Examination of crossover features in a Genetic Algorithm towards applications in cybersecurity

Hugo J. Delgado-Martí
Polytechnic University of Puerto Rico
hugojdm@gmail.com

Alfredo Cruz PhD
Polytechnic University of Puerto Rico
alcruz@pupr.edu

Abstract

Genetic Algorithms offer the benefit of parallelism by comparing populations of solutions and mimicking evolution in the survival of the fittest competition. Several applications of genetic algorithms in cybersecurity have emerged in recent literature, from cryptanalysis to IoT scheduling and network routing. GA has also been used in combinatorial optimization and as SAT solvers. Experiments were conducted to compare crossover features in a genetic algorithm SAT solver. Several new features were tested, such as age-based survival, neighborhood-based fitness, and duplicate pruning. These features helped maintain diversity while keeping convergence towards maximum fitness. Scaling the fitness function values using the mean and standard deviation of the population's fitness improved selection pressure during roulette selection. This implementation is focused on producing an efficient genetic algorithm SAT solver, which could be used in the cryptanalysis of block ciphers.

Keywords: Genetic Algorithms; Evolutionary Computation; SAT Solver; Crossover; GASAT.

Introduction

Several important problems in cybersecurity require optimization, modeling, and simulation. The computer scientist's task is to provide easy algorithms to find solutions in a reasonable amount of time, given restricted computing resources. Evolutionary algorithms and genetic computation may be alternative search methods for complex problems in computer science. Evolutionary computation stems from the search for optimization algorithms inspired by biology, particularly the process of natural evolution (Eiben & Smith, 2015, p. 13). Evolutionary and genetic algorithms can be interpreted as stochastic beam search techniques, where the algorithm tries to find a solution using an objective function to compare solutions (Russell et al., 2010, pp. 110–119) called the fitness function.

John Holland (1992) initially proposed Genetic Algorithms; in them, the Darwinian idea of survival of the fittest inspires the construction of fitness functions to find the optimal solution to a problem. Genetics has brought the idea that a phenotype, or visible characteristics of a species, is tied to its genotype, the genetic code in DNA. In contrast, Genetic Algorithms represent the genes as bits or segments of code, while the phenotype is the characteristics measured by its fitness. Genetic algorithms suit complex systems where conflicting or interacting objectives may produce different solutions or outcomes.

Genetic Algorithms have found applications in "routing, scheduling, adaptive control, game playing, cognitive modeling, transportation problems, the traveling salesman problem, optimal control problems, database query optimization, etc." (Michalewicz, 1996, p. 15). Recently, GAs have seen several implementations in optimization related to physical security, for example, in Baraklı et al. (2023) genetic algorithms were used for threat evaluation in the context of weapon assignment problems. Regarding network design, genetic algorithms considered conflicting interests such as user demand, placement, and routing of equipment and cables (Correia et al., 2023). Baushchert (2019) proposed using GA to find

optimal solutions in constructing a 5G virtual network, which was later improved by an exact solution search using mathematical programming.

Genetic Algorithms and Cybersecurity

Several recent applications of evolutionary algorithms in cybersecurity can be enumerated. Saeed et al. (2023) showed that genetic algorithms improved energy and resource allocation in the problem of Internet of Things (IoT) workflow scheduling. Their results outperformed standard algorithms in workflow scheduling. Devecci et al. (2023) applied an evolutionary algorithm for intrusion detection in a low-power lossy network of IoT devices. Yilmaz and Sen (2019) used grammatical evolution in the early detection of botnets, obtaining better results than previous methods.

Block ciphers are used extensively in lightweight cryptography as authentication protocols. Applications of Genetic Algorithms in cryptography could help improve cryptographic primitives and protocols through cryptanalysis of block ciphers (Pavlenko et al., 2019; Tito-Corrioso et al., 2023). Expressing the structure of the selected cryptographic implementation as a satisfiability problem (SAT) may provide the context for evaluating the implementation using Genetic Algorithms. The knowledge about a cipher, in combination with techniques such as guess-and-determine, is written into equations representing the relation between ciphertext-plaintext pairs. Algebraic cryptanalysis, for example, describes the structure and information about a cipher as a set of polynomial equations that can then be solved as a satisfiability problem (Bard, 2009; Courtois et al., 2008; Pavlenko et al., 2019). It has proven an effective method in the attack of lightweight block ciphers. Since algebraic equations representing a cipher can be transformed into a Satisfiability problem, it can be fed to a Genetic Algorithm to find the key or the plaintext.

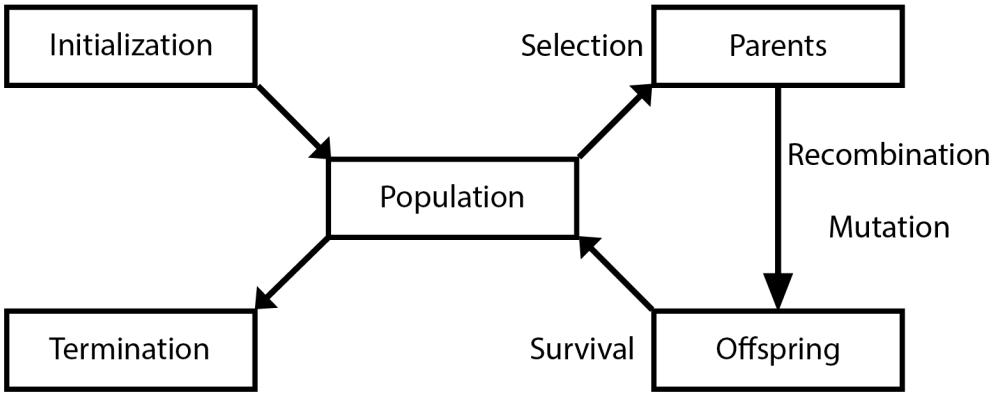


Figure 1 General flowchart of a Genetic Algorithm

Figure 1 represents a general overview of the steps in a Genetic Algorithm. Several options exist for selection, recombination, mutation, and survival. In satisfiability problems, the individuals are bit strings representing the true or false assignment to each variable in the equation.

Satisfiability and Genetic Algorithms

The full set of words of fixed size k using the Boolean alphabet is also called the hypercube of dimension k. Formally represented as $\{0,1\}^k, (k \geq 1)$ for $k \in N$, each word can also be called a vector of length k lying in the hypercube. Discrete functions that take such vectors as input are called Boolean functions of arity k, where k represents the number of variables in the function. The combination of conjunction, disjunction, and negation is a complete set of operators for representing these functions. Respectively, these operators are represented by $\wedge, \vee, \neg$. These symbols represent the logical operators AND, OR, and NOT.

Conjunctive Normal Forms (CNF) are defined as a conjunction of clauses. Clauses are Boolean functions consisting of disjunctions of literals without complementary pairs. A complimentary pair consists of a variable and its negation. Therefore, a disjunction of a complementary pair is a tautology. Literals is another term for the variables in the Boolean formula. A Boolean formula (see Equation 1) with at least one

vector that produces a true output is called satisfiable; if no vector returns true, it is called unsatisfiable. The problem of determining if a Boolean function has a satisfiable assignment is called the Boolean Satisfiability Problem or SAT (Nilsson, 1998, pp. 25-35;217-238).

The SAT problem is NP-complete, and many computer science problems can be reduced to satisfiability problems in polynomial time (Sipser, 2013, p. 304). Research has been conducted using SAT solvers and Genetic Algorithms, proving their significance in many computer science and security areas.

A potential application of genetic algorithms as SAT solvers lies in algebraic cryptanalysis since it represents knowledge about a cryptosystem as a set of polynomial equations under the Galois field of 2nd order, which can be represented as SAT. Algebraic cryptanalysis has been effective against block ciphers such as KeeLoq, an intelligent car key authentication protocol (Bard, 2009).

Methods

There is an indefinite number of specifications on applying the general idea behind a Genetic Algorithm. First is the encoding mechanism; it establishes how to represent the individuals so the program can successfully evolve, mutate, cross genes, and measure their fitness. The mixing number specifies the number of parents from a generation to create offspring in the next generation. A selection process defines the method of choosing the parents, which can be random or mediated by their fitness. During recombination or mating, the genome from one parent is combined with another. The mutation rate establishes how often random mutations will happen, which is necessary to prevent fast convergence. Elitism is the method used to guarantee an increase in the total fitness of the population. At the same time, a minimum threshold can be used to discard low-fitness sections of the population (Russell et al., 2010, p. 116).

Data: A set of Conjunctive Normal Form Clauses $\phi, Maxflip, MaxNbCrossovers$
Result: the best truth assignment to the 3-SAT formula
Begin
CreatePopulation(P)
$NbCrossovers \leftarrow 0$
While no $X \in P$ satisfies ϕ and $NbCrossovers < MaxNbCrossovers$ **do**
/* Selection */
$P' \leftarrow Select(P, NbInd)$
Choose $X, Y \in P'$
/* Crossover */
$Z \leftarrow Crossover(X, Y)$
/* Insertion condition of the child */
$P \leftarrow Replace(Z, P)$
$NbCrossovers \leftarrow NbCrossovers + 1$
If there exists $X \in P$ satisfying ϕ **then**
Return the corresponding assignment.
Else
Return the best assignment found.

Algorithm 1 GASAT Algorithm (following Lardeux et al., 2006 without Taboo Search)**.**

The fitness function is particularly special among the different constituents of a Genetic Algorithm since it represents the population's requirements to adapt (Eiben & Smith, 2015, p. 30). The fitness function assigns numerical values to the qualities of the individuals in the population such that the individuals can be measured objectively. Nilsson (1998, pp. 59–60) describes the fitness function as the mathematical landscape where population individuals reproduce. In the topographic analogy, individuals at higher elevations have a higher chance of reproduction. Strategies are needed to maintain diversity within the population until a satisfying solution is found. The approach proposed in this paper is to develop a genetic algorithm for a SAT problem. A sample Genetic Algorithm presented by Lardeux et al (2006) is shown.

In Algorithm 1 (see also Figure 1), the population P is created randomly as binary strings. The population is tested using the SAT formula in CNF, and the number of true clauses is counted. The individual's fitness is established by the number of satisfied clauses (TRUE). A population sample is chosen for mating, to produce the offspring. A set of new individuals is produced using the crossover function where the genome from both parents is mixed. The new generation is composed of a combination of parents and their offspring. The program will continue until the maximum number of generations is reached or if stagnation is detected by comparing the change in the average fitness in recent generations to the whole evolution. The implementation presented in this paper used binary word representation for the individuals with a word length of 10 bits. The CNF used to represent the SAT problem is presented in Equation 1:

$$
\begin{aligned}
F(X_{1-10}) = {} & (\neg X_2 \vee \neg X_3 \vee \neg X_4 \vee X_5 \vee \neg X_6 \vee X_7 \vee \neg X_8 \vee \neg X_9) \wedge (X_1 \vee X_2 \vee X_4 \vee \neg X_5 \vee \neg X_6 \vee X_8) \\
& \wedge (\neg X_3 \vee \neg X_5 \vee \neg X_{10}) \wedge (X_4) \wedge (X_1 \vee X_2 \vee \neg X_3 \vee X_4 \vee X_6 \vee \neg X_7 \vee \neg X_9 \vee X_{10}) \\
& \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4 \vee X_5 \vee \neg X_6 \vee X_7 \vee \neg X_8 \vee \neg X_{10}) \wedge (\neg X_2 \vee \neg X_{10}) \\
& \wedge (\neg X_1 \vee X_2 \vee \neg X_3 \vee X_5 \vee X_6 \vee X_7 \vee \neg X_8 \vee \neg X_9 \vee \neg X_{10}) \wedge (X_2 \vee X_{10}) \\
& \wedge (X_1 \vee X_3 \vee \neg X_5 \vee X_6 \vee X_7 \vee \neg X_8 \vee X_9 \vee X_{10}) \\
& \wedge (X_2 \vee \neg X_3 \vee \neg X_4 \vee \neg X_5 \vee X_6 \vee \neg X_7 \vee \neg X_8 \vee X_9 \vee X_{10}) \wedge (\neg X_1 \vee \neg X_5 \vee \neg X_6 \vee X_8 \vee X_9) \\
& \wedge (X_1) \wedge (\neg X_4 \vee X_9 \vee \neg X_{10}) \wedge (\neg X_1 \vee X_3 \vee \neg X_4 \vee \neg X_5 \vee \neg X_6 \vee X_8 \vee \neg X_9 \vee \neg X_{10}) \\
& \wedge (X_3 \vee X_4 \vee \neg X_9 \vee X_{10}) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee X_5 \vee X_7 \vee X_8 \vee X_9 \vee \neg X_{10}) \wedge (\neg X_9) \\
& \wedge (\neg X_2 \vee \neg X_5 \vee \neg X_6 \vee \neg X_9 \vee \neg X_{10}) \wedge (\neg X_1 \vee \neg X_3 \vee \neg X_7 \vee X_{10})
\end{aligned}
$$

Equation 1

This formula contains a total of 10 variables and 20 clauses. The function is satisfied if a given variable assignment returns true on all 20 clauses. Each clause has between 1 to 9 variables present. No variable is present more than once in each clause. The function has 1024 respective variable assignments, and the genetic algorithm's task is to find satisfying solutions.

Roulette wheel selection (see Figure 2 and Algorithm 2) was chosen as the selection method for this investigation. During roulette wheel selection, individuals are ranked by fitness, and a cumulative probability distribution is calculated. First, the total sum of the fitness of the population is calculated. Then, to calculate the cumulative fitness for each member of the population, their fitness is divided by the total fitness and then added to the previous member's cumulative fitness. This method returns a cumulative probability, ranging from 0 to 1 so that all members have a probability of being chosen. Each member will then be assigned a probability range whose maximum is its cumulative probability and the minimum is the previous member's cumulative fitness. Members with higher fitness have a broader range because their fractional weight is larger; therefore, they have higher probabilities.

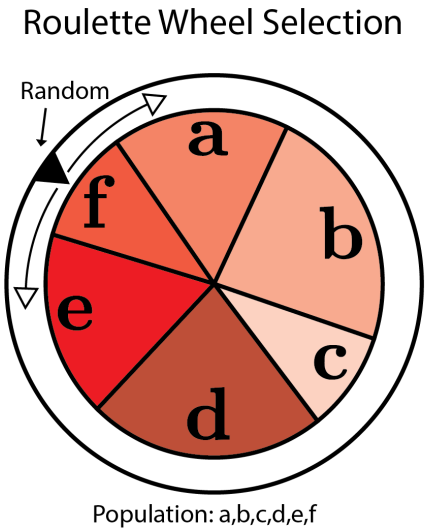


Figure 2 Random methods used during selection.

In Figure 2 a random number between 0 and 1 is chosen and the genome with the cumulative value greater than the random number is the selected individual for reproduction. In Algorithm 2 the selection method is presented once the cumulative distribution has been calculated for all individuals. The roulette wheel chooses individuals until the desired sample size is achieved using the crossover rate and the population size.

Data: A population, P, a cumulative distribution, D, and a sample size S.
Result: A random sample, P'
Begin:
  While $P'_{size} < S$ **do:**
    Choose a random number, R

$$0 \leq R \leq P_{size}$$

    While $R < D[i]$ **do:**

$$i \leftarrow i + 1$$

      Add the element to the sample:

$$P' \leftarrow P[i]$$

Return: P'

Algorithm 2 Roulette Selection

Populations of 8, 16, and 32 random genomes were created before running the algorithm. For each population size, the crossover rate was set to 80% or 100%, and mutation rates of 1/100 or 1/1000 were tested. Several additional features were attempted as part of the experiments. The oldest individual from each generation is removed after 10 consecutive iterations. This feature helps maintain diversity in the population, although part of its genome may remain distributed among other individuals. Three crossover, one-point, two-point, and uniform methods were compared (see Figure 3). In a one-point crossover, a single reference point cuts each parent's genome to produce two offspring. In the two-point crossover, three genome segments are shared between the two offspring, each having the beginning and end of one parent and the middle genetic material of the other parent. In uniform crossover, a random template is chosen to share genes from both parents. Each method produces two offspring from two parents. The experiments were carried over 100 times and their averaged results are presented in Table 1.

Several experiments found that the diversity may stagnate after a few generations when the algorithm falls into local maxima. To avoid premature convergence, duplicate offspring were pruned before adding to the population during the survival phase. If the resulting population was greater than the desired population size, smaller fitness individuals were removed; on the other hand, new random individuals were added if the resulting population was smaller than the population size.

This implementation was programmed in Python version 3.11.7 and run in a Jupyter Notebook running on MacOS Sonoma version 14.2.1. The system is a MacBook Pro 2 GHz Quad-Core Intel Core i5 with 16 GB 3733 MHz RAM.

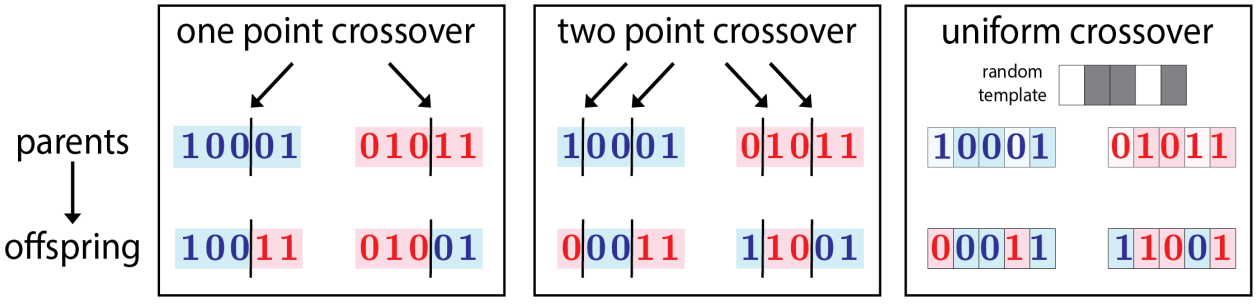


Figure 3 The effect of one-point, two-point, and uniform crossovers on the parent-offspring genomes.

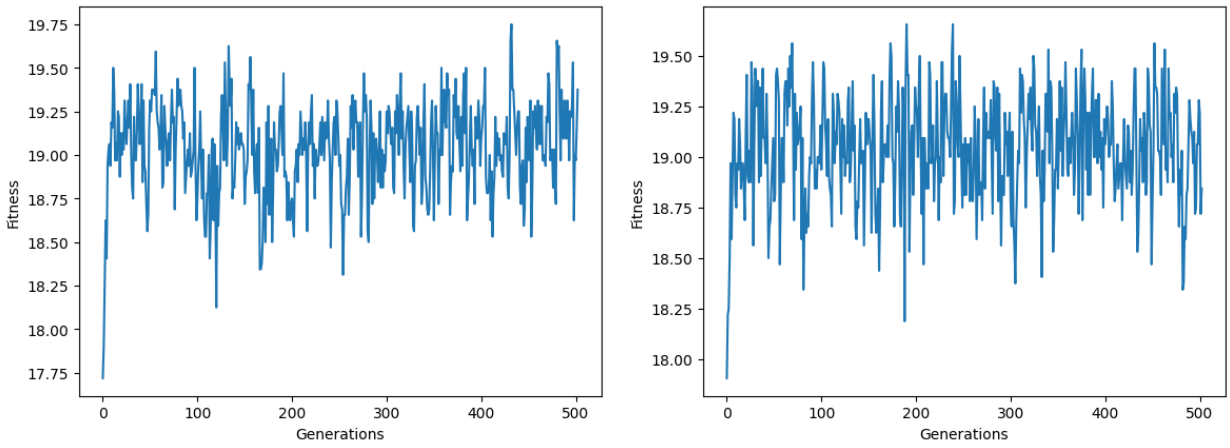
Results

Table 1 presents the results of the experiments. In the first column, the population number is presented. This number represents the number of random members chosen at the beginning of the algorithm. CX type refers to the type of crossover, either one-point, two-point, or uniform crossover, as described in the previous section. The mutation rate indicates the probability that a given offspring will mutate its genome before the next generation is chosen. CPU time is the number of seconds the algorithm takes before stopping, either because the maximum number of generations is achieved or because the moving average fitness is not changing significantly. The column marked generations indicates the average generation during which the algorithm stopped. Notice that stagnation detection starts at the 500th generation. The last column, Solutions, represents the average number of solutions with the given features and parameters.

The results presented in Table 1 show that larger populations produce more results regardless of other features but require longer running times. Using a population of 16 or 32 individuals, the 21 satisfying solutions of the CNF were found soon after the 500th generation when the stagnation detection feature began. A method must be devised to stop the algorithm in a real-case scenario where the number of solutions is unknown. In a cryptanalytic scenario, finding at least one solution might be enough, so the opposite case should also be asked: when should the algorithm stop if no solution is found?

The type of crossover seems to have little effect on the number of results or the CPU times. Since crossover mixes the parents' genetic information, any reasonable combination of genes will produce a reasonable combination for the offspring after crossover. This might not be the case in all CNFs because the solution structure may require combinations that can be achieved with a particular crossover method. For example, a two-point crossover may cut a specific section in the middle of a genome that improves a solution, whereas a uniform or one-point crossover may not.

The crossover rate seems to be the dominant feature in these experiments; a higher crossover rate, such as 80%, yields similar results in terms of solutions than a 60% rate but in a smaller CPU time. The crossover rate, therefore, seems to be the dominant feature when the initial population is small (8) but has less significance when the population is large (16, 32). The best result overall had 8 items in the initial population, 1/100 mutation, and an 80% crossover rate.



(a) (b)

Figure 4 Sample progress curves.

In Figure 4, two sample progress curves show the average fitness per generation for a trial with an initial population of 32 individuals, a crossover rate of 80%, and a one-point crossover. In (a), the mutation rate was set to 0.01, while in (b), the mutation rate was kept at 0.001. The average fitness shows variability because it combines the fitness of the best and worst members of the population in each generation. Due to the random nature of the crossover points, the average may not always improve.

Table 1 Comparison of results for different initial populations, crossover, and mutation

Initial Population	CX Type	CX rate	Mutation Rate	CPU Time (*s*)	Generations	Solutions
8	1 point	0.80	1/100	0.295	501.15	20.89
			1/1000	0.301	501.20	20.91
		0.60	1/100	0.445	547.07	20.93
			1/1000	0.420	534.89	20.97
	2 points	0.80	1/100	0.315	502.76	20.87
			1/1000	0.313	501.56	20.89
		0.60	1/100	0.464	503.49	20.88
			1/1000	0.376	504.49	20.85
	Uniform	0.80	1/100	0.303	501.53	20.80
			1/1000	0.295	501.67	20.65
		0.60	1/100	0.374	503.18	20.80
			1/1000	0.383	504.11	20.86
16	1 point	0.80	1/100	0.499	509.20	21
			1/1000	0.499	511.22	21
		0.60	1/100	0.606	503.66	21
			1/1000	0.676	553.23	21
	2 points	0.80	1/100	0.587	552.97	21
			1/1000	0.515	512.65	21
		0.60	1/100	0.643	508.42	21
			1/1000	0.657	512.55	21
	Uniform	0.80	1/100	0.471	513.12	21
			1/1000	0.469	511.80	21
		0.60	1/100	0.610	507.06	21
			1/1000	0.617	513.47	21
32	1 point	0.80	1/100	0.951	551.03	21
			1/1000	0.854	515.27	21
		0.60	1/100	1.128	529.91	21
			1/1000	1.264	589.95	21
	2 points	0.80	1/100	1.030	509.52	21
			1/1000	1.047	507.11	21
		0.60	1/100	1.280	523.48	21
			1/1000	1.192	512.33	21
	Uniform	0.80	1/100	0.997	513.84	21
			1/1000	0.880	521.72	21
		0.60	1/100	1.226	563.74	21
			1/1000	1.139	534.09	21

Conclusion

As the number of variables and clauses increases, we may need to increase the initial population to cover the solution space efficiently. Increasing the population size also increases the runtime. A balance between these two factors must be achieved in future studies. Some of the results in this investigation may be CNF-dependent, so further tests are required to test and benchmark this implementation of the Genetic Algorithm. Regardless, genetic algorithms present a diversified set of tools that can be optimized for finding solutions and require further study. Genetic Algorithms are an effective stochastic alternative for finding solutions to difficult problems that can be expressed as SAT equations. Degré et al. (2024) used a man-in-the-middle attack to improve the SAT modeling of a reduced round Ascon-hash a lightweight cipher proposed for authenticated encryption and hashing. The algebraic degree of 2 of its substitution boxes makes the cipher prone to algebraic attacks using SAT solvers and therefore make them a candidate to be studied with Genetic Algorithms.

The neglectable dependence on the number of solutions detected and the configuration parameters makes genetic algorithms as SAT solvers a potential alternative. This helps optimize the best choice of parameters and configuration of the SAT solver to the one with the least time complexity. Time constraints are a priority when dealing with large CNFs such as those found in cryptanalysis.

Future Work

Several ideas emerged during this work that require validation by isolating each modification of the genetic algorithm and comparing the speedup obtained to a simple algorithm implementation. Preliminary results show that a neighborhood search, all solutions with a hamming distance of 1 to the current individual, may yield a higher number of solutions in a significantly smaller number of generations and run time. This idea led to the proposition of using a decimal fitness function that averages over the fitness of each neighbor. Still, the algorithm needs to be refined so that increasing the problem size does not affect running time. Other features that adapt the search to the genetic algorithm conditions need to be considered, such as dynamic parameter settings or methods for stagnation detection.

In terms of selection, several other methods need to be compared, such as sigma-scaled selection, where the fitness is rescaled using the average and the standard deviation, and stochastic selection, where the individuals are split into segments and chosen at equal intervals but with the first individual selected randomly using a probability distribution. Future studies must also explore Tabu search and other local search methods.

Genetic algorithms have shown versatility in their implementation and application. Cryptanalytic methods require resource-intensive computation where large data sets are analyzed several times to solve the cryptogram. Heuristic searches, parallelism, and proper representation of the cryptographic structure may be used in a GA application for cryptanalysis. Expressing the structure of a cryptographic implementation as a satisfiability problem (SAT) may provide the context for cryptanalysis using Genetic Algorithms. Algebraic cryptanalysis has proven effective towards lightweight block ciphers such as KeeLoq, an intelligent car-key encryption protocol. Genetic Algorithms have been used as SAT solvers in several applications and have shown better performance in some instances than other random search implementations. Ascon (Dobraunig et al., 2021), the lightweight authenticated encryption and hash function family, has been cryptanalyzed using SAT solvers (Baek et al., 2024; Degré et al., 2024). An application of genetic algorithms to the cryptanalysis of Ascon remains to be investigated.

While block ciphers are considered safe from the quantum threat, quantum computers can solve SAT problems in polynomial time but may require more qubits and gates than are available today. Nevertheless, genetic algorithms and parallel computation may pose a threat to block ciphers.

Acknowledgment

This work was possible thanks to the PUPR PPOHA Research Assistantship Grant.

References

Baek, S., Kim, G., & Kim, J. (2024). Preimage Attacks on Reduced-Round Ascomn-Xof*. *Cryptology EPrint Archive*, *298*, 1–25. https://eprint.iacr.org/2024/298

Baraklı, A. B., Semiz, F., & Atasoy, E. (2023). The Specialized Threat Evaluation and Weapon Target Assignment Problem: Genetic Algorithm Optimization and ILP Model Solution. In J. Correia;, S. Smith, & Raneem Qaddoura (Eds.), *Applications of Evolutionary Computation 26th European Conference, EvoApplications 2023. Brno, Czech Republic, April 12–14, 2023* (pp. 19–34). Springer International Publishing. https://doi.org/10.1007/978-3-031-30229-9_2

Bard, G. V. (2009). *Algebraic Cryptanalysis*. Springer US. https://doi.org/10.1007/978-0-387-88757-9

Bauschert, T., D'Andreagiovanni, F., Kassler, A., & Wang, C. (2019). A Matheuristic for Green and Robust 5G Virtual Network Function Placement. In P. Kaufmann; & P. A. Castillo (Eds.), *22nd International Conference, EvoApplications 2019 Held as Part of EvoStar 2019 Leipzig, Germany, April 24–26, 2019* (pp. 430–438). Springer Nature Switzerland. https://doi.org/10.1007/978-3-030-16692-2_29

Correia, J., Gama, G., Guerrinha, J. T., Cadime, R., Antero Carvalhido, P., Vieira, T., & Lourenço, N. (2023). Automatic Design of Telecom Networks with Genetic Algorithms. In J. Correia;, S. Smith, & R. Qaddoura (Eds.), *Applications of Evolutionary Computation 26th European Conference, EvoApplications 2023. Brno, Czech Republic, April 12–14, 2023* (pp. 269–284). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-30229-9_18

Courtois, N. T., Bard, G. V., & Wagner, D. (2008). Algebraic and Slide Attacks on KeeLoq. In *Fast Software Encryption: Vol. 5086 LNCS* (pp. 97–115). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-71039-4_6

Degré, M., Derbez, P., Lahaye, L., & Schrottenloher, A. (2024). New Models for the Cryptanalysis of ASCON. *Cryptology EPrint Archive*, *298*, 1–21.

Deveci, A., Yilmaz, S., & Sen, S. (2023). Evolving Lightweight Intrusion Detection Systems for RPL-Based Internet of Things. In J. Correia, S. Smith, & R. Qaddoura (Eds.), *26th European Conference, EvoApplications 2023 Held as Part of EvoStar 2023 Brno, Czech Republic, April 12–14, 2023* (pp. 177–193). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-30229-9_12

Dobraunig, C., Eichlseder, M., Mendel, F., & Schläffer, M. (2021). Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *Journal of Cryptology*, *34*(3), 33. https://doi.org/10.1007/s00145-021-09398-9

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44874-8

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. (2nd ed.). MIT Press.

Lardeux, F., Saubion, F., & Hao, J.-K. (2006). GASAT: A Genetic Local Search Algorithm for the Satisfiability Problem. *Evolutionary Computation*, *14*(2), 223–253. https://doi.org/10.1162/evco.2006.14.2.223

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs* (3rd ed.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-03315-9

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kauffmann.

Pavlenko, A., Semenov, A., & Ulyantsev, V. (2019). Evolutionary Computation Techniques for Constructing SAT-Based Attacks in Algebraic Cryptanalysis. In P. Kaufmann, P., Castillo (Ed.), *Applications of Evolutionary Computation. Evo Applications 2019* (Vol. 11454, pp. 237–253). Springer International Publishing. https://doi.org/10.1007/978-3-030-16692-2_16

Russell, S. J., Norvig, P., Davis, E., Hay, N. J., & Sahami, M. (2010). *Artificial Intelligence, A Modern Approach* (3rd Int. V).

Saeed, A., Chen, G., Ma, H., & Fu, Q. (2023). A Memetic Genetic Algorithm for Optimal IoT Workflow Scheduling. In J. Correia, S. Smith, & R. Qaddoura (Eds.), *26th European Conference, EvoApplications 2023 Held as Part of EvoStar 2023 Brno, Czech Republic, April 12–14, 2023* (pp. 556–572). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-30229-9_36

Sipser, M. (2013). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.

Tito-Corrioso, O., Borges-Quintana, M., Borges-Trenard, M. A., Rojas, O., & Sosa-Gómez, G. (2023). On the Fitness Functions Involved in Genetic Algorithms and the Cryptanalysis of Block Ciphers. *Entropy*, *25*(2), 1–13. https://doi.org/10.3390/e25020261

Yilmaz, S., & Sen, S. (2019). Early Detection of Botnet Activities Using Grammatical Evolution. In P. Kaufmann & P. A. Castillo (Eds.), *22nd International Conference, EvoApplications 2019 Held as Part of EvoStar 2019 Leipzig, Germany, April 24–26, 2019* (pp. 395–404). Springer Nature Switzerland. https://doi.org/10.1007/978-3-030-16692-2_26