card".

set the number of fitness calculating $\delta = 0$
for each algorithm number i from the interval $[1, I]$ do:
  initialize population V_i with random individuals
  for each individual x_j from the population V_i do:
    calculate the fitness value Q_i and $\delta = \delta + 1$
  end of cycle
end of cycle
while $i < T$ do:
  for each population V_i do:
    select the individuals from V_i
    recombine individuals from V_i
    mutate individuals from V_i
    for each individual x_j from the population V_i do:
      calculate the fitness value Q_i and $\delta = \delta + 1$
    end of cycle
  end of cycle
  while $\delta < R$ do:
    for each population V_i do:
      select the individuals from V_i
      recombine individuals from V_i
      mutate individuals from V_i
      for each individual x_j from the population V_i do:
        calculate the fitness value Q_i and $\delta = \delta + 1$
      end of cycle
    select survivors from V_i with penalty P and social card M
    end of cycle
  end of cycle

Fig. 2. Pseudocode of proposed coevolutionary algorithm.

3 Experimental Study

The proposed regular expression optimization algorithm was tested using the Python programming language in the PyCharm development environment. The experiments were performed on a computer with the following specifications: MacBook Air 13 2020 A2337 (CPU: Apple M1 3.2 GHz 5 nm, ARMv8.5-A, 3.2 GHz, 8 cores; RAM: 8 GB; 64-bit operating system).

To evaluate the efficiency of the proposed coevolutionary algorithm, the RE «*((((a|bb?)|(a|b\a?)|([0-9]|.ab))|(cd|dc|ce|de).ab)|(dc))?*» was formed. For the proposed expression, the set of terminals $t = (a, b, c, d, ., [0-9])$, the set of functions $f = (|, ?, \backslash)$. In addition, the function is a grouping of RE constructions, that is, a logical "AND". The construction "[0-9]" is presented in the experiments as a non-expandable group, that is, it is a terminal.

The parameters of the GEP algorithm are presented in Table 1. For all individual algorithms, the parameters were not changed.

Table 1. Launch parameters for the GEP algorithm.

Maximum tree depth	Chromosome length	Number of individuals in a population	Number of "elite" individuals
3	5	10	1

The parameters of the coevolutionary algorithm are presented in Table 2. The coevolutionary algorithm was launched with the number of individual algorithms equal to 3, 4, 5 and 6. Each launch was performed 20 times, and the arithmetic mean value of the individual fitness metric was recorded (5).

Table 2. Launch parameters for the coevolutionary algorithm.

Shared resource size	"Social card" size	Penalty size	Adaptation interval length
400	20%	10%	5

The constant $N^{iter} = 100$. This constant defines how many times the measurement of the running time of a certain RE will be launched. The measurement of the running time of the RE was carried out using the *timeit* library of the Python programming language; during the measurement, the Timer module was used, which disables the garbage collector of the programming language itself, thereby reducing the impact of external processes on the measurement of the running time of the RE.

In each experiment, in addition to running coevolution, each individual GEP algorithm was run separately. After obtaining the results, the samples of minimum values obtained for one population of all individual algorithms were compared with the best algorithm. The samples were compared using the nonparametric Wilcoxon statistical test [10]. The obtained results are presented in Figures 3, 4, 5 and 6 (for 3, 4, 5 and 6 individual algorithms, respectively). All figures show graphs of the minimum value of the individual fitness function (5) at each iteration of the algorithm.

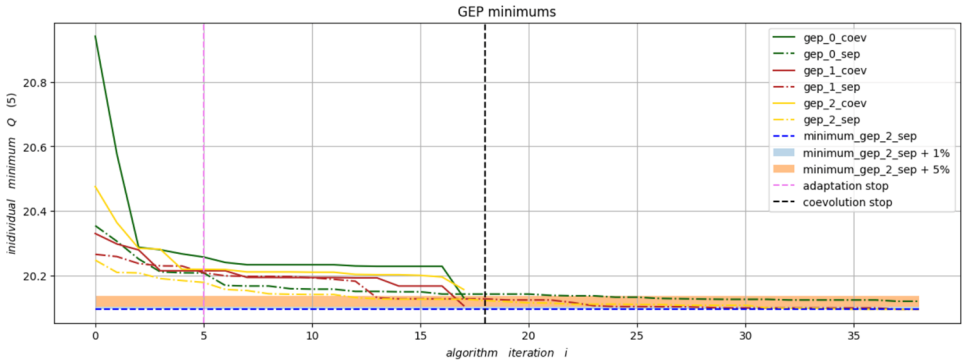

Fig. 3. The plot of the average minimum fitness function value (5) of the GEP algorithms for experiment №1 (3 individual algorithms).

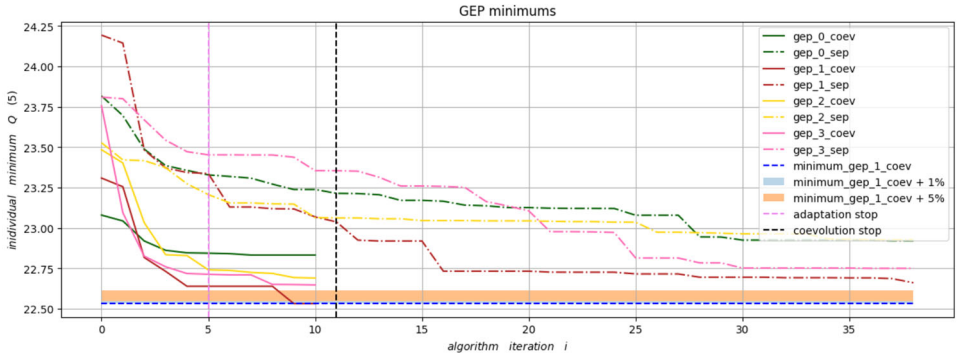

Fig. 4. The plot of the average minimum fitness function value (5) of the GEP algorithms for experiment №2 (4 individual algorithms).

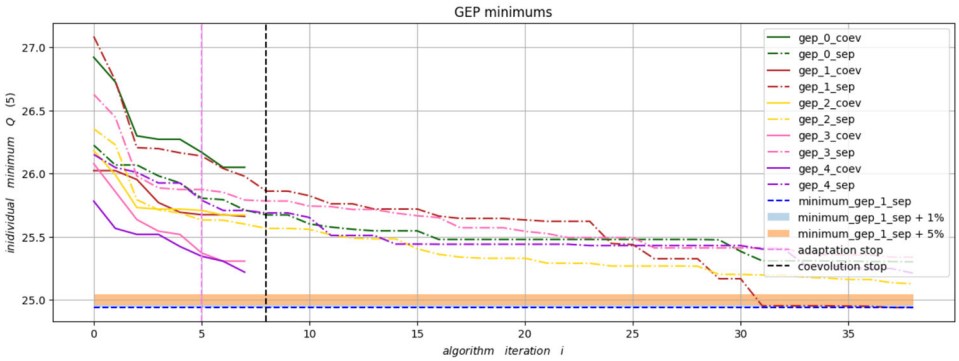

Fig. 5. The plot of the average minimum fitness function value (5) of the GEP algorithms for experiment №3 (5 individual algorithms).

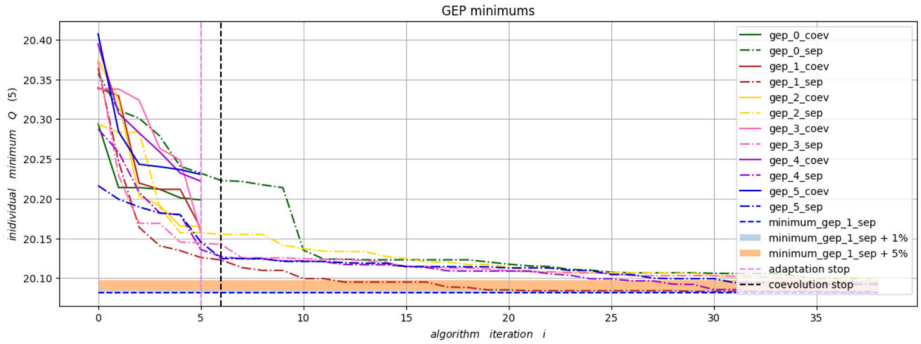

Fig. 6. The plot of the average minimum fitness function value (5) of the GEP algorithms for experiment №4 (6 individual algorithms).

The x-axis shows the iteration (generation) number of the algorithm, and the y-axis shows the value of the fitness function of the individual (5) multiplied by 10^6 for better readability. The graphs immediately display the indicators for pairs of algorithms: if the algorithm is run as part of the coevolution algorithm, the *_coev* prefix is used; if the algorithm is run individually, the *_sep* prefix is used. A separate blue dotted line shows the value of Q_{min}, which is the minimum determined by comparing all measurements of the fitness function of the individual (5) for all algorithms over all iterations. In addition, the value of Q_{max},which is the maximum value of the fitness function (5), is defined in a similar way.

For a better understanding of the convergence of the algorithms to the minimum Q_{min} it is useful to consider the values closest to the minimum Q_{min}. Let $\Delta_1 = (Q_{max} - Q_{min})/100$, and $\Delta_2 = 5 \cdot (Q_{max} - Q_{min})/100$. The the following intervals can be defined: $[Q_{min}, Q_{min} + \Delta_1)$ and $[Q_{min} + \Delta_1, Q_{min} + \Delta_2)$. These intervals are shown in Figures 2–6 by filling them with blue and pink colors, respectively.

It is advisable to estimate the number of individuals whose values fall within the above-mentioned intervals. Let E_1^i, E_2^i and E_3^i be the number of individuals of the ith algorithm which fitness function (5) values fall within the intervals $[Q_{min}, Q_{min} + \Delta_1)$, $[Q_{min} + \Delta_1, Q_{min} + \Delta_2)$ and $[Q_{min} + \Delta_2, +\infty)$ respectively. An analysis of the values of E_1^i, E_2^i and E_3^i will allow us to determine the convergence to the minimum of Q_{min} found during each experiment. Table 3 shows the values of E_1^i, E_2^i and E_3^i for individual algorithms (prefix sep) and algorithms launched in coevolution (prefix coev).

Table 3. Measurements of the values E_1^i, E_2^i and E_3^i.

Experiment number	Metric	***i*th algorithm**					
		$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
1	$E_{1,coev}^i$	0	0	0	-	-	-
	$E_{2,coev}^i$	1	1	0	-	-	-
	$E_{3,coev}^i$	17	17	18	-	-	-
	$E_{1,sep}^i$	0	15	8	-	-	-
	$E_{2,sep}^i$	17	11	19	-	-	-
	$E_{3,sep}^i$	22	13	12	-	-	-
2	$E_{1,coev}^i$	0	2	0	0	-	-
	$E_{2,coev}^i$	0	0	0	0	-	-
	$E_{3,coev}^i$	11	9	11	11	-	-
	$E_{1,sep}^i$	0	0	0	0	-	-
	$E_{2,sep}^i$	0	0	0	0	-	-
	$E_{3,sep}^i$	39	39	39	39	-	-
3	$E_{1,coev}^i$	0	0	0	0	0	-
	$E_{2,coev}^i$	0	0	0	0	0	-
	$E_{3,coev}^i$	8	8	8	8	8	-
	$E_{1,sep}^i$	0	8	0	0	0	-
	$E_{2,sep}^i$	0	0	0	0	0	-
	$E_{3,sep}^i$	39	31	39	39	39	-
4	$E_{1,coev}^i$	0	0	0	0	0	0
	$E_{2,coev}^i$	0	0	0	0	0	0
	$E_{3,coev}^i$	6	6	6	6	6	6
	$E_{1,sep}^i$	0	18	0	0	7	0
	$E_{2,sep}^i$	0	9	0	4	6	8
	$E_{3,sep}^i$	39	12	39	35	26	31

By analyzing the graphs and Table 3, it can be seen that the algorithms launched in coevolution were able to obtain the minimum Q_{min} before the common resources were exhausted only in one case (experiment 2). Having analyzed the “best”, i.e. smaller values of the fitness function (5), it can be concluded that the smallest solution ($Q_{min} = 20{,}03 \cdot 10^{-6}$) for all four experiments was obtained by launching an individual GEP algorithm. It should be noted that with an increase in the resource RE, the coevolutionary algorithms may be superior in finding the minimum compared to individual algorithms (in Figures 3-6, it is clear that the coevolutionary algorithms stopped much earlier than the individual algorithms).

As a result of comparing Q_{min} with all Q values (5) for each algorithm, it is easy to see that the coevolutionary algorithm is better than the "worst" of the individual algorithms, and even better than the "average" algorithm, but not always better than the "best" of the algorithms. It is important to take into account that not in every experiment the coevolutionary algorithms had enough resources to achieve the best result.

The graphs of the quality of the coevolution algorithms (6) for each experiment are shown in Figure 7. In Figure 7, the end of coevolution is marked with a purple dotted line. From the graphs, it is easy to see how much an increase in the number of algorithms with an unchanged size of the total resource affects the duration of coevolution. We can also pay attention to the accelerated growth of resource consumption, which in turn could affect the quality of the solution found by coevolution in the above experiments.

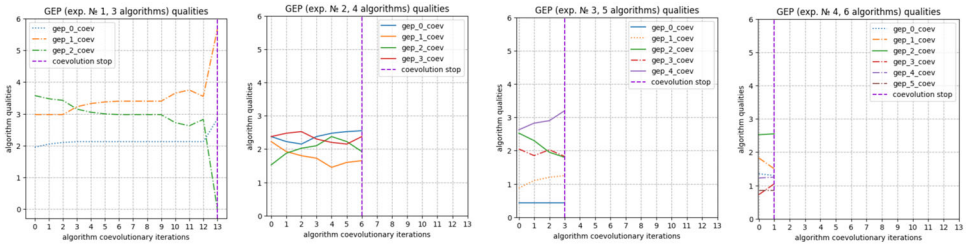

Fig. 7. Quality graphs of algorithms during experiments 1, 2, 3 and 4 in the coevolution state.

Having analysed Table 3, we can conclude that the given parameters of the coevolutionary algorithm (Table 2) showed the best result in Experiment 2. In Experiments 1, 3 and 4, individual algorithms showed the best results.

During the experiments, it was noticed that not every ET is able to be correctly compiled into some RE. Usually, this was due to internal limitations of the regular expression mechanism of the re module of the Python programming language. The graph of the number of individuals in different variants of running the algorithms (as part of coevolution and separately) is shown in Figure 8. It can be noted that coevolutionary algorithms (shown as a continuous line in Figure 8) have, on average, fewer individuals with errors in the population than individual algorithms (shown as a dashed line in Figure 8). This is due to the fact that in the case of a "loss" of some algorithm in the coevolution process, the losing algorithm loses a certain number of individuals in the population sorted by increasing values of the function (5). In case an individual has an erroneous structure, it is assigned the maximum possible value stub of the fitness function (5), entered by the researcher. In the experiment $stub = 10^3$. Because of this, erroneous individuals of the population are removed as a result of the coevolutionary algorithm.

In Figures 3-6, it can be seen that in different experiments had different results for the final best solution. This is due to the limited resources and the stochastic nature of the optimization. It can also be seen that in two cases out of four, the coevolutionary result at the time of coevolution completion was better (i.e., smaller) than the individual algorithms. In the other two cases, the coevolution results at the time of completion did not differ from the individual algorithms by more than 5 percentage points (all results were in the interval Δ_2). From this we can conclude that coevolution, on average, gets a better solution, faster than the individual algorithms.

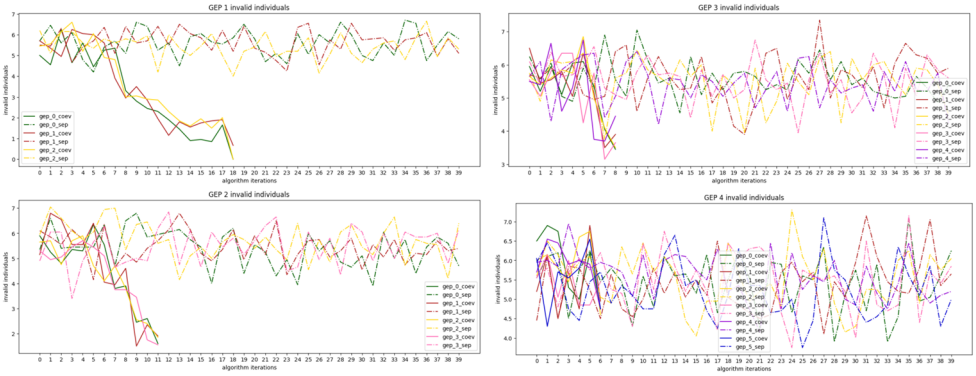

Fig. 8. Graphs of the number of erroneous individuals in the algorithms in all 4 experiments.

The problem of invalid individuals can be solved by adding rules for checking each individual. For example, on forbidden combinations or parameters for functions. If an individual has such a feature, it may be advisable to reinitialize it. Such an approach makes

sense, since Figure 8 shows that the number of individuals in the coevolutionary algorithm decreases as the population improves (due to the penalty).

4 Conclusion

The brevity and efficiency of regular expressions in the field of text data analysis have led to the fact that searching by a certain pattern has become very widespread. At the moment, there are a considerable number of RT implementations that repeat, in one way or another, the syntax of classical RE dialects [7]. Representing RE as an abstract "black box" model allows optimizing the speed of RT regardless of the internal implementation.

In the presented work, RE optimization was carried out using a coevolutionary optimization algorithm based on the GEP algorithm. Analysis of the results showed that the coevolutionary algorithm is capable of solving the RE optimization problem faster than it can be done by running individual algorithms. It should be noted separately that future studies will analyse the results of the coevolutionary algorithm when changing the values of such parameters as the total resource size, the penalty size, the size of the social card, and the size of the adaptation interval. Further research is planned to be related to the optimization of RE using some rules (equivalent replacements of constructions), testing other population algorithms, as well as testing other coevolution schemes and other population quality functions in coevolution. In addition, it is planned to study the possibility of "treating" individuals (ET) that are not able to compile into RE, with the aim of forming a population only from "viable" individuals.

References

1. Q. Chen, X. Wang, X. Ye and G. D. Dillig, *Multi-modal Synthesis of Regular Expressions*, in Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (2020).
2. A. Cataltepe and V. Kosoy, Time complexity for deterministic string machines, arXiv (2024) https://doi.org/10.48550/arXiv.2405.06043
3. G. Xing, Minimized Thompson NFA, International Journal of Computer Mathematics **81(9),** 1097-1106 (2004).
4. A. Currin, K. Korovin, M. Ababi, K. Roper, D. B. Kell, P. J. Day, R. D. King, Computing exponentially faster: Implementing a nondeterministic universal Turing machine using DNA, arXiv (2016) https://doi.org/10.48550/arXiv.1607.08078
5. F. Yu and Z. Chen, *Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection,* in Proceedings of the Architecture for Networking and Communications systems, ANCS (2006).
6. I.E. Tarasov, P.N. Sovietov, D.V. Lulyava, D.I. Mirzoyan, *Method for designing specialized computing systems based on hardware and software co-optimization,* Russian Technological Journal **12**, 3 (2024).
7. L. A. Demidova, N. A. Moroshkin, *Architecture of a regular expression translator with optimization of intermediate states* in Proceedings of the International Scientific Conference on Information Technologies, InfoTech-2024 (2024).
8. R. Sergienko, E. Semenkin, *Michigan and Pittsburgh methods combination for fuzzy classifier design with coevolutionary algorithm,* 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico (2013) pp. 3252-3259, doi: 10.1109/CEC.2013.6557968.

9. C. Ferreira, *Gene Expression Programming: A New Adaptive Algorithm for Solving Problems* Complex Systems, **13**, 2 (2001).
10. S. Couch, Z. Kazan, K. Shi, A. Bray, A. Groce, A Differentially Private Wilcoxon Signed-Rank Test, arXiv (2018) https://doi.org/10.48550/arXiv.1809.01635