Tatra Mt. Math. Publ. **91** (2025), 51–68
DOI: 10.2478/tmmp-2025-0029

TATRA MOUNTAINS
Mathematical Publications

SOLVING SPARSE MRHS SYSTEMS WITH GENETIC ALGORITHMS

EUGEN ANTAL — PAVOL ZAJAC — SABINA PEKAREKOVÁ

Institute of Computer Science and Mathematics Slovak University of Technology in Bratislava
SLOVAKIA

ABSTRACT. Multiple Right-Hand Sides (MRHS) equations represent a mathematical formalism with applications in algebraic cryptanalysis. Solving MRHS equation systems is in general a difficult problem. In this article, we investigate the efficient use of genetic algorithms for solving random sparse MRHS systems. Our experiments suggest that the steady-state selection method with low elitism and mutation rate gives the best results. If the systems are sparse, the system size does not have a significant impact on the success of the algorithm. On the other hand, the method is very sensitive to the system density, with the success rate rapidly declining with increased system density.

1. Introduction

Multiple Right-Hand Sides (MRHS) equation [13] is a formal inclusion of the form $\mathbf{xM} \in S$, where on the left-hand side we have a linear equation system and S is an arbitrary set of vectors. In general, solving MRHS equation systems is a hard problem (the decision version is NP-complete). A system of MRHS equations can provide an efficient algebraic description of linear and non-linear components of block ciphers that can be used in algebraic cryptanalysis [23]. MRHS cryptanalysis was applied to DES [15], AES [13], Trivium [21], GOST [24], JH and Keccak [1], Present [10], DESL [12], LowMC [7], Ascon [17], and possibly others. MRHS equations can also be used in linear cryptanalysis [6, 16], and even in post-quantum cryptography [18, 25].

A new type of MRHS solver based on bit-flipping that is suitable for solving sparse MRHS equation systems was introduced in [22]. A logical extension of this type

2020 Mathematics Subject Classification: 68T20, 94A60.
Keywords: MRHS problem, genetic algorithm, algebraic cryptanalysis, MRHS equation systems.
This research was supported by Ministry of Education, Science, Research and Sport of the Slovak Republic through grants VEGA 2/0054/24, and VEGA 1/0105/23; and by the Slovak Research and Development Agency under the Contract no. APVV-23-0292.

of solver is a solver based on evolutionary algorithms: the problem of solving an MRHS system is transformed into an optimization problem by using a specific fitness function that reaches the global optimum in proper solutions of the MRHS system. This idea was implemented [17] in the form of a hill-climbing algorithm.

A Genetic Algorithm (GA) is an optimization technique inspired by the two fundamental principles of natural evolution: natural selection and genetic dynamics. The foundational concepts of genetic algorithms were first introduced by Holland [8]. Authors of [4] demonstrate that the Boolean Satisfiability Problem (SAT) serves as a GA-efficient canonical problem and that other NP-complete problems with suboptimal GA representations can be efficiently solved by first translating them into SAT problems (which we know that can be done in polynomial time). In addition to solving SAT problems, GAs have also been applied to other NP-complete problems, such as the N-puzzle problem [5], Vertex Cover problem [2], Maximum Clique problem [3], Subset Sum problem [20], and the decision version of the Traveling Salesman problem [11].

A basic implementation of MRHS solver based on genetic algorithms was explored in [9]. Independently, a genetic algorithm for solving MRHS systems was proposed in [19], but the algorithm was only tested on a specific small MRHS system instance. In this contribution, we aim to systematically explore the use of genetic algorithms for solving random sparse MRHS equations. We explore the impact of various parameters of the used genetic algorithm, such as population size, selection types, crossover, mutation rates, etc. on the success rate of the algorithm.

The paper is structured as follows: Section 2 contains a description of an MRHS system. Section 3 describes a usage of GA to an MRHS problem. Section 4 is devoted to the experimental results that we achieved. We conclude our paper with Section 5.

2. MRHS systems and MRHS problem

Although Multiple Right-Hand Sides equation systems (MRHS systems) can be defined over a general field [13], we will restrict ourselves to MRHS systems over binary field $\mathbb{F}_2$. Notation: we suppose that all vectors are row vectors, typed in boldface, such as $\mathbf{x} \in \mathbb{F}_2^n$. Similarly, matrices are typed in capital boldface, such as $\mathbf{M} \in \mathbb{F}_2^{n \times m}$.

DEFINITION 2.1. Let $\mathbf{M} \in \mathbb{F}_2^{n \times l}$, and let $S \subset \mathbb{F}_2^l$. A Multiple Right-Hand Sides equation (defined by the tuple $(\mathbf{M}, S)$) is a formal inclusion in the form

$$\mathbf{xM} \in S.$$

Any vector $\mathbf{x} \in \mathbb{F}_2^n$, for which the inclusion holds, is called a solution of the MRHS equation.

DEFINITION 2.2. A Multiple Right-Hand Sides equation system (MRHS system) is a set of MRHS equations defined by tuples $(\mathbf{M}_i, S_i)$ for $i = 1, 2, \ldots, m$ with common dimension n. Vector $\mathbf{x} \in \mathbb{F}_2^n$ is a solution of the MRHS system, if and only if it is a solution of each MRHS equation in the system. MRHS problem is a problem of finding a solution for a given MRHS system (if it exists).

Note that in general the dimension of individual MRHS equations in a system can be different, $\mathbf{M}_i \in \mathbb{F}_2^{n \times l_i}$. In the experimental part, we will focus on MRHS systems with equal dimensions of right-hand side sets, that is $l_i = l$ for each i, but the general idea works for any type of MRHS systems. We will also fix the number of vectors in each S_i, $|S_i| = k$, with a random selection of vectors in each S_i, for consistent evaluation of results.

A MRHS system can also be written in a joint form by concatenating matrices on the left-hand side, and using a Cartesian product on the right-hand side:

$$\mathbf{x}\,(\mathbf{M}_1 \mathbf{M}_2 \cdots \mathbf{M}_m) \in S_1 \times S_2 \times \cdots \times S_m.$$

The joint form of the system is useful for algebraic operations with the system, such as are used in the systematic solver of MRHS systems by [14] (RZ solver).

A basic principle of heuristic algorithms for solving MRHS systems is the concept of bit-flipping [22]. For an arbitrary vector $\mathbf{x}$, we can compute left hand side vector $\mathbf{v} = \mathbf{x}\,(\mathbf{M}_1 \mathbf{M}_2 \cdots \mathbf{M}_m)$, and represent it as a block vector $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m)$, with blocks $\mathbf{v}_i$ having dimension l_i, corresponding to the dimension of S_i. We will say that MRHS equation i is satisfied, if $\mathbf{v}_i \in S_i$, otherwise it is unsatisfied. MRHS system is solved, if each MRHS equation in the system is satisfied. The bit-flipping algorithm attempts to increase the number of satisfied MRHS equations by changing individual bits of $\mathbf{x}$. The hill-climbing algorithm combines bit-flipping with greedy heuristic, and restarts: choose random starting $\mathbf{x}$, change one of the bits that provides the largest increase of satisfied MRHS equations, and restart when an increase is impossible. Both the RZ solver and hill-climbing-based method (HC solver) are publicly available at:

`https://github.com/zajacpa/mrhs-solver`.

Our previous research of bit-flipping-based solvers has established their suitability for solving sparse MRHS systems, with degrading performance associated with increasing density of joint matrix $\mathbf{M}$. We say that the MRHS system is sparse, if each column of $\mathbf{M}$ contains exactly one non-zero element (has Hamming weight 1). In experiments, we further require that $\mathbf{M}$ has full row rank (to maintain full dimension n of the solution space) and that each $\mathbf{M}_i$ has full column rank (to have full prescribed dimension l_i).

To demonstrate changing behavior with increased density, we define density parameter d. An MRHS system with density $d > 0$ contains $d \cdot \dim(\mathbf{xM})$ non-zero elements in the joint matrix. The non-zero elements are distributed randomly, but in such a way that each block has full column rank. This means that with $d = 1$ each column

has exactly 1 non-zero element. For $d > 1$, parameter d becomes an expected number of non-zero elements in each column (with minimum at least 1).

Note that bit-flipping and evolutionary algorithms are not suitable for cases where the solution of the MRHS system does not exist. This is because of the missing halting condition (termination criteria), if the solution does not exist, the algorithm would be stuck in a restart loop forever. In practice, we restrict the number of iterations of the algorithm, and if the solution is not found within a specified number of iterations, we suppose that it does not exist. To avoid this problem in batch experiments, we artificially create at least one solution: We generate a random $\mathbf{x}$, and compute vectors $\mathbf{v}_i = \mathbf{x}\mathbf{M}_i$. In each set S_i, we replace a random vector from S_i with the generated $\mathbf{v}_i$. In our experiments, we set the parameters of the system (n,m,l,k) in such a way that the expected number of solutions is exactly 1[1]. By changing the vectors in S_i to fit the artificial replacement solution, we have ensured that one solution is present, and due to the probability distribution, we expect that no other solution of the system exists (if there was one prior, it was replaced when changing S).

3. Using genetic algorithms for solving MRHS problem

Genetic Algorithm is an optimization technique used to solve hard optimization problems. In a simpler sense, it works as an iterative search process controlled by the comparison of new solution candidates with candidates from the previous computation cycle, where the candidates with better *fitness values* are favored. Fitness value represents the quality of a solution candidate (its score) and is computed by a *fitness function*. The core elements of a GA are:

- *Gene* is the basic building element of GA-low-level encoded data.
- *Chromosome* consists of genes and is a solution candidate for a given problem. In other words, it represents the parameters of the fitness function.
- *Population* is a fixed-size group of chromosomes. *Generation* represents a population in a specific computation phase.

The following operations are performed on these elements:

- *Selection* is a process of selecting chromosomes into a sub-population based on a selection strategy.
- *Mutation* means some small random change of genes in a chromosome.
- *Crossover* is an operation, where two or more chromosomes (parents) are selected and new solutions (children) are reproduced based on them.

[1]With $l = 3$, $k = 4$, each random S_i is satisfied with probability $1/2$. We then expect $2^n \cdot (1/2)^m$ solutions, which is 1 when $n = m$.

There are several GA settings (schemes) that affect its behavior. We used a standard GA scheme provided with the used PyGAD library available at:

`https://pygad.readthedocs.io/`,

which consists of the following steps:

(1) Initialize population of N individuals.

(2) Evaluate the fitness of individuals in the population.

(3) Select parents, do crossover, do mutation, and update the population.

(4) Repeat, until a solution is found, or too many generations passed.

To be able to solve MRHS problems with a GA, it is necessary to specify it as an optimization problem (the problem representation and the fitness function). We are using binary representation of chromosome $\mathbf{x}$ of length n, $\mathbf{x} \in {\mathbb{F}_2}^n$, therefore genes are bits (values $0/1$). The evolutionary solver uses a population represented by chromosome (potential solution vectors) $\mathbf{x}$.

The fitness function computes a score of each individual (chromosome) in the population based on "distance" of $\mathbf{x} \cdot \mathbf{M}$ from $S_1 \times S_2 \times \cdots S_m$:

$$d_i = \min \left\{ \frac{\text{dist}(\mathbf{x} \cdot \mathbf{M}_i, \mathbf{v})}{\dim(\mathbf{x} \cdot \mathbf{M}_i)}; \mathbf{v} \in S_i \right\},$$

$$\text{score} = 1 - \frac{\sum d_i}{m},$$

where dist is the Hamming weight of vector difference:

$$\text{dist}(\mathbf{y}, \mathbf{v}) = w_H(\mathbf{y} \oplus \mathbf{v}).$$

The individual components d_i of the fitness function measure how far is the partial solution from satisfying specific S_i (minimum Hamming distance is taken relative to the dimension of S_i). Final score sums these distances and normalizes the result into the interval $(0, 1)$. Maximum score of 1 means that $\mathbf{x}$ is a solution, because it satisfies all MRHS equations.

It is possible to use a simpler scoring function that just measures the number of unsatisfied MRHS equations. Preliminary experiments with the simpler function produced worse results, thus it was not used for full experiments.

The original population is randomly selected. Exactly half of the current population is selected into the mating pool in each generation. Then the crossover and mutation operations are performed. Due to the binary representation of chromosomes, we are using a single-point crossover method implemented in the PyGAD library. It selects a point randomly at which crossover takes place between the pairs of parents. The used mutation is also adapted to the binary representation. In the chromosomes, each gene

is selected for applying the mutation operation with the predefined probability. The random mutation operation implemented in the used PyGAD library is then applied to the selected genes, randomly changing its value to 0 or 1.

4. Experimental results

To gain a better understanding of the applicability of genetic algorithms for solving MRHS systems, we have performed a series of experiments. We have focused on random systems with reasonably large sizes. The default size is 48 unknowns which restricts the search space to 2^{48} potential vectors $\mathbf{x}$. In the initial experiments (Section 4.1, also reported at CECC'24) we have focused on tuning the genetic algorithm parameters on a small number (10) of systems.

Full experiments were conducted on a set of 1000 randomly generated MRHS systems. These systems were generated with the MRHS solver version 1.9 from `https://github.com/zajacpa/mrhs-solver`, with the seed corresponding to the counter value (1 to 1000), and option "r" enabled (to enforce the existence of a random solution). The default setting was $n = m = 48$ (number of variables equal to the number of MRHS equations), and $l = 3,\ k = 4$ (each RHS with 4 vectors of dimension 3). The parameters m, l, k should ensure that there is approximately 1 solution in a randomly generated system (each MRHS equation discards 1/2 of the potential solutions). The default density was $d = 1$, meaning a sparse MRHS system (with a single non-zero element in each column of the joint system matrix).

In Section 4.2 we explore the effect of the system size. We have both reduced and enlarged the number of unknowns by 3 and 6 bits, all the other settings remained default. We have also conducted an extra experiment with a very large system of 100 unknowns (search space size 2^{100}).

Finally, in Section 4.3 we explore the effect of the matrix density. We have increased the density parameter to $d = 2$, and $d = 3$, respectively. We also run an experiment with dense systems, where the joint matrix is completely random (which roughly corresponds to density $d = n/2$) for comparison.

In each test case, we tested n different MRHS systems and evaluated the success rate — the number of systems we were able to solve successfully. We also investigated the convergence of fitness functions (the dependence of the average fitness value on the number of generations). For most of the experiments, we are presenting only the charts of the success rate (estimated solutions found).

The figures have x-axis (number of generations) expressed in logarithmic scale, due to the exponential character of the problem. It is possible to solve MRHS systems by randomly trying different $\mathbf{x}$'s. By doubling the number of generations of "random guessing" we should improve the success rate of naive algorithmic solution linearly.

Thus, the success rate of naive algorithm would be represented by a straight line in a figure with x-axis in logarithmic scale.

4.1. Initial tuning of parameters

Setting the parameters of a genetic algorithm for the chosen optimization problem is not trivial. It is necessary to select the optimal ratio of diversity and selection pressure to be able to solve the optimization problem effectively. For this reason, we investigate several different parameters of the genetic algorithm.

All experiments in this section were conducted with a fixed set of random MRHS systems with

$$n = m = 48, \quad l = 3, \quad k = 4, \quad \text{and} \quad d = 1.$$

This corresponds to a hard instance of a problem, with a single expected solution in a space of 2^{48} potential unknown vectors.

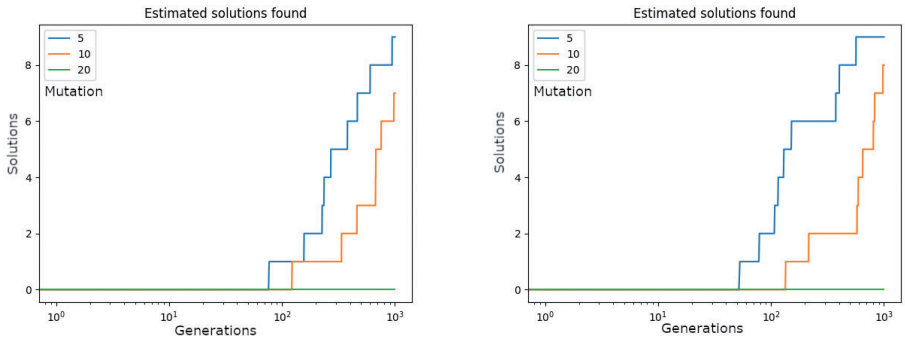
In our first experiment, we focused on investigating the most suitable settings for the basic parameters of the algorithm. The goal was to identify the minimum number of generations, the population size, and the mutation rate required for the algorithm to converge to the correct solutions. We were using 10 % elitism, single point crossover, 50 % of the population selected to the mating pool with tournament selection, and tested the following parameters:

- population size 10, 100, 1000;
- generations 100, 1000, 5000;
- gene mutation rate 5 %, 10 %, 20 %, 30 %, 40 %, 50 %.

For small population size (10 individuals), the GA needs at least 1000 generations to converge. The correct solutions were found only for 5 % and 10 % gene mutation rates, where the lowest mutation converged the fastest. Increasing the number of generations slightly increases the efficiency of finding the correct solution for the 5 % mutation rate, and significantly for the 10 % mutation rate. For higher mutation rates the algorithm is still converging, however, the correct solutions were not found until the termination criteria.

Increasing the population size significantly increases the success rate. With a population of 100 individuals, the GA converges to the correct solution between 100 and 1000 generations for 5 % and 10 % gene mutation rate, and above 1000 generations also for 20 % gene mutation rate. Increasing the population size to 1000 individuals increases the efficiency (the algorithm converges faster, the correct result is achieved sooner), and also increases the probability of finding the correct solution for the 10 %, and 20 % gene mutation rates. From the achieved results, it is clear that selecting a higher mutation rate is not efficient, and it is necessary to use high selection pressure.

In the next step, we investigated if we can achieve good results with a different selection method, and with smaller elitism. We fixed the termination criteria to 1000 generations, and tested the following parameters:



(A) random (B) roulette

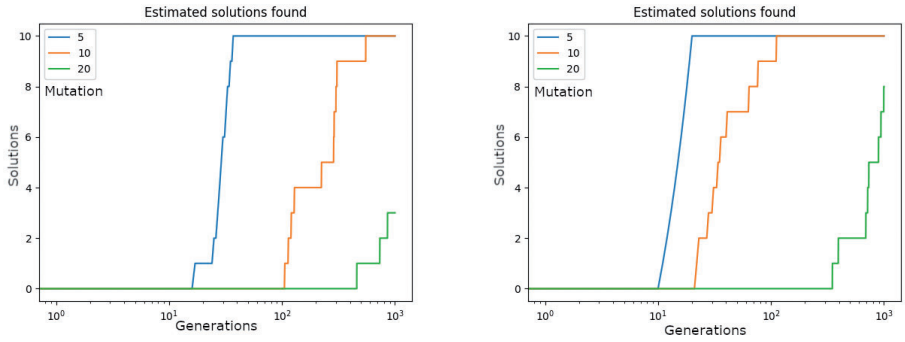
FIGURE 1. Comparison of random and roulette selection methods and mutation rates (using 1 % elitism, 1000 generations and population size 1000).

- elitism 1 %, 10 %;
- population size 10, 100, 1000;
- gene mutation rate 5 %, 10 %, 20 %;
- rank, roulette wheel, steady-state, tournament, and random selection methods.

First, we tested the smaller elitism. The rank selection method was the only one, where we were not able to obtain the correct result. Random and roulette wheel selection methods were almost equally successful, but they were efficient only in case of small mutation rates and higher population sizes. The tournament and steady-state selection methods performed the best. We were able to achieve the correct solution with 5 % mutation rate from population size 100 and with 10 % mutation rate from population size 1000 (see Figures 1 and 2). The steady-state selection method converged faster than the tournament selection method and was also able to find the correct solution with the 20 % mutation rate in more cases.

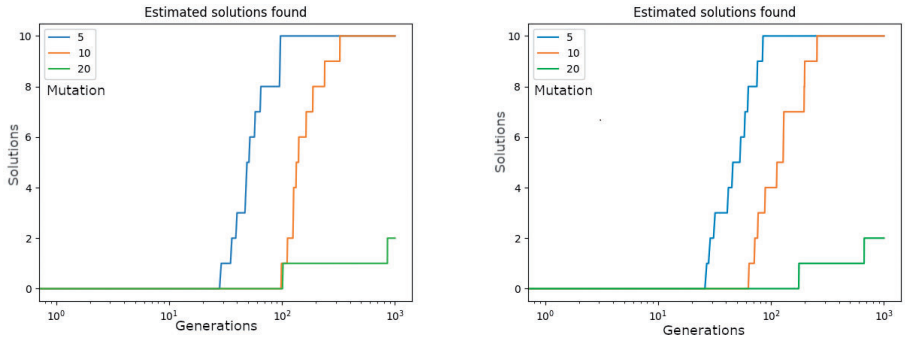
Increasing the elitism to 10 % had almost no effect on population size 10. The differences started to show only when the population size was larger. In general, the effectiveness of the selection methods copies the result from the previous experiment. However, higher elitism increased the probability of achieving the correct solution and also increased the success rate for cases using a 20 % mutation rate. With 10 % elitism, we were able to achieve the correct result for all inputs for all tested mutation rates using the steady-state selection method and population size 1000 (see Figures 3 and 4).

Note that the running times depend on the population size, but also on the used hardware, and implementation details such as the chosen programming language and GA library. With the PyGAD library, our experiments (solving 10 MRHS systems with a population size of 1000) took approximately 6–7 minutes on a personal computer.



(A) tournament (B) steady-state

FIGURE 2. Comparison of tournament and steady-state selection methods and mutation rates (using 1 % elitism, 1000 generations and population size 1000).

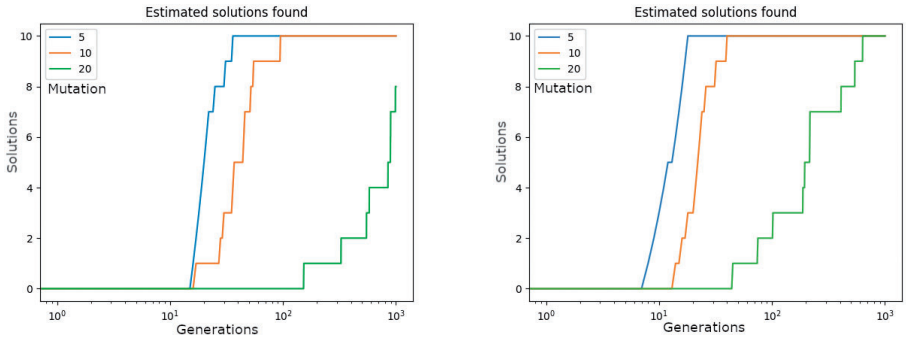


(A) random (B) roulette

FIGURE 3. Comparison of random and roulette selection methods and mutation rates (using 10 % elitism, 1000 generations and population size 1000).

We believe that it is possible to achieve significantly better running times with optimized implementation. A more fair comparison than running time is the explored space. Out of 2^{48} potential candidates, our experiments explored at most $1000 \times 5000 \approx 2^{22}$ candidates, and were able to find a solution even with smaller population and generation settings.

From the obtained results we have selected the following parameters as the optimal settings to solve sparse MRHS equations: 1 % elitism, 5 % gene mutation rate, population size 100, 1000 generations, and steady-state selection method (see Figure 5).



(A) tournament. (B) steady-state.

FIGURE 4. Comparison of tournament and steady-state selection methods and mutation rates (using 10 % elitism, 1000 generations and population size 1000).

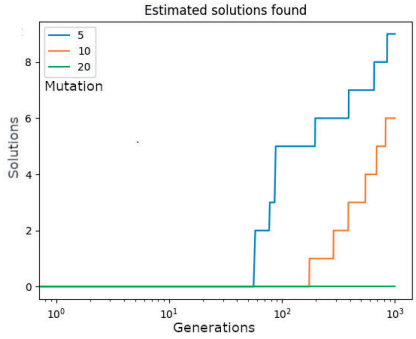
It is a good compromise between quality and computational (time) complexity. A larger population requires significantly longer running times with only a small improvement in success rate.

We repeated the original experiment with random MRHS systems with 48 unknowns using the selected (optimal) GA settings, but this time we examined 1000 MRHS system instances. We were using a dedicated computational server instead of a personal computer. The obtained results are in Figure 6. We were able to find the correct solution for $\approx 98\,\%$ of the systems, therefore we can state that GA with the selected parameters is effective for solving sparse MRHS systems.

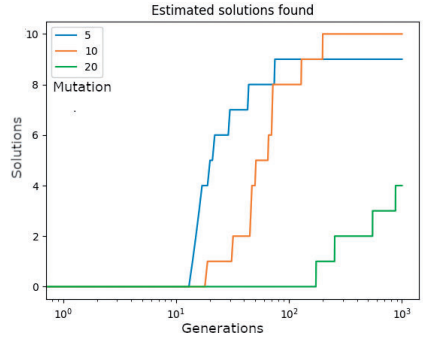
4.2. Effect of system size

Our next goal was to find out how much the number of unknowns affects the success rate of sparse MRHS systems. We tested sparse MRHS systems with 42, 45, 51, 54, and 100 unknowns, without changing the other parameters of the MRHS systems. We examined 1000 systems for each value of unknowns and used the best GA settings as before.

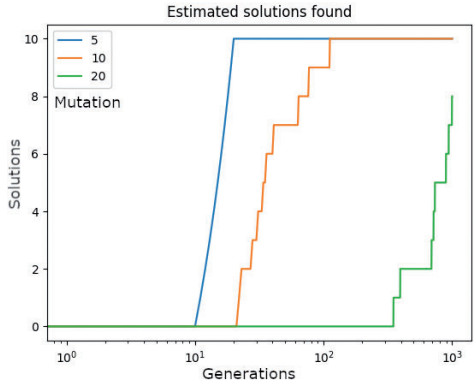
The results (see Figure 7) indicate that using a number of unknowns similar to the one we used to tune the GA parameters (range 42–54) does not affect the success rate. We were able to successfully solve almost all the examined MRHS systems, and the behavior of the used GA was almost the same. However, when we increased the number of unknowns significantly ("doubled" to 100 unknowns), the success rate also decreased significantly down to 30.9 %.



(A) steady-state, population size 10



(B) steady-state, population size 100

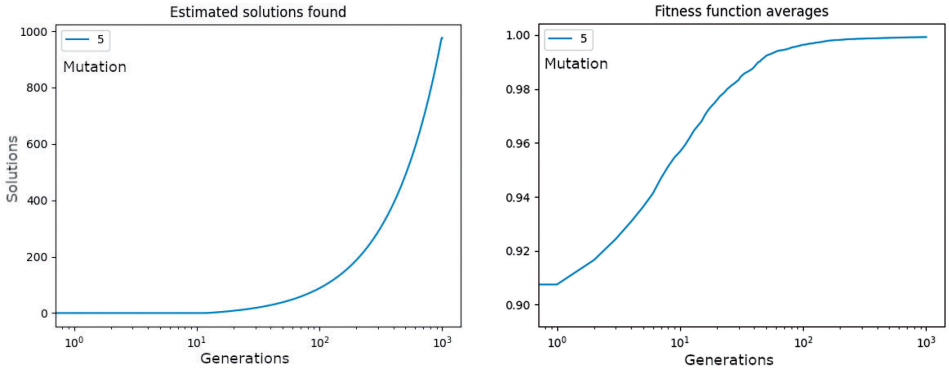


(C) steady-state, population size 1000

FIGURE 5. Dependence of success rate of the steady-state selection methods on the population size using 1 % elitism and 1000 generations.

4.3. Effect of system density

So far we have only solved sparse MRHS systems, which can be considered as simpler instances of MRHS problems. Therefore, in the next step, we focus on MRHS systems with higher density. We tested systems with density $d = 2, 3$, as well as systems with completely random joint matrix, and left the other parameters of the MRHS systems the same as it was in the initial experiment. We examined 1000 systems for each density and used the best GA settings as before.



(A) Success rate (B) Fitness

FIGURE 6. Final experiment results for the 1000 sparse MRHS systems.

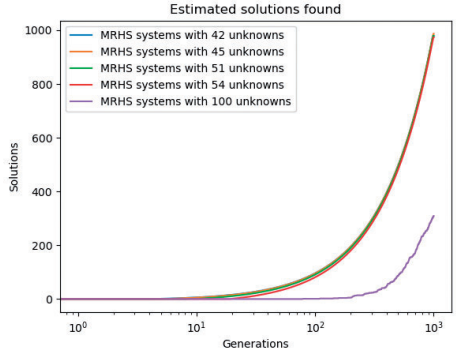
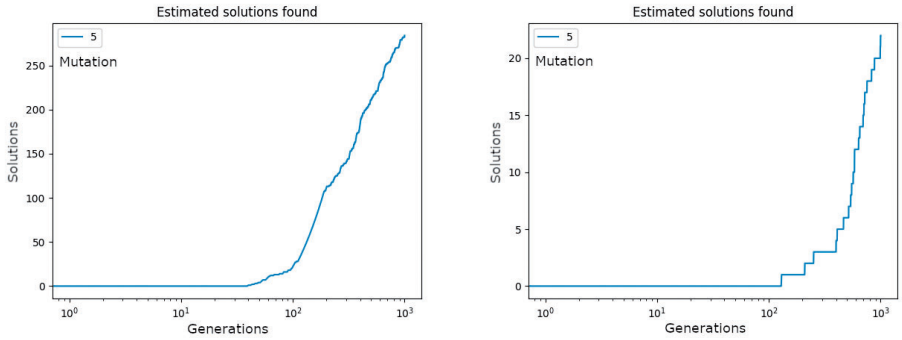


FIGURE 7. Comparison of success rates for sparse MRHS systems with 42, 45, 51, 54, and 100 unknowns (using steady-state selection with 1 % elitism, 1000 generations and population size 100).

The system density significantly affects the success rate of the used method (see Figure 8). In the case of $d = 2$, we were able to achieve the correct result only in the case of 28.4 % of the systems. Increasing the density to $d = 3$ reduced the success rate even more. The correct solutions were estimated only in the case of 2.2 % of the systems. We were not able to solve any of the tested dense random MRHS systems.

To increase the success rate, first, we increased the population size from 100 to 1000 chromosomes and kept the other parameters of the GA unchanged. For density $d = 2$ this increased the success rate to 46.3 %. In the case of $d = 3$, it tripled the success rate (still only 5.9 %). For dense random MRHS systems, we were still unable to obtain the correct solutions. The obtained results are in Figure 9.



(A) density $d = 2$ (B) density $d = 3$

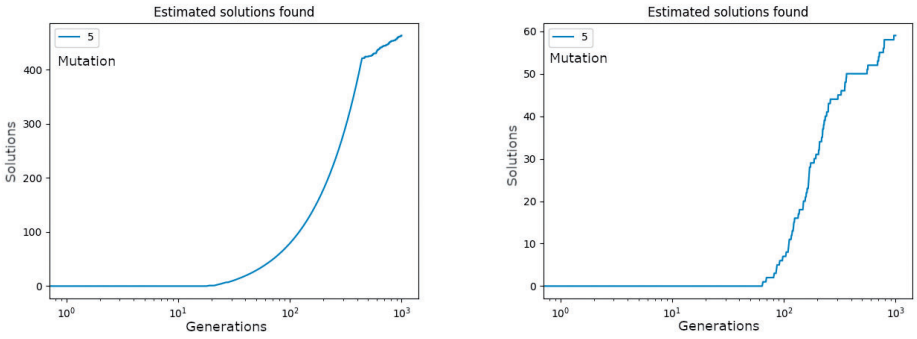
FIGURE 8. Experiment results for MRHS systems with density $d = 2,3$ using steady-state selection with population size 100 and 1000 generations.

We alternatively tried changing the number of generations from 1000 to 10 000 instead of changing the population size (keeping it to 100). Increasing the number of generations gives more time for the GA to converge. For $d = 2$, it increased the success rate up to 53.1 %, and for $d = 3$ up to 10.1 %. For dense random MRHS systems, we were still unable to obtain the correct solutions. The obtained results are in Figure 10.

Increasing the number of generations has a slightly better impact on the success rate than increasing the population size. We repeated the previous experiment with 25 000 generations. For $d = 2$, it increased the success rate up to 60 %, and for $d = 3$ up to 14 %. For dense random MRHS systems, we were still unable to obtain the correct solutions. The obtained results are in Figure 11.

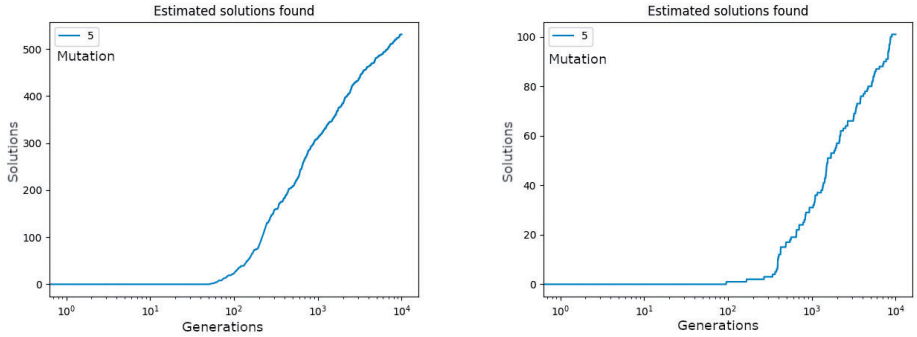
Please note that for one experiment (testing 1000 MRHS systems) with initial setting (1000 generations and population size 100) the computational time using a dedicated computational server[2] is approximately 40 minutes. However, increasing the population size to 1000 or increasing the number of generations to 10 000 increases the computation time of one experiment to approximately 8–8.5 hours. An experiment with a population size of 100 and 25 000 generations requires approximately 19.5–20 hours. We believe that with a combination of higher population size and generations, it is possible to increase the success rate even more, but this will require more computation power and further investigation.

[2]We ran the experiments on a machine with 12-core AMD Ryzen 9 5900X CPU (24 threads), 64 GB RAM and Ubuntu 20.04-1 operating system. Individual experiments were run on 20 threads.



(A) density $d = 2$ (B) density $d = 3$

FIGURE 9. Experiment results for MRHS systems with density $d = 2, 3$ using steady-state selection with population size 1000 and 1000 generations.



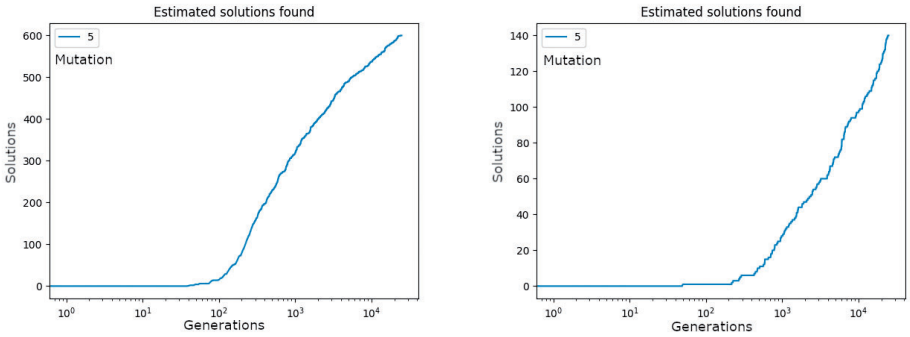
(A) density $d = 2$ (B) density $d = 3$

FIGURE 10. Experiment results for MRHS systems with density $d = 2, 3$ using steady-state selection with population size 100 and 10 000 generations.

5. Discussion

The main findings the experiments have shown can be summarized as follows:

- Sparse MRHS systems can be solved easily, and faster than expected. Using the correct GA setting, we were able to solve 98 % of the randomly generated sample systems with $n = 48$ unknowns using population size 100 and 1000 iterations. During the algorithm, we have thus explored only approximately 2^{17} points, out of the search space 2^{48}.



(A) density $d = 2$ (B) density $d = 3$

FIGURE 11. Experiment results for MRHS systems with density $d = 2, 3$ using steady--state selection with population size 100 and 25000 generations.

- Surprisingly, the number of unknowns had a very small impact on the performance of the GA for sparse MRHS systems. Even systems with 54 unknowns could be solved with the same success rate as systems with only 42 unknowns. However, this behavior is changed by significantly increasing the number of unknowns. We were able to achieve only a 31 % success rate in the case of systems with 100 unknowns.

  Note that the RZ solver [14], which is the best currently implemented deterministic solver of MRHS systems (written in C programming language) required on average 5908 seconds to solve the equations from the dataset of 100-variable MRHS systems, using (on average) approximately $2^{37.4}$ search operations. In practice, even an unoptimized genetic algorithm starts to outperform RZ solver on sparse MRHS systems that are large enough.

- The system density has a significant impact on the GA performance. When the system had on average $d = 2$ non-zero elements per column, the success rate decreased to 28 %, and for $d = 3$, the success rate was only 2 %. By increasing the population size and number of iterations we can improve these values to 60 %, and 14 %, respectively. This conforms to expectations that larger computational effort will increase the success rate. We still examined only approximately 2^{22} points of the search space (which is a significantly smaller fraction than an exhaustive search would require to obtain a similar success rate).

  Experiments show that increasing population size is preferable to increasing the number of generations. We suppose that it would be possible to fine-tune the parameters even more to solve almost all systems, but the computational effort involved would be too large. However, consider that potential attackers have

enough time to optimize GA parameters if it enables them to find the solution of the target system faster than the effort spent on parameter tuning.

- We were unable to solve random dense MRHS systems (with $n = 48$ unknowns) at all. This is an expected result, as in a random system each input bit influences essentially every MRHS equation in the system, and the fitness function becomes essentially randomly distributed with a single hidden global maximum.

  Note that in this research we have not examined the fitness landscape of the problem, but have tuned the parameters in an ad-hoc way. Fitness function analysis might help in tuning the parameters and fitness function in cases that are more difficult to solve.

Our research has confirmed that genetic algorithms are suitable for solving sparse MRHS systems. The performance decreases with increasing system density, and there seems to be a point when GAs would not be able to outperform an exhaustive search. It is an interesting open research question where exactly this point lies (and whether it depends on system size).

Note that we have examined only a specific family of MRHS systems with right-hand side sets of fixed size $l = 3, k = 4$, and random contents/matrix. In the future, we plan to examine also different MRHS families, including systems related to specific cipher designs.

Acknowledgement. We thank Peter Švec for his technical support.

Data availability.
The experimental data that support the findings of this study are publicly available at the following site:

https://github.com/zajacpa/mrhs-solver.

REFERENCES

[1] ADAMČEK, P.—LODERER, M.—ZAJAC, P.: *A comparison of local reduction and SAT-solver based algebraic cryptanalysis of JH and Keccak*, Tatra Mt. Math. Publ. **53** (2012), 1–20.

[2] ANGEL, E.—CAMPIGOTTO, R.—LAFOREST, C.: *Algorithms for the vertex cover problem on large graphs*, IBISC–Universited'Evry-Val d'Essonne Research report no. 1 (2010).

[3] BHASIN, H.—KUMAR, N.—MUNJAL, D.: *Hybrid genetic algorithm for maximum clique problem*, International Journal of Application of Innovation in Engineering & Management (2013).

[4] DE JONG, K. A.—SPEARS, W. M. et al.: *Using genetic algorithms to solve NP-complete problems.* In: *ICGA* (1989), pp. 124–132.

[5] DROGOUL, A.—DUBREUIL, C.: *A distributed approach to n-puzzle solving*. In: Proceedings of the Distributed Artificial Intelligence Workshop (1993).

[6] FAUSKANGER, S.—SEMAEV, I.: *Separable statistics and multidimensional linear cryptanalysis*, IACR Transactions on Symmetric Cryptology (2018), 79–110.

[7] GRASSI, L.—KALES, D.—RECHBERGER, C.—SCHOFNEGGER, M.: *Survey of key-recovery attacks on LowMC in a single plaintext/ciphertext scenario* (2020).

[8] HOLLAND, J. H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. Second edition, 1992.

[9] KUPKOVIČ, J.: *MRHS Solver Based on evolutionary algorithms*, MSc. Thesis, Slovak University of Technology in Bratislava 2023. (In Slovak)

[10] LACKO-BARTOŠOVÁ, L.: *Algebraic cryptanalysis of present based on the method of syllogisms*, Tatra Mt. Math. Publ. **53** (2012), 201–212.

[11] LARRANAGA, P.—KUIJPERS, C. M. H.— MURGA, R. H.—INZA, I.—DIZDAREVIC, S.: *Genetic algorithms for the travelling salesman problem: A review of representations and operators*, Artificial intelligence review **13** (1999), 129–170.

[12] MATHEIS, K.—STEINWANDT, R.—SUÁREZ CORONA, A.: *Algebraic properties of the block cipher DESL*, Symmetry **11** (2019), Paper 1411. https://doi.org/10.3390/sym11111411

[13] RADDUM, H.—SEMAEV, I.: *Solving multiple right hand sides linear equations*, Des. Codes and Cryptography **49** (2008), no. 1–3 147–160.

[14] RADDUM, H.—ZAJAC, P.: *MRHS solver based on linear algebra and exhaustive search*, J. Math. Cryptol. **12** (2018), 143–157.

[15] RADDUM, H.—SEMAEV, I.: *New technique for solving sparse equation systems*. Cryptology ePrint Archive, Paper 2006/475, 2006, https://eprint.iacr.org/2006/475.

[16] SEMAEV, I.: *New results in the linear cryptanalysis of DES* Cryptology ePrint Archive (2014).

[17] SMIČÍK, M.—ZAJAC, P.: *Algebraic cryptanalysis of Ascon using MRHS equations*, Tatra Mt. Math. Publ. **87** (2024), 1–24.

[18] SMIČÍK, M.—ZAJAC, P.: *Using MRHS solver for decoding problem — extended abstract*. Central European Conference on Cryptology 2024, Warsaw, 2024.

[19] TITO-CORRIOSO, O.—BORGES-QUINTANA, M.—BORGES-TRENARD, M. A.: *Improving search of solutions of MRHS systems using the genetic algorithm*, Revista Cubana de Ciencias Informáticas **17** (2023).

[20] WANG, R. L.: *A genetic algorithm for subset sum problem*, Neurocomputing **57** (2004), 463–468.

[21] ZAJAC, P.: *Solving trivium-based Boolean equations using the method of syllogisms*, Fund. Inform. **114** (2012), 359–373.

[22] ZAJAC, P.: *On solving sparse MRHS equations with bit-flipping*, Publ. Math. Debrecen **100** (2022), suppl. 8, 683–700.

[23] ZAJAC, P.: *Algebraic cryptanalysis with MRHS equations*, Cryptography **7** (2023), Paper 19, https://doi.org/10.3390/cryptography7020019

[24] ZAJAC, P.—ČAGALA, R.: *Local reduction and the algebraic cryptanalysis of the block cipher GOST*, Period. Math. Hungar. **65** (2012), 239–255.

[25] ZAJAC, P.—ŠPAČEK, P.: *A new type of signature scheme derived from a MRHS representation of a symmetric cipher*, Infocommunications Journal **11** (2019), 23–30.

Received August 8, 2025
Revised September 3, 2025
Accepted September 17, 2025
Publ. online December 20, 2025

Institute of Computer Science and Mathematics
Slovak University of Technology in Bratislava
Ilkovičova 3
841 04 Bratislava
SLOVAKIA

E-mail: eugen.antal@stuba.sk
pavol.zajac@stuba.sk
sabina.pekarekova@stuba.sk