DE GRUYTER
OPEN

TATRA MOUNTAINS
Mathematical Publications

DOI: 10.1515/tmmp-2017-0026
Tatra Mt. Math. Publ. **70** (2017), 179–197

EVOLUTIONARY COMPUTATION IN CRYPTANALYSIS OF CLASSICAL CIPHERS

Eugen Antal — Martin Eliáš

ABSTRACT. Evolutionary computation has represented a very popular way of problem solving in the recent years. This approach is also capable of effectively solving historical cipher in a fully automated way. This paper deals with empirical cryptanalysis of a monoalphabetic substitution using a genetic algorithm (GA) and a parallel genetic algorithm (PGA). The key ingredient of our contribution is the parameter analysis of GA and PGA. We focus on how these parameters affect the success rate of solving the monoalphabetic substitution.

1. Introduction

Historical (also called classical) ciphers belong to historical part of cryptology. They had been used until the expansion of computers and modern cryptosystems. Under a classical cipher we can consider a standard cryptosystem based on commonly used definition [6].

Comparing the main properties of classical ciphers to modern cryptosystems (used nowadays) leads to major differences in their properties. Some selected differences [5] are:

- The encryption algorithm of classical ciphers can be performed using paper and pencil (or some mechanical device) easily.
- Classical ciphers are mostly used to encrypt text written in some natural language.
- Classical ciphers are vulnerable to statistical analysis.

2010 Mathematics Subject Classification: 94A60, 68P25.
Keywords: historical ciphers, grid, MPI, genetic algorithm, parallel genetic algorithm.
This work was partially supported by grants VEGA 1/0159/17. We are grateful to the HPC center at the Slovak University of Technology in Bratislava, which is a part of the Slovak Infrastructure of High Performance Computing (SIVVP project, ITMS code 26230120002, funded by the European region development funds, ERDF), for the computational time and resources made available.

In this paper we use the following notation:

- P — plain text;
- C — cipher text;
- $\mathcal{A}_P$ — plain text alphabet;
- $\mathcal{A}_C$ — cipher text alphabet;
- k — key (secret parameter of the cryptosystem);
- e_k — encryption function parametrised with a key;
- d_k — decryption function parametrised with a key.

2. Cryptanalysis of classical ciphers

Cryptanalysis of historical ciphers can be seen (in a simplified way) as an effort to translate the encrypted text (also called cipher text, C) into a readable, meaningful text (plain text, P). The term cryptanalysis thus represents a method aimed at the correct decryption of the C [6], [7]. According to the Kerckhoffs' principle, we assume the knowledge of the cryptosystem, thus the aim of the cryptanalysis is to find the correct key used to encrypt the given ciphertext[1].

Based on the level of automatisation, the cryptanalysis can be: *computer assisted*, *partially automatic* and *automatic* cryptanalysis [5], [6]. In the first case, the computer serves only as a helper tool to perform some computation. In the case of partially-automatic cryptanalysis the computer searches the solution space automatically, but some operation steps are controlled by a cryptanalyst. The automatic way works as a black-box, the cryptanalyst obtains only the final solution.

The methods of automated cryptanalysis are mostly based on the following [5] approaches:

(1) *Brute force* — the cryptanalysis is based on the full key space search. This approach is applicable only in cases, when the key space is not too large and its performance highly depends on the methods of key quality evaluation. The efficiency of this method can be increased by negative tests or fast scoring functions [5], [6].

(2) *Specific methods based on cipher's weaknesses* — in some cases, it is possible to construct a special cryptanalysis method that uses some cipher specific weaknesses[2] [5].

[1] "A process of key recovery, or plain text recovery without the knowledge of the key" [7].

[2] The attack on the Zodiac Z408 homophonic cipher based on the cyclic repetition of symbols can serve as an example [3].

(3) *Using meta-heuristics* — an intelligent solution space search, where the brute-force is not applicable. The main idea is to evaluate the quality of keys using a fitness function, where the goal is to find the key with the highest fitness value (the global optimum of the function) more efficiently than in the case of brute-force [4], [5].

Meta-heuristics are based on iterative search of the solution space. One or more solution candidates are examined in each iteration. The search process is controlled by comparison of new solution candidates with candidates from the previous iteration, where the candidates with better fitness values are favoured. One of the most popular meta-heuristic methods is a genetic algorithm.

2.1. Genetic algorithm

The genetic algorithm (GA) is based (inspired) on the principle of natural evolution by C h a r l e s D a r w i n. The core elements are genes, chromosomes and population, where following genetic operations are performed on these objects: selection, mutation and crossover [8], [9].

- **Gene** is the basic building element of GA (low level encoded data). In most cases genes are encoded numerically or binary.
- **Chromosome** consists of genes, and represents the parameters (or properties) of a given problem. In our case it represents a key of a cryptosystem.
- **Population** is a fixed size group of chromosomes.
- **Generation** represents a population in some computation phase (iteration).
- **Fitness function** computes the fitness value (score) of a chromosome. The goal is to find the maximal (or minimal) value of a fitness function.
- **Fitness** represents the quality of a solution candidate.
- **Selection** is a process of selecting chromosomes into a subpopulation based on a selection strategy. This subpopulation serves as an input to a method, where chromosomes are modified via genetic operations (crossover and mutation). The main idea behind the selection is to investigate the chromosomes depending on their fitness value.
- **Mutation** means some small random change of genes in a chromosome. The aim is to maintain the diversity of chromosomes.
- **Crossover** is an operation, where two or more chromosomes (parents) are selected and new solutions are reproduced based on them. The crossover creates new solution candidates (children) containing information from both parents.

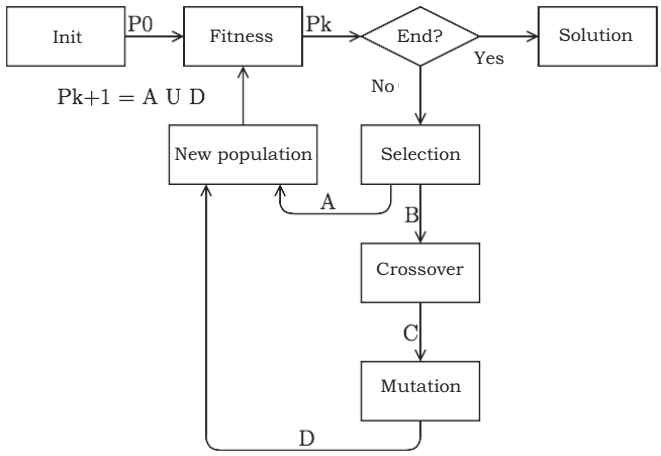


FIGURE 1. Structure of GA (based on [8]).

A very important process is to specify the problem encoding (fitness function parameters). In case of classical ciphers the chromosome represents the key, thus the genes can be encoded as letters or numbers. The next step is to specify the search space and the fitness function and then the population size and the scheme.

The main principle of GA is visible on Figure 1:

(1) **Init** — Generates the initial population P_0.
(2) **Fitness** — Evaluates the quality of the current population P_k using a fitness function.
(3) **End?**— The termination criteria are tested. If the criteria are satisfied, the best chromosomes from the population represent the solution.
(4) **Selection** — If the termination criteria are not satisfied, new subpopulations are selected. One or more chromosomes with the best fitness are selected (group A). These chromosomes ensure the convergence (achieving at least as good solutions as in the previous generation). Another subpopulation is moved into the crossover step (group B).
(5) **Crossover** — Parents from B are mated. Child chromosomes C are now moved to the mutation process.
(6) **Mutation** — Small random changes are performed on input chromosomes producing D.
(7) **New population** — P_{k+1} is a union of groups $A \cup D$. GA continues with step 2.

The presented scheme serves only as an example. A custom GA may differ in any step. In this paper we investigate several different schemes.

2.2. Parallel genetic algorithm

The parallel genetic algorithm (PGA) can be seen as an additional organisation level of a genetic algorithm. The goal of PGA is to increase the performance of GA. PGA offers some new benefits in comparison with GA: parallel search in a large multimodal search space, better organisation, higher efficiency on the same number of fitness function calls, higher probability of leaving local optimas [9].

The computation is performed on isolated and independent subpopulations called *islands*[3] [9]. Each island represents an instance of GA. The information between islands is exchanged using a *migration* process [9]. The time and the way of migration depends on the selected *migration model*. During the migration process one or more chromosomes are selected in one island and replace the same number of chromosomes in a different island. The most frequently used strategy is selecting the best/random chromosomes from island A and replacing the random/worst chromosomes in island B [9]. *Topology* describes how the islands are connected together [9] (different topologies are shown on Figure 2).

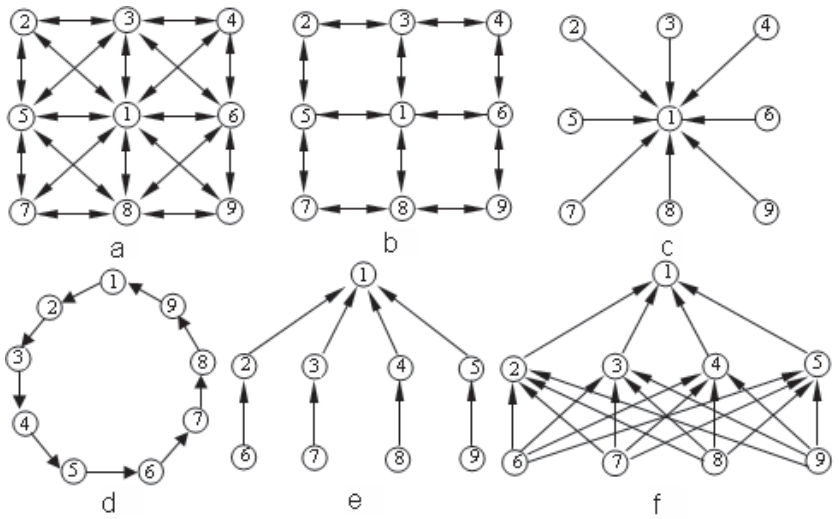


FIGURE 2. Different topologies of PGA [9].

[3]In most cases 4 or more islands are used [9].

3. GRID

One of our goals was to investigate the possibility of cryptanalysis of classical ciphers on a grid. Grid can be seen as a group or network of computers connected via LAN or another network technology. The nodes can be separated geographically. The goal of this network is to provide a large computation power. We used a supercomputer IBM iDataPlex available in CVT STU[4] [10], that consists of 52 computational nodes. Each node has the following configuration:

- cpu: 2 x 6 cores Intel Xeon X5670 2.93GHz;
- ram: 48GB;
- hdd: 2TB 7200RPM SATA;
- gpu: 2 x NVIDIA Tesla M2050 448 cuda cores, 3GB ECC ram;
- operating system: Scientific Linux 6.4;
- network connection: 2 x 10Gb/s Ethernet.

In summary we have 624 cpus, 3584 cuda cores, 2,5TB ram with performance of 6,76 TFLOPS (max. 40kW) [10].

4. Experiments with GA

Our first experiment deals with empirical solving of a monoalphabetic substitution using a genetic algorithm. We searched for those parameter settings of GA, which can produce the best success rate. We used our own implementation of GA, written in *C++*. We focused on parameters like: population size, number of iterations of GA, combination of genetic operations (schemes), namely:

- Population size: $10, 20, 50$ and 100 chromosomes.
- Iterations: 10000 and 50000.
- Schemes: presented in Table 1.

The first column in Table 1 is the name of the created scheme. The next columns represents the chain of genetic operations. Row of a scheme represents a chain of operations applicable to the actual population. As it can be seen on the scheme *J*, the first operation in the first row is selection of the best chromosome (elitism), this chromosome proceeds to the next generation without any changes.

[4]hpc.stuba.sk

The second row contains the same selection, where the chromosome is mutated (1 x swapping two randomly selected genes in the chromosome). The third row

Table 1. Schemes of GA.

Scheme	Selection	Mutation	Crossover	Subpop.
A	Tournament(n)	Swap(1)	-	-
B	Random(n)	Swap(1)	-	-
C	Tournament(n/2)	Swap(1)	-	-
	Tournament(n/2)	-	-	-
D	Tournament(n/2)	Swap(1)	-	-
	Random(n/2)	-	-	-
E	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	(n-2) * Tournament(2)	-	Singlepoint(2)	
F	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	(n-2) * Tournament(2)	-	Singlepoint(1)	-
G	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	$\big((n-2)/2\big)$ * Tournament(2)	-	Singlepoint(2)	Swap(1)
H	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	(n-2) * Random(2)	-	Singlepoint(1)	-
I	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	(n-2)/2 * Random(2)	-	Singlepoint(2)	-
J	Elite(1)	-	-	-
	Elite(1)	Swap(1)	-	-
	$\big((n-2)/2\big)$ * Random(2)	-	Singlepoint(2)	Swap(1)

of scheme J contains a $\big((n-2)/2\big)$-time call of random selection (of two chromosomes). These randomly chosen chromosomes are mated by a single point crossover, and their children are additionally mutated.

The success rate of these schemes was empirically tested on generated cipher texts of lengths $\{50, 100, 150, \ldots, 2000\}$. We used 100 different texts for each length (totally 4000). Each cipher text was created from a randomly selected[5] plain text. Each plain text was encrypted with a randomly generated key (permutation). After the execution of a GA, the best chromosome from the population was used as an output. After decryption, the result was compared with the correct plain text. In our experiments we used fitness functions based on Manhattan distance of 2-grams (also 3-grams) from their referential values. Analysis of different fitness functions is out of scope of this work. Extensive analysis of usable fitness functions is presented in work [2].

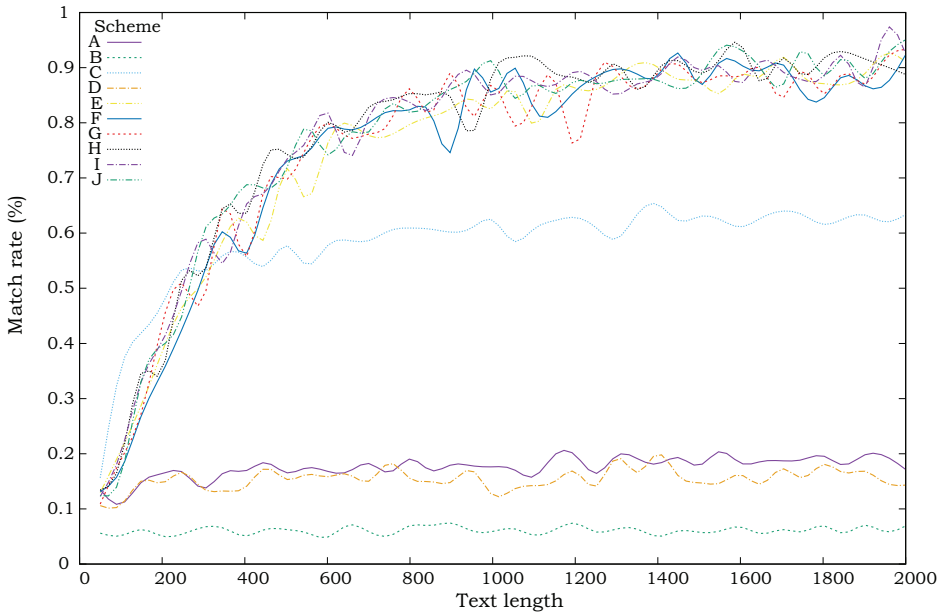
4.1. GA results

This experiment was performed on 96 computational nodes on hpc.stuba.sk. $320,000$ runs of GA were called for one fitness function.

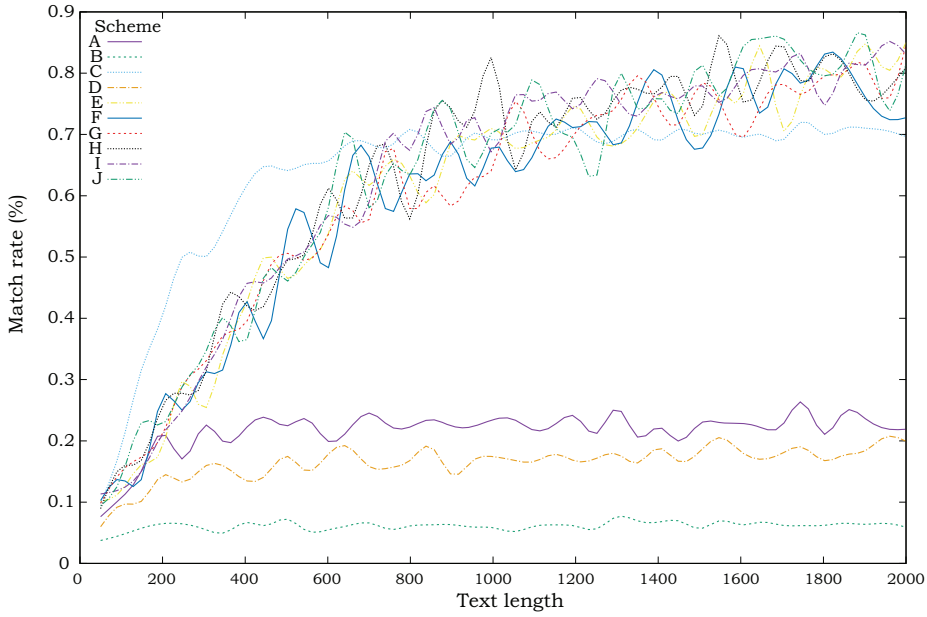
Results for the smallest investigated settings (10000 iterations and 10 chromosomes) are visible on Figure 3. Scheme B (random selection and swap as mutation) produces the worst results, as expected. The random selection gives too much diversity. Slightly better results can be obtained by schemes *A* and *D*.

The most surprising result is for scheme *C*. We expected that scheme without elitism cannot perform very well. The best results can be obtained by using schemes containing elitism, namely scheme *E, F, G, H, I, J*. Results for GA with 50 chromosomes are visible on Figure 4. Increasing the population size gives a low increase of success rate for schemes *A* and *D* and a higher increase of success rate for scheme *C*. In case of other schemes we also obtained better results. We obtained the best results for 50000 iterations and 100 chromosomes (Figure 5). Increasing the population size and iterations count helps to find better solutions for shorter text lengths.

[5]From the *Open American National Corpus* (AONC) [1]

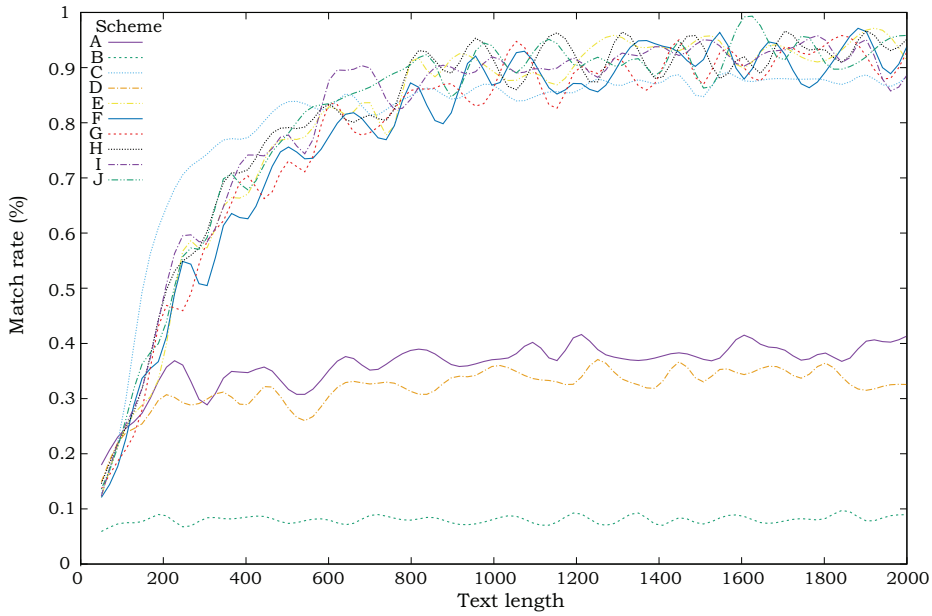


(A) 2-gram function.

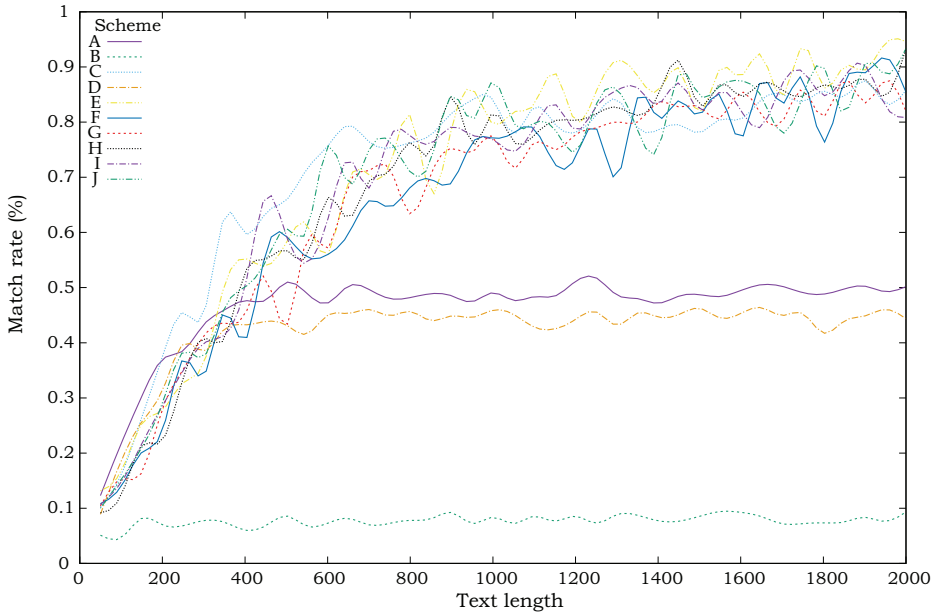


(B) 3-gram function.

FIGURE 3. GA results for 10000 iterations and 10 chromosomes.

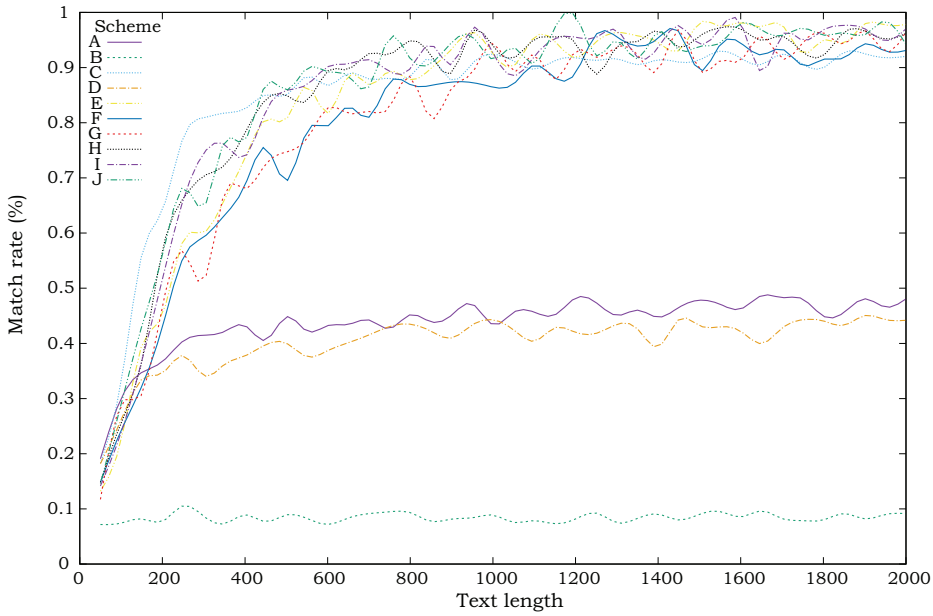


(A) 2-gram function.

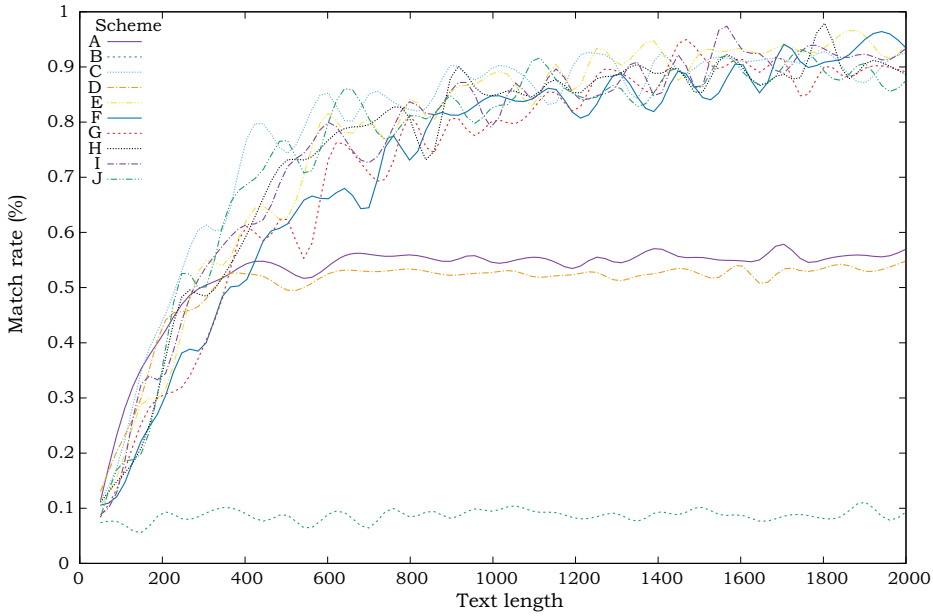


(B) 3-gram function.

FIGURE 4. GA results for 10000 iterations and 50 chromosomes.

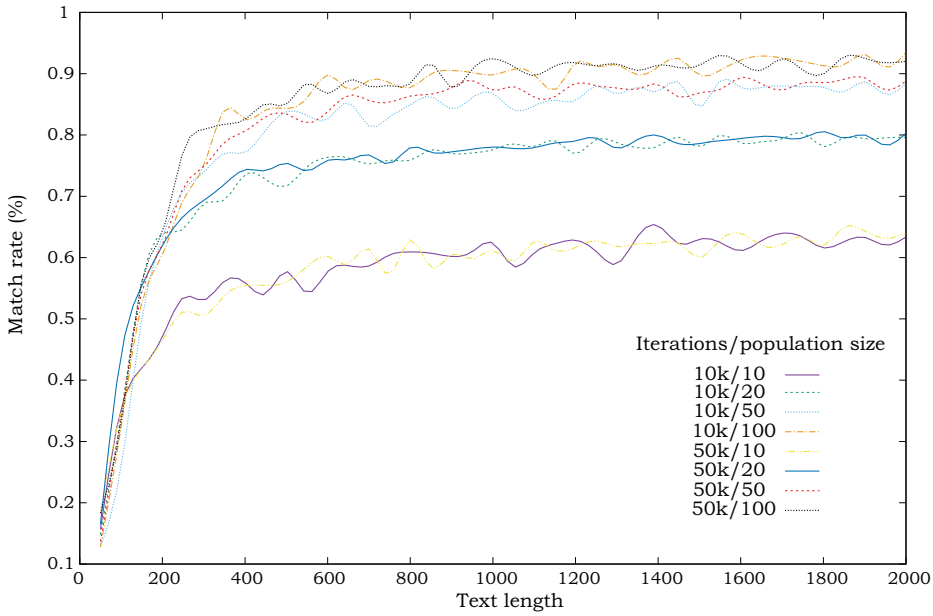


(A) 2-gram function.

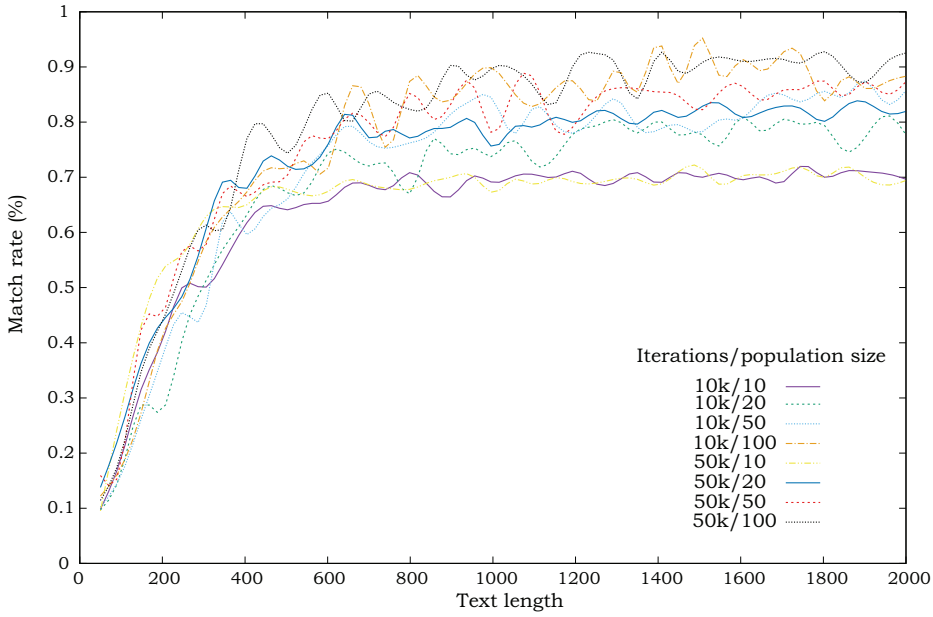


(B) 3-gram function.

FIGURE 5. GA results for 50000 iterations and 100 chromosomes.

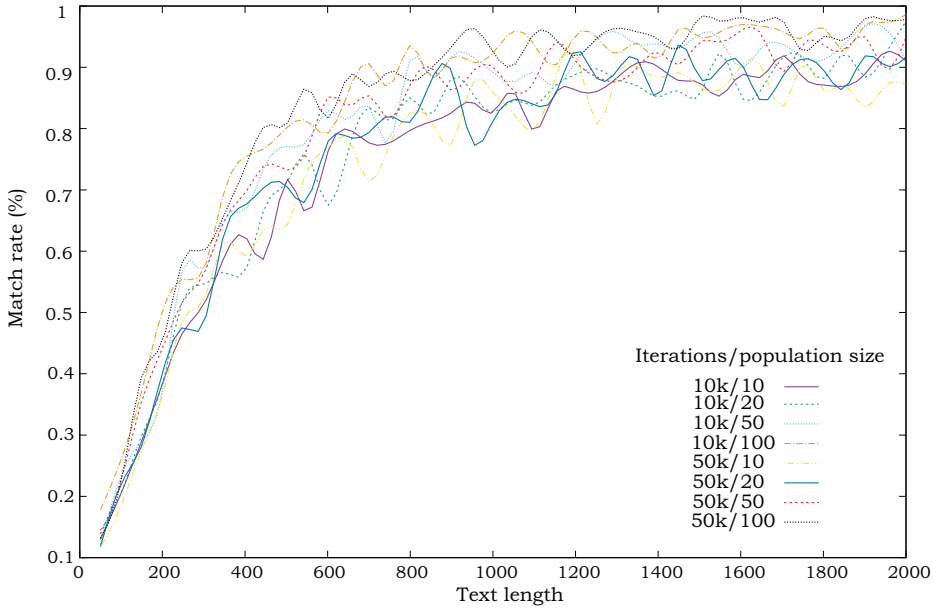


(A) 2-gram function.

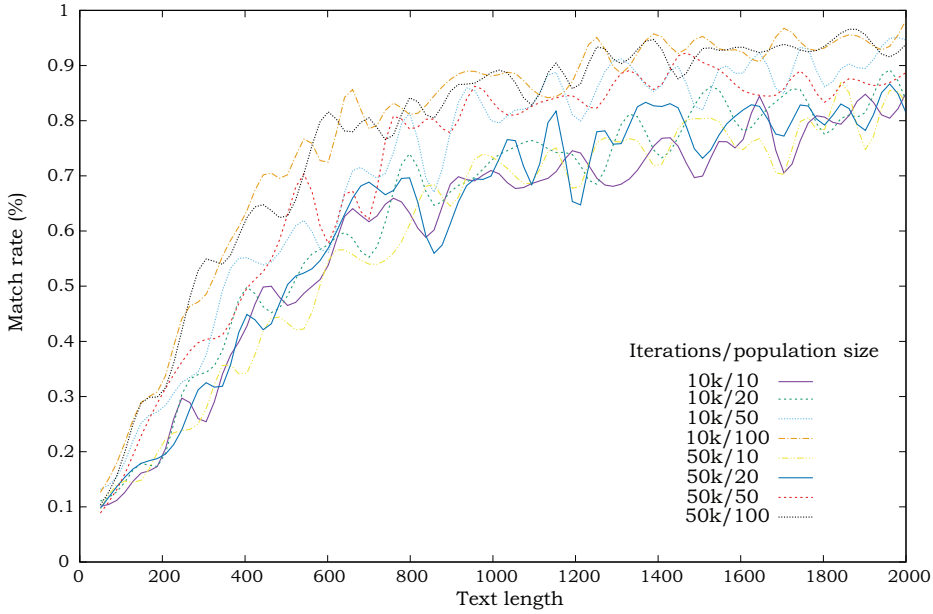


(B) 3-gram function.

FIGURE 6. GA results for scheme *C*.



(A) 2-gram function.



(B) 3-gram function.

FIGURE 7. GA results for scheme *E*.

To summarise this experiment, it stems from our results that the missing selection pressure (scheme B) does not allow the GA to converge to good results. As visible for scheme E (Figure 7), the increase in the populations size and iterations increases the success rate only slightly. As mentioned before, the most surprising result is for scheme C (Figure 6), since it can produce notable results without elitism. Elitism is very important for GA to obtain usable results. We recommend to use schemes C or E with minimum 50 chromosomes and with 10000 iterations.

5. Experiments with PGA

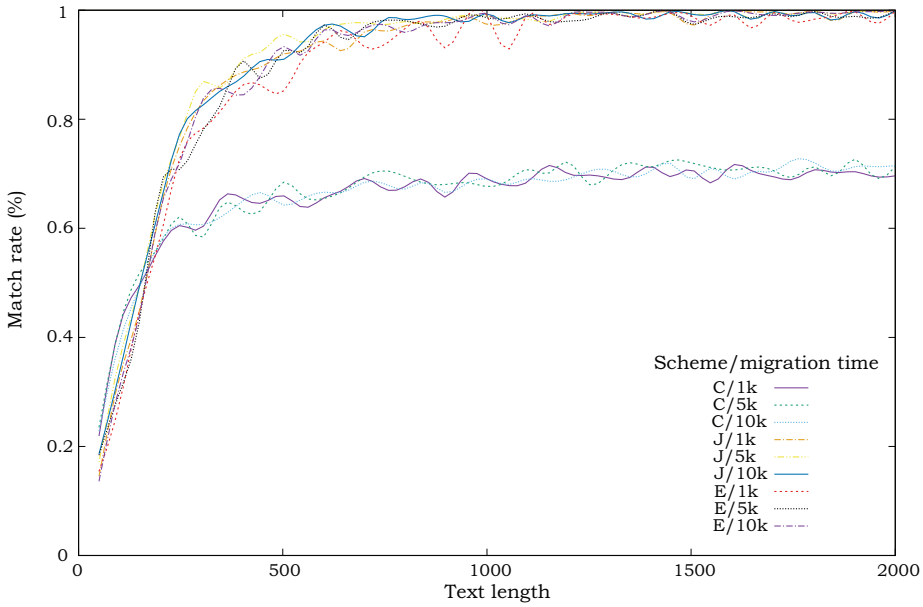
In the next experiment we deal with parallel genetic algorithm. We focused on parameters topology, number of islands and migration time. We adapted schemes C, E and F from the previous experiment to be used within the PGA. We used topologies b, d, e, f from Figure 2 with 3, 5 and 11 islands. For 3 and 11 islands we correct some topologies, scheme b has a triangular shape in the case of 3 islands, and an additional square was added to nodes 4 and 6 in the case of 11 islands. We selected 30000 GA iterations, with the following migration times: 30, 6 and 3 times (once per 1000, 5000 and 10000 iterations).

The experiment was performed on the same dataset. The final result was the best chromosome selected from all populations (all islands) for topologies b and d. In the case of topologies e and f, the best result was selected from the island located on top of the topology.

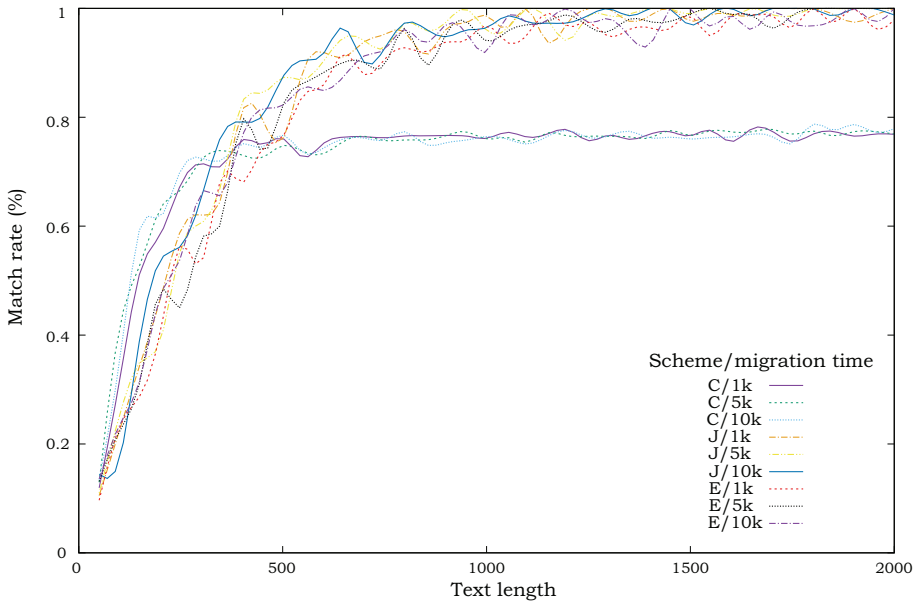
5.1. PGA results

$7,884,000$ runs of GA were called for one fitness function in this experiment. Result for topologies b a d are visible on Figures 8 and 9. We can state from the results that the topology does not really influence the success rate in the case of our problem. But the PGA obtains better results compared to GA. In the case of schemes E and J we get 80 % success rate for short texts of length 250/500 letters and almost 100 % success rate for texts longer than 1000 letters.

All schemes have better results for PGA. As we can see on Figures 10 and 11 increasing the population size (also increasing the number of islands) increases the success rate. Bigger difference can be seen between population 20 (Figure 10) and 50 (Figure 11), especially in the scheme C.

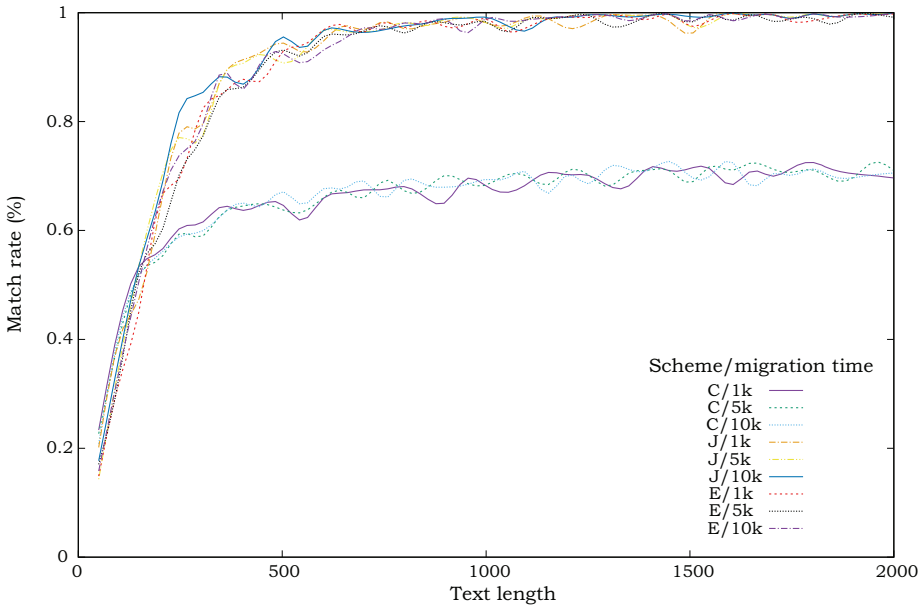


(A) 2-gram function.

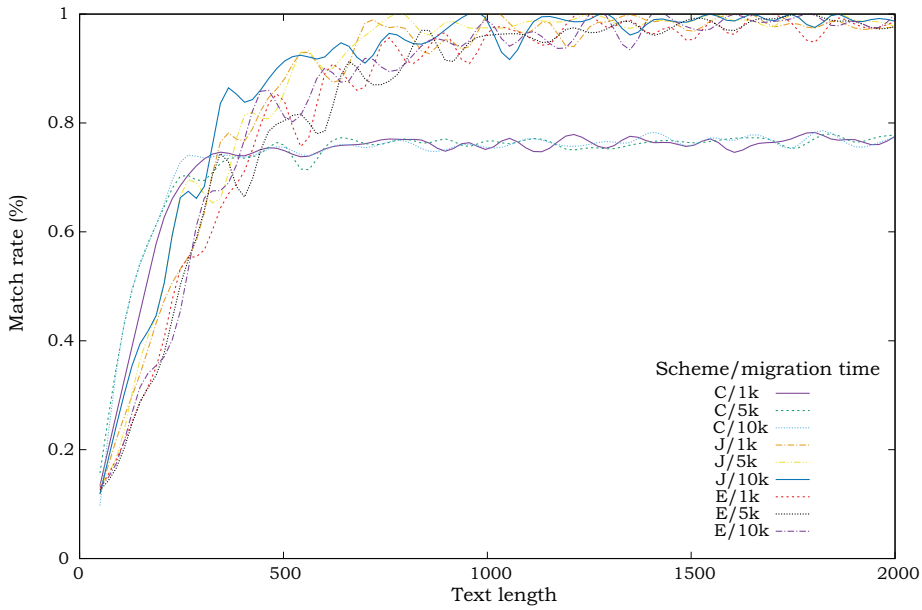


(B) 3-gram function.

FIGURE 8. Success rate depending on cipher text length (topology b/3, 10 chromosomes).

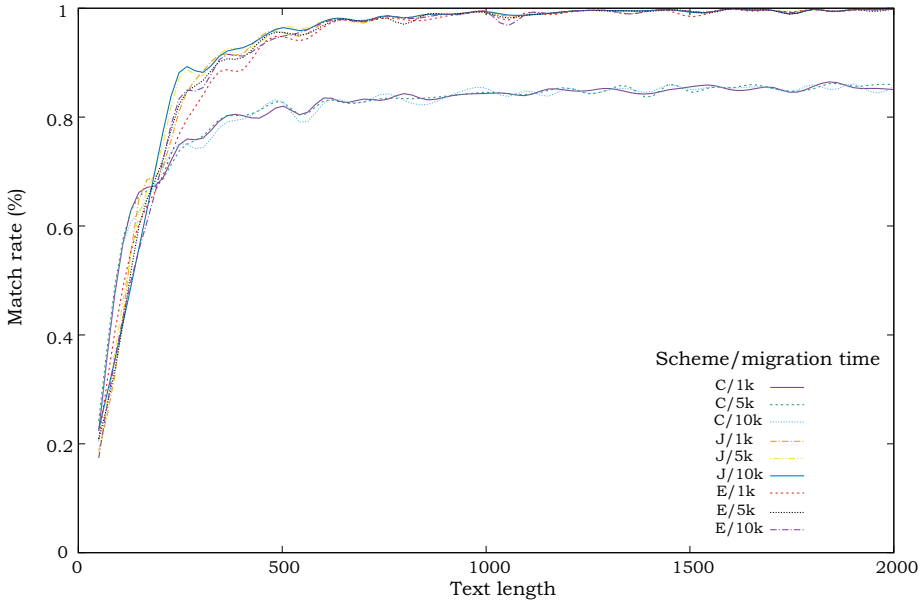


(A) 2-gram function.

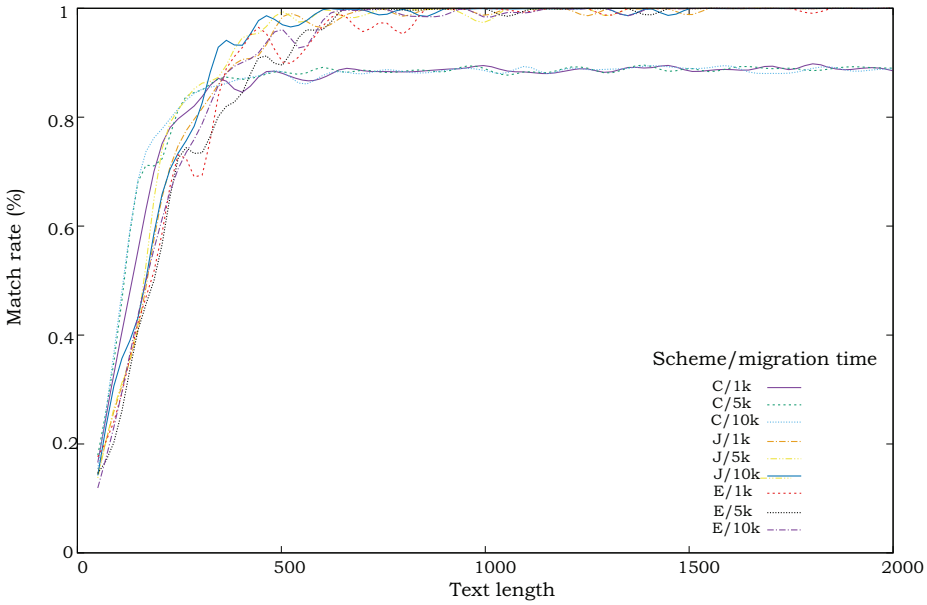


(B) 3-gram function.

FIGURE 9. Success rate depending on cipher text length (topology d/3, 10 chromosomes).

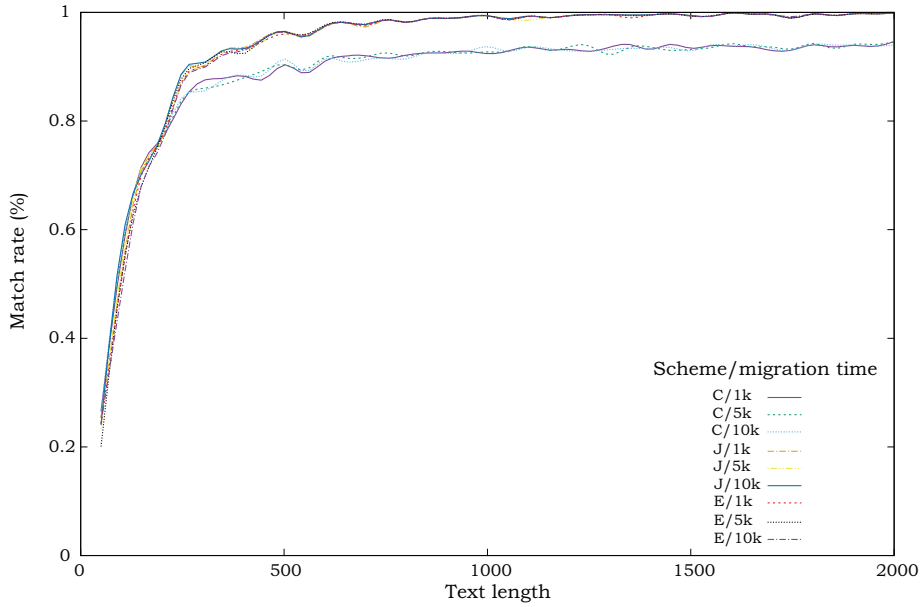


(A) 2-gram function.

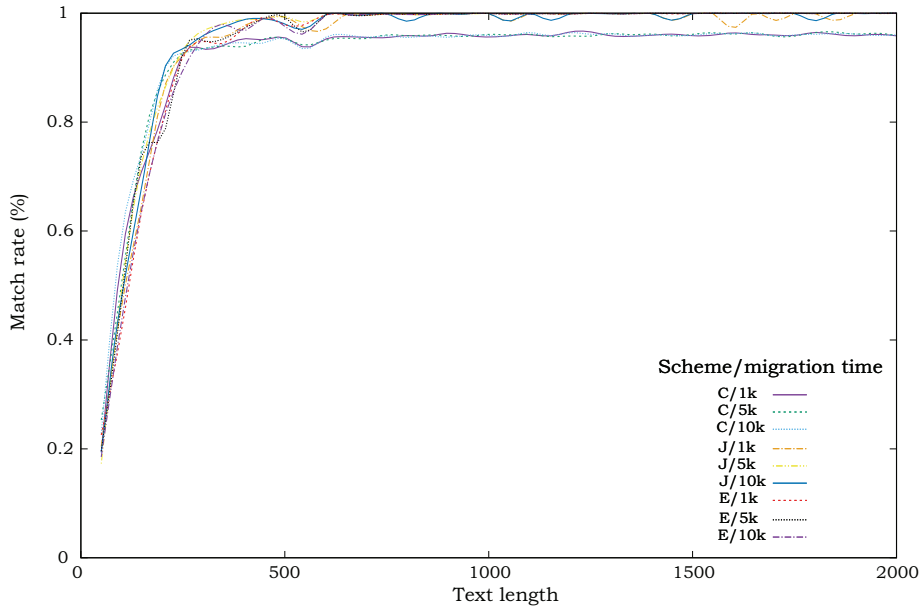


(B) 3-gram function.

FIGURE 10. Success rate depending on cipher text length (topology b/5, 20 chromosomes).



(A) 2-gram function.



(B) 3-gram function.

FIGURE 11. Success rate depending on cipher text length (topology b/11, 50 chromosomes).

Summary

In the first experiment we investigated different schemes of a genetic algorithm. The goal was to find a suitable scheme producing the best results. This experiment was performed on 96 nodes of grid. We were able to obtain 80 % success rate for seven schemes out of ten for texts of length 500 letters and more than 90 % for texts longer than 1000 letters. The best schemes were selected and used in the next experiment with PGA. Four different topologies and three migration times were tested in these schemes[6]. From achieved results we can state, that in our case (solving monoalphabetic substitution) the topology has almost no effect on the result. The success rate depends on the size of populations and on the number of islands. Using PGA we were able to achieve near 100 % success rate for texts of length 500 letters. We achieved very notable results also on shorter text lengths.

REFERENCES

[1] *American National Corpus Project* `http://www.anc.org/data/oanc/download/`
[2] ANTAL, E.: *Modern Cryptanalysis of Classical Ciphers*. PhD Thesis, Slovak University of Technology, Bratislava, 2017. (In Slovak)
[3] ANTAL, E.—VARGA, J.: *Zodiac*, In: Mikulášská kryptobesídka, 2010; Sborník příspěvků, Praha, 2.–3. 12. 2010. Praha: Trusted Network Solutions 2010, pp. 89–90. (In Slovak)
[4] CLARK, A. J.: *Optimisation Heuristics for Cryptology*. PhD Thesis, Queensland University of Technology, 1998.
[5] GROŠEK, O.—VOJVODA, M.—ZAJAC, P.: *Classical Ciphers*. STU Bratislava, 2007. (In Slovak)
[6] GROŠEK, O.—ZAJAC, P.: *Automated cryptanalysis of classical ciphers*. In: Encyclopedia of Artificial Intelligence, 2008, pp. 186–191.
[7] ——— *Automated cryptanalysis*. In: Encyclopedia of Artificial Intelligence, 2009, pp. 179–185.
[8] SEKAJ, I.: *Evolutionary Computations and their Usage in Practice* Iris, 2005. (In Slovak)
[9] SEKAJ, I.—ORAVEC, M.: *Parallel evolutionary algorithms*. In: Umelá Inteligencia a Kognitívna Veda III, 2011, Slovak Univerzity of Technology, Bratislava, 2011, pp. 243–267. (In Slovak)
[10] *STUBA klaster — IBM iDataPlex*, `https://www.hpc.stuba.sk`

Received October 31, 2017

Institute of Computer Science and Mathematics
Slovak University of Technology
Ilkovičova 3
812 19 Bratislava,
SLOVAKIA
E-mail: eugen.antal@stuba.sk

[6]The execution time was approximately two and a half week in the case of 2-gram fitness function.