Solving for the RC4 stream cipher state register using a genetic algorithm

Benjamin Ferriman
School of Computer Science
University of Guelph
Guelph, ON N1L 1L9

Charlie Obimbo
School of Computer Science
University of Guelph
Guelph, ON N1L 1L9

Abstract—**The RC4 stream cipher has shown to be quite resilient to cryptanalysis for the 26 years it has been around. The algorithm is still one of the most widely used methods of encryption over the Internet today being implemented through the Secure Socket Layer and Transport Layer Security protocols. Genetic algorithms are a sub-class of evolutionary algorithms that have been used to help solve many different problems of optimization in a variety of disciplines. In this paper we will examine the abilities of the genetic algorithm as a tool to help solve the permutation that is stored as the state register of the RC4 stream cipher. Finally, we will show that on average the genetic algorithm can solve 100% of the keystream in $2^{121.5}$ generations.**

I. INTRODUCTION

Over the past twenty years the Internet has evolved astronomically as a tool for education, pleasure, and economics, to name a few applications. In today's society there are very little tasks in one's daily life which are not facilitated by the Internet in some way, shape, or form. As the services available over the Internet continue to expand, new and old problems of security arise and must be accounted for in order to properly facilitate these applications.

One of the largest sectors which continues to grow is on-line banking and other electronic financial transactions (conveniently distinguished as E-Commerce). These two applications face many of the same problems as traditional physical banking, but also a new set of challenges that have amounted due to the use of the Internet. The most obvious contemporary issue is that of *communication*. Traditionally a customer simply communicated with a bank teller where the environment could be controlled as well as the manor of communication (i.e. whether something could simply be conveyed through speech or read privately by the customer). With the advent of on-line banking there is an unknown communication between the customer (client computer) and the teller (bank servers). The very fact that on-line banking improves the ease of use for a customer by virtually letting them do their banking anywhere with an Internet connection also hinders their ability to know specifically how their private communication with the bank system is being conducted.

Besides this, there it also the convenient and ubiquitous use of mobile computing. With advent of smart-phones, the main use of the Internet is quickly shifting to being used mainly in the mobile computing environment. According to PewResearch [?] as of May 2013, 63% of adult cell owners use their phones to go online and 34% of cell internet users go online mostly using their phones, and not using some other device such as a desktop or laptop computer. As can also be seen on Table I, obtained from the United States Census Bureau Data [?], the younger population, between the ages of 10 and 90 comprise over 60% of the population of the US, and according to PewResearch, as can be seen on Table II, about three-quarters of these have and use Smartphones.

TABLE I: Demographics of the US population, 2012

Age	Population	Percentage	Cummulative Percentage
All ages	308,827	100.0%	
Under 5	20,110	6.5%	6.5%
5 - 9	20,416	6.6%	13.1%
10 - 14	20,605	6.7%	19.8%
15 - 19	21,239	6.9%	26.7%
20 - 49	124,607	40.3%	67.0%
50 - 59	42,842	13.9%	80.9%
60 - 64	17,501	5.7%	86.6%
65 & older	41,506	13.4%	100.0%

TABLE II: Smartphone owners in 2014 [?]

	Have a smartphone
All Adults	**58%**
Gender	
a. Men	61%
b. Women	57%
Race	
a. White	53%
b. African American	69%
c. Hispanics	61%
Age Group	
a. 18 - 29	83%
b. 30 - 49	74%
c. 50 - 64	49%
d. 65+	19%

With new attacks on Internet-based encryption protocols coming to light in the past four months, a lot of focus has shifted from traditional forms of cryptanalysis to methods of circumvention to attack these ciphers. One cipher that is still widely used and investigated is the RC4 stream cipher. Due to it's simplicity and robustness (efficient for both software and hardware) [1], the RC4 stream cipher is one of the most implemented encryption schemes online and over computer networks. It's usage is seen in the *Secure Socket Layer (*SSL) [2] and *Transport Layer Security (*TLS) [3] protocols as well as the now obsolete *Wired Equivalent Privacy (*WEP) [4] protocol. It is also used in *Wi-Fi Protected Access (*WPA and WPA 2) protocols (when TKIP is not selected by default).

In recent years, focus has been drawn to implementing *genetic algorithms* as a tool for cryptanalysis. These tools, from the evolutionary algorithms family, have been used in the past to help solve permutation problems such as the traveling salesman problem ([5], [6], [7], [8]). Since the state register of RC4 (see Section I-A1) is a permutation, the researchers would like to investigate the effectiveness of using a genetic algorithm to attempt and solve the permutation sequence of the RC4 state register.

This paper is organized into several sections to present our findings. We will first present the reader with a background of the RC4 stream cipher and genetic algorithms in Sections I-A and I-B respectively. Next, we will propose an implementation of a genetic algorithm in Section II. This will be followed up with an examination and discussion of the results of our experiments in Section III. Finally, we will conclude our findings and present any future avenues of research in Sections IV and V respectively.

The following paper proposes a genetic algorithm to try and solve for the state register permutation of the RC4 stream cipher. The operators that will be investigated include: partially mapped crossover, edge recombination crossover, swap mutation, and inversion mutation. In addition, an *adaptive mutation method* will be utilized in order to reduce the occurrence of a candidate solution becoming stuck in a local optimum over the vast search space. Finally, this paper will show that on average the genetic algorithm will be able to discover 100% of the keystream and replicate the state register in $2^{121.5}$ generations.

A. RC4 Overview

The RC4 stream cipher was invented by Ron Rivest in 1987 while working at RSA Security and designed as a *non-linear feedback shift register* (non-LFSR). It is a stream cipher meaning: given identical initialization keys, the algorithm will produce the same keystream for all parties involved in the communication. RC4 allows the initialization key $\mathcal{K}$ to be of length 40 to 2048 bits. The cipher produces a keystream z of word size n from a state register S of size 2^n consisting of all bit permutations of a n-bit word. Two word sized index pointers, i and j, are used to help perform permutations on the register S. Generally RC4 is implemented with n = 8 bit words.

There are two algorithms that make up RC4. The first algorithm is called the *key scheduling algorithm* (*KSA*) and is used in conjunction with $\mathcal{K}$ to initialize the shift register S into a pseudo-random ordering (see Algorithm 1). The second algorithm, the *pseudo-random generation algorithm* (*PRGA*), uses the register S to produce a pseudo-random keystream z of n-bit words during each iteration of the PRGA loop. The keystream generation process is witnessed in Algorithm 2.

Algorithm 1 Key Scheduling Algorithm (KSA) for RC4

```
Input: Shared Key $\mathcal{K}$
Output: State Register $S$
for $i = 0 \to 2^n - 1$ do
    $S_i \leftarrow i$
$j \leftarrow 0$
for $i = 0 \to 2^n - 1$ do
    $j \leftarrow j + S_i + \mathcal{K}_i \bmod 2^n$
    Swap($S_i, S_j$)
return($S$)
```

Encryption works by dividing the plaintext $\mathcal{P}$ into n-bit words and xor'ing them with the keystream z to produce the ciphertext $\mathcal{C}$. This can be expressed as:

$$\mathcal{E}(\mathcal{P}, z) = \mathcal{P} \oplus z = \mathcal{C}$$

Decryption is done by xor'ing each n-bit word of $\mathcal{C}$ with the keystream z producing $\mathcal{P}$. Decryption can be represented as:

$$\mathcal{D}(\mathcal{C}, z) = \mathcal{C} \oplus z = \mathcal{P}$$

Algorithm 2 Pseudo-random Generation Algorithm (PRGA) for RC4

```
Input: State Register $S$
Output: Keystream bytes $z$
$i \leftarrow 0$
$j \leftarrow 0$
while 1 do
    $i \leftarrow i + 1 \bmod 2^n$
    $j \leftarrow j + S_i \bmod 2^n$
    Swap($S_i, S_j$)
    $z \leftarrow S_{S_i + S_j \bmod 2^n}$
    output($z$)
```

1) RC4 State Register: The state register S is represented in the following way. The register as a whole is represented as an array S while each word in the array is represented as $S[i]$, where $i \in \{0, \dots, 2^n - 1\}$ making up a unique permutation. Within each permutation value of $S[i]$, there exists a binary value denoted as $b_{(i)}$ where $i \in \{0, \dots, n\}$. An illustration of this representation can be seen in Figure 1. The standard word size is $n = 8$ bits which implies that the total storage of a particular candidate solution in the algorithm is 2048 bits or 256 bytes; practical implementations of RC4 would require 2064 bits to account for the two index pointers (i and j).

B. Genetic Algorithms

Genetic Algorithms (GAs) were introduced as a viable optimization algorithm by John Holland in 1975 [9]. The algorithm utilizes methods found in biology to help evolve a set of candidate solutions (called *chromosomes*) to solve an optimization problem using a cost function over time ([10], [11], [12], [13]). GAs are members of the evolutionary family of algorithms due to their implementation of various evolutionary operators such as reproduction and mutation.

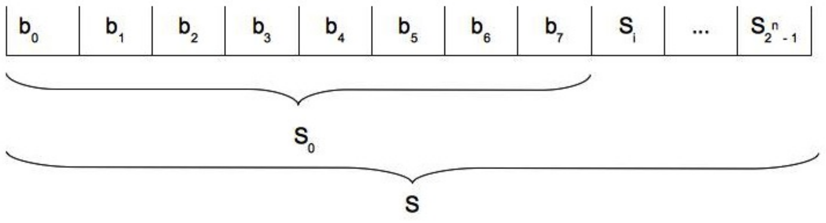


Fig. 1: State Register bit/byte Representation

Solution strength is measured by *fitness* and is determined by a *fitness function* $f(x)$ which quantifies the quality of the solution with a desired requirement [14]. A group of solutions or *population* are a small sample of all possible solutions, thus there is no guarantee that a best fit solution will be the global optimum for the problem. Reproduction or *crossover* is conducted each generation of the algorithm so the stronger candidate solutions have a better probability of entering the next generation of the algorithms life cycle. This is known as *selection*. To ensure that a population does not converge too quickly on a local optimal solution, a process called *mutation* is also introduced in the reproduction process. A basic algorithm of a GA can be seen in Algorithm 3.

Algorithm 3 General Genetic Algorithm

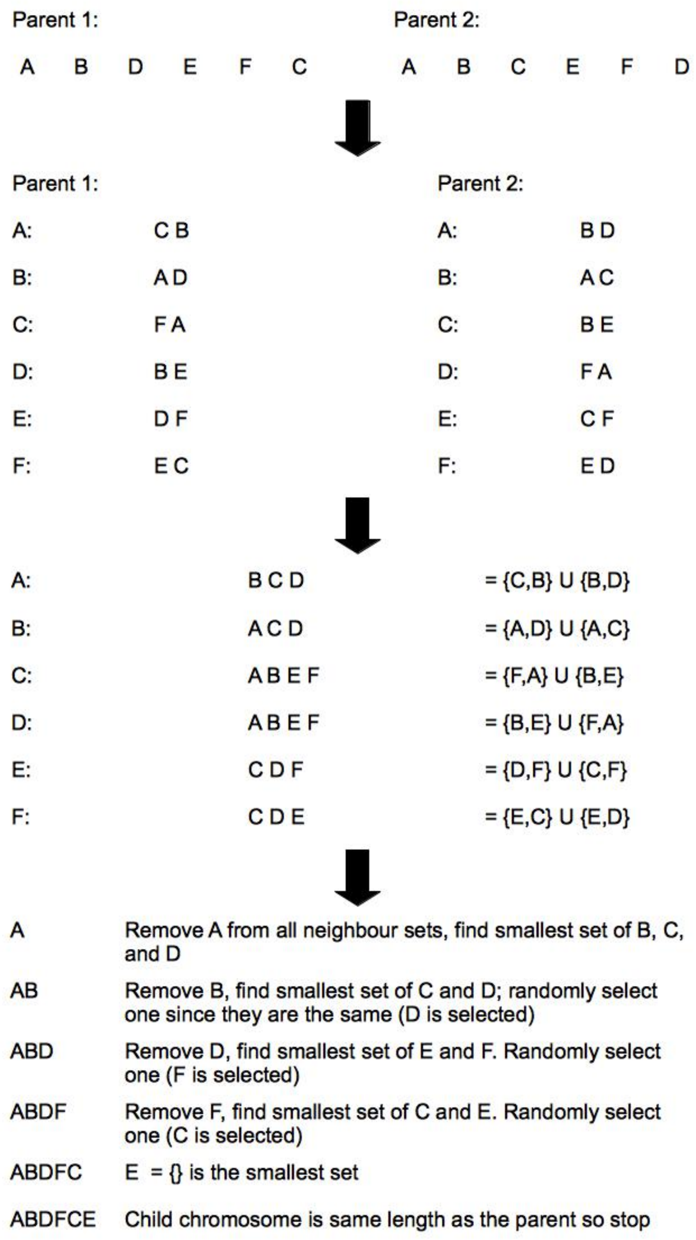
```
Input: Generate initial population $p$ of size $N$ randomly
while Max iteration not met or fitness not satisfactory do
  for $i = 1 \to N/2$ do
    Select $p_1$ and $p_2$ from $population$
    $(c_1, c_2) \leftarrow$ Crossover($p_1, p_2$)
    Mutate($c_1$)
    Mutate($c_2$)
    Insert $c_1$ and $c_2$ into $newpoulation$
  Replace $population$ with $newpopulation$
  for $i = 1 \to N$ do
    fitness($p_i$)
```

C. Related Work

One of the first uses of genetic algorithms to aid in the cryptanalysis of a cipher was in 1993 by Richard Spillman. He chose to attack the *Knapsack cipher* using a genetic algorithm [15]. A GA was utilized to decrypt cipher text letter by letter with an average decryption time of 84 seconds making the attack a practical use of a soft computing method. In [16], a genetic algorithm was used to attack the *Chor-Rivest Knapsack public key crypto system* with very promising results in all test cases. This also happened to be the first public attack on the crypto system. The algorithm was able to attack the cipher with a minimal search space as well as a very small number of generations. The encryption algorithm uses modular multiplication as well as logarithmic functions implying that a genetic algorithm should be able to tackle the far simpler arithmetic involved in the RC4 stream cipher. Brown et al. made use of a genetic algorithm to attack a substitution permutation network. While the study was intended to find weak keys, it was later noted that a GA would be a good tool to attack the scheme [17]. The research used SwM as a genetic operator to attack the network and it was also a method proposed to attacking other relevant crypto systems such as *AES* and *3DES*. Surprisingly, no work has been done on the cryptanalysis of RC4 using genetic algorithms.

In [18], it was found that the use of a genetic algorithm could be used to break a simplified implementation of *Data Encryption Standard* (DES) called *S-DES*. The research was successful in retrieving the key used for encryption as well as performing the task in less than 1/5 the time it would take to brute force the key (remember this is S-DES and is far easier to brute force than regular DES would be). The following paper will utilize several different methods than the research just mentioned. The first area is the method used as a crossover method; the previous paper used a ring crossover method that does not respect the properties of ordered chromosomes such as permutations. The second deviation is the use of letter frequency analysis as the fitness function which is unnecessary for our GA since we are solving for the keystream and not a decrypted plaintext.

Genetic Algorithms have had a recent introduction as a form of not only cryptanalysis, but steganography as well. In [19], researchers used a genetic algorithm to encode image information in a watermark that could then be hidden or kept in plain sight within the image. The method utilized by the researchers was also highly resilient to many common attacks of detection [19]. What makes this paper relevant is that the watermarking problem is encoded as a permutation problem. As a result, the authors examine several *ordered crossover* (OX) methods including *partially mapped crossover* (PMX) and *edge recombination crossover* (ER) as well as a method called *cycle crossover* (CX) that was not chosen for this task due to its larger computational needs over the other methods proposed. The authors also experimented with both the *swap mutation method* (SwM) and the *inversion mutation method* (InvM).



Parent 1:

A B D E F C

Parent 2:

A B C E F D

Parent 1:		Parent 2:	
A:	C B	A:	B D
B:	A D	B:	A C
C:	F A	C:	B E
D:	B E	D:	F A
E:	D F	E:	C F
F:	E C	F:	E D

A:	B C D	= {C,B} U {B,D}
B:	A C D	= {A,D} U {A,C}
C:	A B E F	= {F,A} U {B,E}
D:	A B E F	= {B,E} U {F,A}
E:	C D F	= {D,F} U {C,F}
F:	C D E	= {E,C} U {E,D}

A	Remove A from all neighbour sets, find smallest set of B, C, and D
AB	Remove B, find smallest set of C and D; randomly select one since they are the same (D is selected)
ABD	Remove D, find smallest set of E and F. Randomly select one (F is selected)
ABDF	Remove F, find smallest set of C and E. Randomly select one (C is selected)
ABDFC	E = {} is the smallest set
ABDFCE	Child chromosome is same length as the parent so stop

Fig. 2: Example of the steps involved in edge recombination crossover (ER)

II. Proposed Genetic Algorithm

The proposed GA will try to solve for the permutation that is represented as the state register S in RC4 using a variety of crossover and mutation methods. Due to the requirements of a permutation, special crossover and mutation algorithms must be implemented in order to not destroy the ordered property of a candidate solution. An attempt to *brute-force* the permutation would work out to $256! = 8.578\text{x}10^{506}$ possible permutations. Any improvement on this value can be seen as an improvement on such an approach.

The GA will evaluate the fitness of a candidate solution based on how well it can replicate the keystream that will be provided. Each keystream will be 256 bytes in length as suggested in [20] to find a unique solution. The fitness value will be an integer value from 0 to 256 representing how many bytes of the keystream were successfully replicated using a candidate state register.

Through various testing of different parameters for the GA, a population of size 5 has been selected to help evolve the best solution. For the proposed experiments, *tournament selection* has also been chosen as the selection method with a tournament size of 2. Finally, *elitism* will also be implemented which ensures that the best solution from a generation is carried over to the next one. This in turn does not allow the genetic operators to accidentally destroy the best solution found thus far by selection.

A. Proposed Crossover Methods

Traditional single-point and two-point crossover methods would destroy the permutation represented in the chromosome. Due to this constraint, ordered crossover methods are employed.

The first algorithm used for ordered crossover is *partially mapped crossover* (PMX). This method resembles two-point crossover in that two points are selected and the bits (bytes in our case) between them are exchanged to create two new children. It differs when the original parents are added to the child chromosomes, each value is added to the child in order of appearance until that value is already present in the child from the original crossover. At this point the duplicated value is replaced in the child chromosome by one of the values that was removed also due to the initial crossover.

The second method used is *edge recombination crossover* (ER). This method uses an adjacency matrix which is a list of each node and their respected neighbours. A master adjacency matrix is constructed by taking the union ($\cup$) of the two parent matrices. From the new matrix, a starting node is randomly selected and removed from all neighbouring sets and the node is appended to the empty child list. The next node appended is the smallest nodes set of the previous set. In the event which there are multiple sets that are the smallest, the set to use is randomly chosen. The process is repeated until the child list is the same length as the parent chromosome. A full example of ER is seen in Figure 2.

B. Proposed Mutation Methods

Similar to the crossover methods, the mutation methods employed must uphold the permutation property that exists for the candidate solution.

The first method for mutation that is called the *swap mutation method* (SwM). This form of mutation is very straight forward and is conducted by selecting two random indices in the state register and simply swapping their contents.

The second mutation method that is implemented is the *inversion mutation method* (InvM). This algorithm also preserves the permutation requirement of the chromosome. The process requires two indices in the chromosome to be selected with the sub-sequence between the two indices being simply reversed.

1) Adaptive Mutation Method: The GA will also use an adaptive mutation method. The default rate of mutation is set to 4% but there is a ceiling mutation rate also set at 15%. During the iterations, the best fitness is sampled at a predetermined rate proportional to the total amount of iterations. If the fitness appears to stagnate over these samples, the mutation rate is increased by 1% until it hits the ceiling rate in order to encourage more diversity in the population. Conversely if the fitness seems to improve over this period, the mutation rate will decrement by 1% until it is back to the original rate of mutation.

III. Experiments and Discussion

In the following section we will present the results of our experiments (see Section III-B) and evaluate the data collected (see Section III-C).

A. Equipment Used

All experiments were conducted on an Intel i7 dual-core CPU running at 2.8 GHz. The machine had 4 Gb of DDR3 RAM available to it. All three methods of exploration were programmed using C and were compiled with gcc version 4.2.1. Aside from certain programming optimizations including reducing any use of system functions that could be time intensive (i.e. *malloc()* and *free()*), optimizations at the compiler level were done using the *-O3* flag built into the gcc compiler.

B. Results

Several experiments were conducted on word sizes 6, 7, and 8. Of the results collected, PMX and SwM were shown to be the best operators for the state register problem. Further, adaptive mutation was shown to be a successful method of not allowing the candidate solutions to fall into a local optima too early. The results of utilizing both PMX and SwM with adaptive and non-adaptive mutation are exhibited in Table III.

	n					
	6		7		8	
Generations	non-adaptive	adaptive	non-adaptive	adaptive	non-adaptive	adaptive
10000	4.5	4.7	4.7	4.2	3	3.8
100000	8.4	10.6	11.6	11.3	10.5	10.5
1000000	11.8	11.8	17.4	16.9	18.1	20.7
10000000	12.8	16.2	21.4	23.9	29.5	26.2

TABLE III: Best average fitness for word size n using a genetic algorithm

The graph in Figure 4 shows that the fitness increases logarithmically and increases consistently for all sizes of the problem.

1) Comparison of Crossover Operators: Two crossover operators were examined. The operators were PMX and ER. While it was found that ER was more successful when solving the traveling salesman problem [21], when solving the RC4 permutation problem, PMX was found to be a far better technique. It is predicted that due to the requirement that the register produces the keystream consecutively, PMX disturbs the candidate solution far less than ER. Thus PMX maintains the integrity of higher fitness solutions in the crossover phase.

PMX was able to evolve a solution far better than ER by almost 10%. This is shown in Figure 5 where $n = 6$ and adaptive mutation is utilized. Both methods grow logarithmically, but the ER method just does not produce the results that the PMX achieves. Finally ER was far more time consuming when running the GA (especially for generations of 1 million and greater).

2) Comparison of Mutation Operators: The two ordered mutation operators chosen were SwM and InvM. Each of these approaches preserves the permutation property of the state register S. Experiments confirmed that SwM was a far better candidate for evolving fitness than InvM. The improvements that SwM improved the percentage fitness over InvM are seen in Figure 3. In fact, InvM seemed to reduce the best fitness found after about 1 million generations. This is believed to be the case because SwM make a number of small changes to the to the candidate solution while InvM makes a larger impact on the solution and can potentially mutate the entire solution if the indexes randomly selected were 0 and $2^n - 1$.

C. Discussion

From the results displayed in Figure 4, we were able to extrapolate an equation for the curve of $n = 8$. Using logarithmic regression we were able to derive Equation 1. This equation allowed us to determine on average how many generations it would take before 100% of the keystream could be recovered.

$$f(x) = 0.0131306222 \ln(x) - 0.1065234375 \qquad (1)$$

Using Equation 1, it can be predicted that on average the keystream could be completely replicated after approximately $4.0\text{x}10^{36}$ or approximately $2^{121.5}$ generations. This attack would be a great improvements over other theoretical attacks such as [22] which had a complexity of 2^{241}. While this attack has a large complexity in terms of computer power necessary to conduct it, it is still a great improvement over $8.578\text{x}10^{506}$, which is what is needed to brute-force the state register permutation. Finally if the GA was implemented as a hardware chip, it could pump out one generation each cycle.

IV. Conclusions

In closing, we presented a genetic algorithm capable of evolving a candidate solution to try and solve the permutation of the state register S. Further, we added the capability for the GA to utilize adaptive mutation in order for the population to converge at a far slower rate since the search space is so vast to begin with ($8.578\text{x}10^{506}$). We also did a comparison of two common ordered crossover operators and two common ordered mutation operators and found that PMX out-performed ER for this type of problem while SwM proved to be a much better mutation operator than InvM for the same problem.

It has been shown that for different sizes of the permutation problem that the best fitness improves logarithmically over generations. Finally, it was derived through extrapolation that on average 100% of the keystream could be derived in about $2^{121.5}$ generations. This attack is far better than previously outlined theoretical attacks.

V. Future Work

Other operators for ordered problems such as cycle crossover should be investigated to confirm whether the ones examined in this paper were the best candidates for this particular problem. The GA could be parallelized to further improve the running time of the algorithm and allow higher amounts of generations to be tested in reasonable time.

It would also be interesting to see how well the GA fairs with other encryption algorithms which would also make an interesting study of the versatility of the GA when used for cryptanalysis.

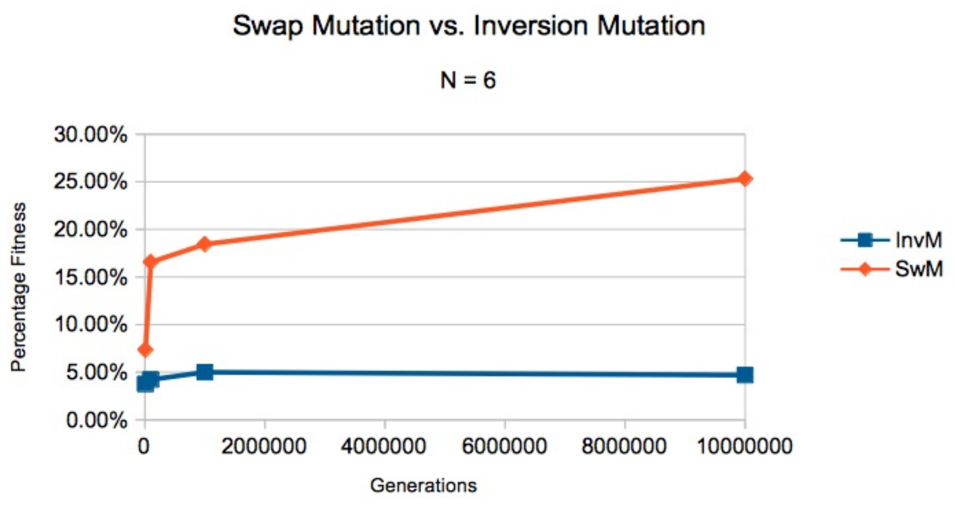


Fig. 3: Swap mutation vs. inversion mutation for $n = 6$ using adaptive mutation

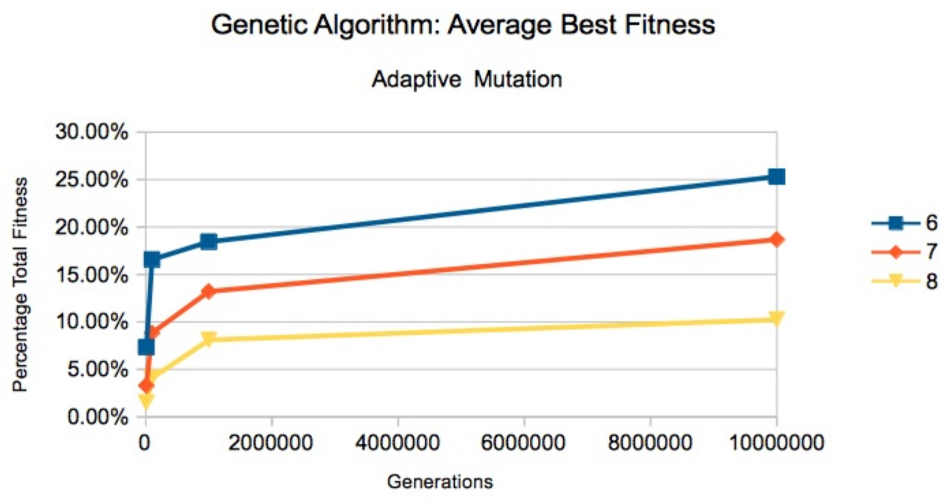


Fig. 4: Genetic algorithm best average fitness for various word sizes n

References

[1] P. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou, "Hardware implementation of the RC4 stream cipher," in *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, vol. 3. IEEE, 2003, pp. 1363–1366.

[2] F. A., K. P., and K. P., "RFC: 6101: The Secure Socket Layer (SSL) Protocol Version 3.0," *Internet RFC 6101*, 2011. [Online]. Available: http://tools.ietf.org/html/rfc6101.txt

[3] D. T. and A. C., "RFC: 2246: The TLS Protocol (v1)," *Internet RFC 2246*, 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[4] A. H. and J. S., "RFC: EAP Mechanism using TLS and SASL (version 1) draft," *Internet RFC*, 2001. [Online]. Available: http://tools.ietf.org/html/draft-andersson-eap-tls-sasl-00

[5] P. J. Hancock, "Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification," in *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*. IEEE, 1992, pp. 108–122.

[6] D. R. Jones and M. A. Beltramo, "Solving Partitioning Problems with Genetic Algorithms." in *ICGA*, 1991, pp. 442–449.

[7] H.-F. Wang and K.-Y. Wu, "Hybrid genetic algorithm for optimization problems with permutation property," *Computers & Operations Research*, vol. 31, no. 14, pp. 2453–2471, 2004.

[8] P. Prinetto, M. Rebaudengo, and M. S. Reorda, "Hybrid genetic algorithms for the traveling salesman problem," in *Artificial Neural Nets and Genetic Algorithms*. Springer, 1993, pp. 559–566.

[9] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Oxford, England: U Michigan Press, 1975.

[10] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-16, NO. 1, January 1986.

[11] Z. Michalewicz, "Genetic algorithms, numerical optimization, and constraints," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, vol. 195. Morgan Kaufmann, San Mateo, CA, 1995, pp. 151–158.

[12] C.-Y. Lin and P. Hajela, "Genetic algorithms in optimization problems

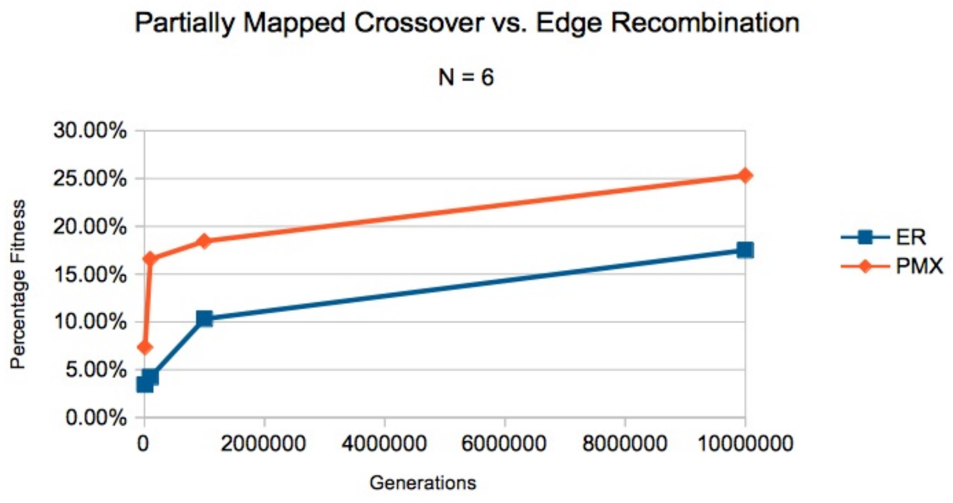


Fig. 5: Partially mapped crossover vs. edge recombination for $n = 6$ using adaptive mutation

with discrete and integer design variables," *Engineering Optimization*, vol. 19, no. 4, pp. 309–327, 1992.

[13] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA journal on computing*, vol. 6, no. 2, pp. 154–160, 1994.

[14] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," in *Proceedings of the first european conference on artificial life*. Cambridge: The MIT Press, 1992, pp. 245–254.

[15] R. Spillman, "Cryptanalysis of Knapsack Ciphers using Genetic Algorithms," *Cryptologia*, vol. 17:4, pp. 376–377, October 1993.

[16] I. Yaseen and H. V. Sahasrabuddhe, "A genetic algorithm for the cryptanalysis of chor-rivest knapsack public key cryptosystem (pkc)," in *Computational Intelligence and Multimedia Applications, 1999. ICCIMA '99. Proceedings. Third International Conference on*, 1999, pp. 81–85.

[17] J. Brown, S. Houghten, and B. Ombuki-Berman, "Genetic Algorithm Cryptanalysis of a Substitution Permutation Network," *IEEE Symposium on Computational Intelligence in Cyber Security*, pp. 115–121, March 2009.

[18] L. Sharma, R. Sharma, and B. K. Pathak, "Breaking Simplified Data Encryption Standard Using Genetic Algorithm," *Journal of Computer Science and technology*, vol. 12:5, pp. 55–59, 2012.

[19] V. Alvarez, J. A. Armario, M. D. Frau, F. Gudiel, M. B. Guemes, E. Martin, and A. Osuna, "GA based robust blind digital watermarking," *Dept. Matematica aplicada, Dept. Algebra*, pp. 376–377, 2012.

[20] G. Carter, E. Dawson, and K. Wong, "An Analysis of the RC4 Family of Stream Ciphers against Algebraic Attacks," *Proc. 8th Australian Information Security Conference (AISC 2010)*, 2010.

[21] P. Larranaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129 – 170, 1999.

[22] A. Maximov and D. Khovratovich, "New state recovery attack on RC4," in *Advances in Cryptology–CRYPTO 2008*. Springer, 2008, pp. 297–316.