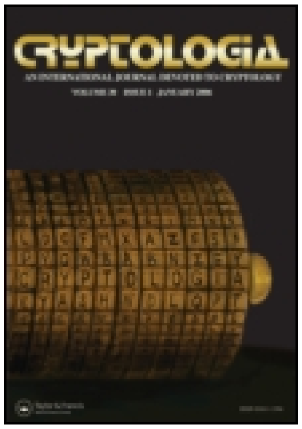

Cryptologia
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ucry20

USE OF A GENETIC ALGORITHM IN THE CRYPTANALYSIS OF SIMPLE SUBSTITUTION CIPHERS

Richard Spillman [a] , Mark Janssen [a] , Bob Nelson [a] & Martin Kepner [a]
[a] Department of Computer Science Pacific Lutheran University Tacoma WA 98447 USA. Email: SPILLMAN_R@PLU or JANSSEN@PLU.

Published online: 04 Jun 2010.

To cite this article: Richard Spillman , Mark Janssen , Bob Nelson & Martin Kepner (1993) USE OF A GENETIC ALGORITHM IN THE CRYPTANALYSIS OF SIMPLE SUBSTITUTION CIPHERS, Cryptologia, 17:1, 31-44, DOI: 10.1080/0161-119391867746

To link to this article: http://dx.doi.org/10.1080/0161-119391867746

USE OF A GENETIC ALGORITHM IN THE CRYPTANALYSIS OF SIMPLE SUBSTITUTION CIPHERS

Richard Spillman, Mark Janssen, Bob Nelson, and Martin Kepner

ADDRESS: Department of Computer Science Pacific Lutheran University Tacoma WA 98447 USA. Email: SPILLMAN_R@PLU or JANSSEN@PLU.

ABSTRACT: This paper considers a new approach to cryptanalysis based on the application of a directed random search algorithm called a genetic algorithm. It is shown that such a algorithm could be used to discover the key for a simple substitution cipher.

KEYWORDS: Simple substitution cipher, genetic algorithm, key search.

1.0 INTRODUCTION

Cryptanalysis is the process of recovering the plaintext and/or key from a cipher. It may involve the use of just the ciphertext, portions of several ciphertext items, some plaintext, complicated mathematical procedures, computer algorithms, and usually some luck. Many cryptographic systems have a finite key space and, hence, are vulnerable to an exhaustive key search attack. Yet, these systems remain secure from such an attack because the size of the key space is such that the time and resources for a search are not available. In these cases, the cryptanalyst looks for trapdoors; exploitable regularities in either the cipher system or the language or both; combinations of plaintext and ciphertext; or anything else which may prove helpful in breaking the cipher. To a cryptanalyst, the concept of a random search of the key space would be considered the last refuge of the hopelessly naive. A random search through a finite but large key space is not usually an acceptable cryptanalysis tool.

In this paper, however, we explore the possibility of using a random type search to break a cipher. The focus of this work is on the use of a genetic algorithm to conduct a *directed* random search of a key space. The ability to add *direction* to what seems to be a random search is a feature of genetic algorithms which suggests that they may be able to conduct an efficient search of a large key space. Goldberg emphasizes the fact that while a genetic algorithm may appear to be random it is in fact a powerful search technique.

> The genetic algorithm is an example of a search procedure that uses random choice as a tool to guide a highly exploitative search through a coding of a parameter space. ...The important thing to recognize at this juncture is that randomized search does not necessarily imply directionless search [2, p. 12].

The goal of this initial work is to illustrate the feasibility of using genetic algorithms as a cryptanalysis tool. Hence, our algorithms are only applied to simple substitution ciphers in this first report in order to clearly present the fundamentals of the tool without becoming trapped in the details of a more complicated cipher.

2.0 REVIEW OF GENETIC ALGORITHMS

Genetic algorithms (GAs) where first suggested by John Holland in the early seventies [3]. Over the last 20 years they have been used to solve a wide range of search, optimization, and machine learning problems. As the name indicates, these algorithms attempt to solve problems by modeling the way in which human genetic processes seem to operate (at least at a simple level). A good survey of the nature and use of genetic algorithms can be found in a paper by Liepins and Hilliard [4].

The easiest way to consider a GA is in the context of a function optimization problem. The goal is to search through a space of function inputs (each represented by some binary string) to find the input which maximizes a given target function. A genetic algorithm begins with a randomly selected population of function inputs represented by strings. The GA uses the current population of strings to create a new population such that the strings in the new population are on average "better" than those in the current population. For example, a population of strings may be generated to search for an optimal value for the function x^2-x+1 as shown Figure 1. The third element in this sample population represents $x = 10$. It produces the best function output.

Population	Function Value		New Production
0001	1		high function
0110	31	$\Longrightarrow$	
1010	91		value terms

Figure 1. String population.

Three process which have a parallel in human genetics are used to make the transition from one population generation to the next. They are selection, mating, and mutation. The basic GA cycle based on these three processes is shown in Figure 2.

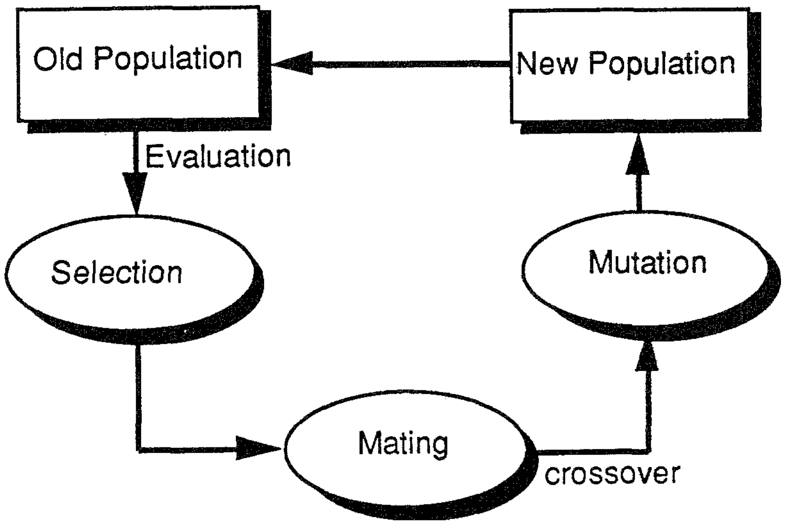

Figure 2. The basic genetic algorithm cycle.

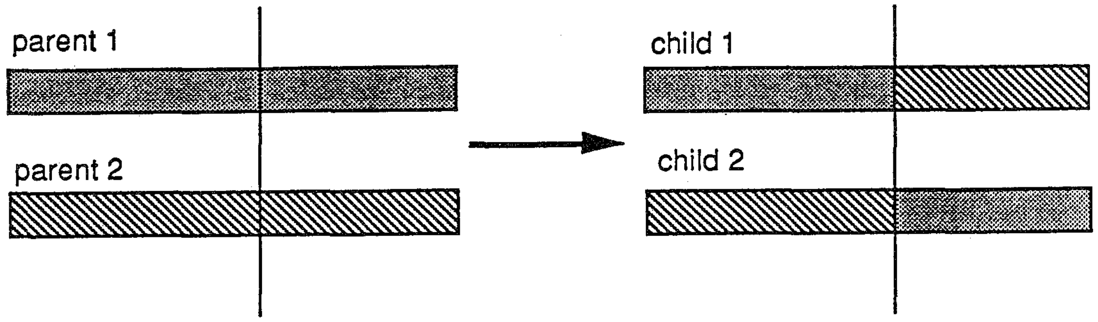

Figure 3. The crossover operation.

The first step is the selection process. The selection process determines which strings in the current generation will be used to create the next generation. This is done by using a biased random selection methodology. That is, parents are randomly selected from the current population in such a way that the "best" strings in the population have the greatest chance of being selected. In Figure 1 the string 1010 has the greatest chance of being a parent while 0001 has the least chance. By using the "best" points to determine the next population, the algorithm seems to move in the most promising direction in its overall search. The second step is the mating process. The mating process determines the actual

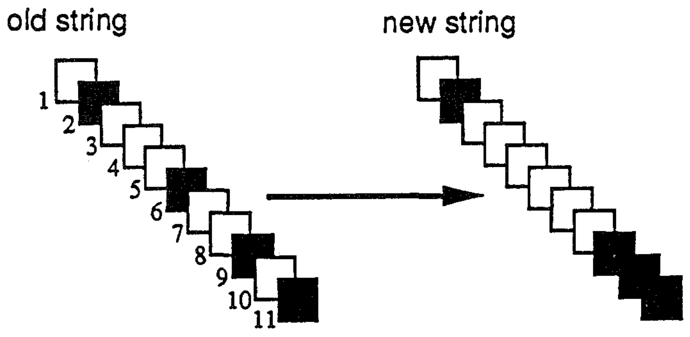

Figure 4. Mutation process.

form of the strings in the next generation. At this point, two of the selected parents are paired. If the length of the each string is r, then a random number between 1 and r is selected, say s. The mating process is one of swapping bits $s+1$ to r of the first parent with bits $s+1$ to r of the second parent. In this way, two new strings are created as shown in Figure 3.

The final step is one of mutation. A fixed, small mutation probability is set at the start of the algorithm. Bits in all the new strings are then subject to change based on this mutation probability. In Figure 4, bits 6 and 10 are mutated. The result is a new generation of strings. These three steps are repeated to create each new generation. It continues in this fashion until some stopping condition is reached (such as a maximum number of generations).

This algorithm has proven to be very effective in some search problems [5]. While it may seem to be a random search, in fact, the improvement in each generation indicates that the algorithm provides an effective directed search technique. It is this search capability which is applied to the problem of cryptanalysis of simple substitution ciphers in this paper.

3.0 APPLICATION OF GAs TO KEY SEARCH

The focus of this paper is to apply a genetic algorithm to the problem of searching through the key space of a simple substitution cipher. There are three systems which need to be defined in order to set up such an approach. The first is the representation scheme. The problem is to set up a string representation of possible keys in a way that allows the mating and mutation processes to operate. The second is to determine a mating process consistent with the key representation. The third is to select a mutation process. Possible methods for all three are suggested in this section.

3.1 KEY REPRESENTATION

There are several ways in which a simple substitution key could be represented. For the purpose of this study, a key for decoding a cipher is given by an ordered list of the 26 letters of the alphabet. The order expresses the underlying substitution. That is, the key CDEFGHIJKLMNOPQRSTUVWXYZAB indicates that the most frequent character in the cipher represents a plaintext C, the second most frequent character in the cipher represents a plaintext D, the third most frequent character represents a plaintext E, etc. As this representation indicates, the size of the key space is 26! or greater than 4×10^{26} which suggests that a purely random search is not acceptable.

Once a representation scheme is available for a genetic algorithm, it is also necessary to supply an evaluation function. This function is used to determine the "best" representations. The evaluation function selected for this study is based on single character and digram frequency analysis. The process is as follows:

1. A given key is used to decipher the ciphertext
2. The resulting text is subjected to both single character and digram frequency analysis
3. The results of this frequency analysis are compared to standard English frequencies to determine a error value
4. The error value is normalized and subtracted from 1 so that larger fitness values represent smaller errors
5. The fitness value is raised to the 8th power to amplify small differences.
6. The summation terms are divided by 4 to reduce sensitivity to large differences. Overall the fitness of a key is given by:

$$\text{Fitness} = \left(1 - \sum_{i=1}^{26} \left\{ |SF[i] - DF[i]| + \sum_{j=1}^{26} |SDF[i,j] - DDF[i,j]| \right\} /4 \right)^8$$

$SF[i]$ is the standard frequency of character i in English plaintext. $DF[i]$ is the measured frequency of the decoded character i in the ciphertext. So, the first summation term is the sum of the differences between the standard frequency of the each character and the frequency of the ciphertext character which decodes to that character. SDF is the standard frequency of digrams and DDF is the measured frequency of digrams. When measured frequencies match the standard frequencies, the summation terms evaluate to 0 so the fitness value is 1. The fitness equation is bounded below by 0 though it does not actually evaluate to 0. The fact that a fitness value of 0 is never achieved does not affect the algorithm since high fitness values are more important than low fitness values. As a result, the search process is always moving towards fitness values closer to 1. The overall guiding concept is that keys which produce plaintext with single character and diagram frequencies close to that of standard English are "better" than keys which produce plaintext with poor frequency distributions.

3.2 THE MATING PROCESS

Given a population of keys, each one with a fitness value, the algorithm progresses by randomly selecting two keys for mating. The selection is weighted in favor of keys with a high fitness value. That is, keys with a high fitness value have a greater chance of being selected to generate children for the next generation. The two parents generate two children using a crossover operation. For this study, the crossover operation involves scanning the two key strings from both ends selecting the most frequent character to pass on to the child. For example, given two parents:

```
CDEFGHIJKLMNOPQRSTUVWXYZAB

FGHIJKLMNOPQRSTUVWXYZABCDE
```

the first character of the first child is determined by selecting the most frequent (in the ciphertext) of the two characters C and F (say it is F) as in Figure 5.

Child 1 FDEIJHHLMKOPQRSTUVWXYXABCDE

Child 2 CDVFDFHIJKLMNOPQRSTUVWXBZAE

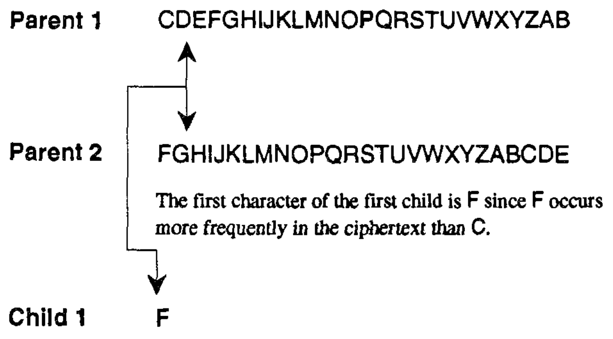
Parent 1
CDEFGHIJKLMNOPQRSTUVWXYZAB
Parent 2
FGHIJKLMNOPQRSTUVWXYZABCDE
The first character of the first child is F since F occurs more frequently in the ciphertext than C.
Child 1
F

Figure 5. The first step of the mating process.

The second character of the first child is determined by selecting the most frequent of the two characters D and G (say it is D). This process continues until it must select the most frequent of the last two characters in this case of B and E (say it is E). If the most frequent character already appears in the child, then the other character in the pair is automatically selected. If both characters have already appeared in the child, then a missing character is randomly selected. This process is repeated to create the second child by scanning in reverse (right to left) order. In this case the scan would be from the pair BE to the pair CF. The result might be:

```
Child 1:  FDEIJHLMKOPQRSTUVWXYXABCDE

Child 2:  CDVFDHIJKLMNOPQRSTUYWXBZAE
```

3.3 THE MUTATION PROCESS

After the new generation has been determined, the keys are subjected to a low rate mutation process. For this study when a character in the key string is selected for mutation, it is swapped with a randomly selected character in the key string if the fitness of the key string is in the lower half of the the gene pool. If the fitness of the string is in the top half of the population, the character is swapped with the character to its right. In Figure 6, the character D in the top string is mutated by swapping it with the character to its right ("E") to produce the bottom string.

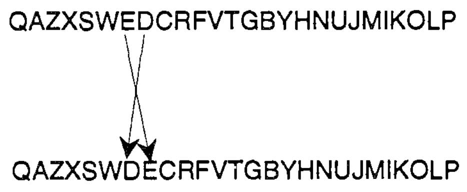

Figure 6. Mutate the character D.

3.4 THE COMPLETE ALGORITHM

These processes are combined to create the complete genetic algorithm. The steps of the algorithm are:

1. A random population of key strings is generated.

2. A fitness value for each key in the population is determined.

3. A biased (based on fitness) random selection of parents is conducted.
4. The crossover operation is applied to the selected parents.
5. The mutation process is applied to the children.
6. The new population of key strings is scanned and used to update a list of the "best" 10 keys across the generations.

This process will stop after a fixed number of generations and the best keys will be used to decipher the ciphertext.

4.0 RESULTS

This algorithm was programmed in Turbo Pascal and run on 386DX (25Mhz, no FPU). It was applied to ciphertext created using a simple substitution cipher. Both the population size and the mutation rate were varied to explore the operation of the algorithm. In general it was found that a fitness value of about .9 was sufficient to expose the vowel substitution as well as enough of the consonant substitutions to allow for the entire key to be determine by visual examination of the deciphered text. One hundred generations were usually enough to reach this level of fitness and the algorithm was fast enough that this took less than a minute on an 386 PC. The best way to illustrate the genetic algorithm in operation is to look at the development of the gene pool over time. For example, we ran one sample in which the ciphertext had been encoded using the key:

```
ETAOSINRHLDUCMWGPFYBVKXQZJ
```

That is, the most frequent character in the ciphertext is mapped to a plaintext E, the next most frequent ciphertext character represents a plaintext T, etc. For this example, only 10 keys were in the gene pool. The system began by generating 10 random keys as shown:

```
 1.  MBZXDEVQJHNASIWKYOPGUCTLRF
 2.  EGRCPMFITZYSOJQHNXKWUVADLB
 3.  CVFSWYAIUZPJXQETKRLOHDNGBM
 4.  SATLFHUBOZCEJVGRMWKDPYXIQN
 5.  EINQVSAUKXWTBGZROFHMLDPCYJ
 6.  QFMBDZJLPGOKCAHEYRTWUNISVX
 7.  BMORYDZLEPAFTNJCQIHVWGKSXU
 8.  AQEMFNUGBPZDVIXKLOCTRYWSJH
 9.  OKCFHEXLQMPJASVGDYWRZNBTIU
10.  ZPAGIKHFMTSUXEQJCLODVNBRYW
```

As would be expected none of the random keys are very close to the actual key which is reflected in the fact that the average fitness for these keys is 0.3965. The best of these 10 random keys (key 5) has a fitness value of only .4818.

After only 20 generations, this pool of 10 keys begins to take on a set of common features that approach the exact key:

```
 1.  ETAOISNRHDLUCYMVPBGFQKXWJZ
 2.  VTAOSINRHELWZCMYDBFGPKUXQJ
 3.  ETAOSINRHWLUCMZYDBGFPKVXJQ
 4.  ETAOSINRHWLUCZMYDBFGPKVQXJ
 5.  ETASIONRWHLCUZMYDBFGVKPXQJ
 6.  ETAOISNRWHLUCZMKYBFGPDVXQJ
 7.  TEAOSINRHWLUCZMYBFDGPKVXQJ
 8.  TEAOSINRWHQUZCMYDBFGPKVLXJ
 9.  ETAOISNRWLDCUBMYQZPGHKXVFJ
10.  ETOAISNHRLDWCZMYUBFGPKVXQJ
```

Now the average fitness is 0.75. The best key (key 3) has a fitness value of .8276. Notice that ETAO has moved to the front of almost every key in the population.

After 40 generations, the pool begins to converge at a high rate of speed:

```
 1.  ETAOISNRHLDUMCYVWGFBKPJQZX
 2.  ETAOSINHRLCDUMYVWFGPBKZJQX
 3.  ETAOSINRHLDUMCYWFGVBKPJQZX
 4.  ETADOINRHVLUMCYSFWGBPKZJQX
 5.  ETAOISNRHLUDCMYVWGFPBKZQJX
 6.  ETAOSINRHLUDCMVYWFGPBKZQJX
 7.  ETAOIMNRHLDCGSYWFPXBUKQJZV
 8.  ETAOSINRHLDUCMYVWFGPBKZQJX
 9.  ETASOIRNHDLUCMYWVGFPBKJQZX
10.  ETANOISRHDLUCMYWVFGPBKZQJX
```

The average fitness of the gene pool has risen to 0.808 with the best key (key 8) coming in at 0.8758. An examination of key 8 indicates that 18 of the substitutions are correctly identified by that key.

By generation 100 (and a total run time of only seconds), the gene pool looks like:

```
 1.  ETAOSINRHLDUCMWGPYFBVKXQZJ
 2.  ETAOSINRHLDUCMGWPYFBVQKXZJ
 3.  ETAOSINRLDYCUMWGPFXBVKHQZJ
```

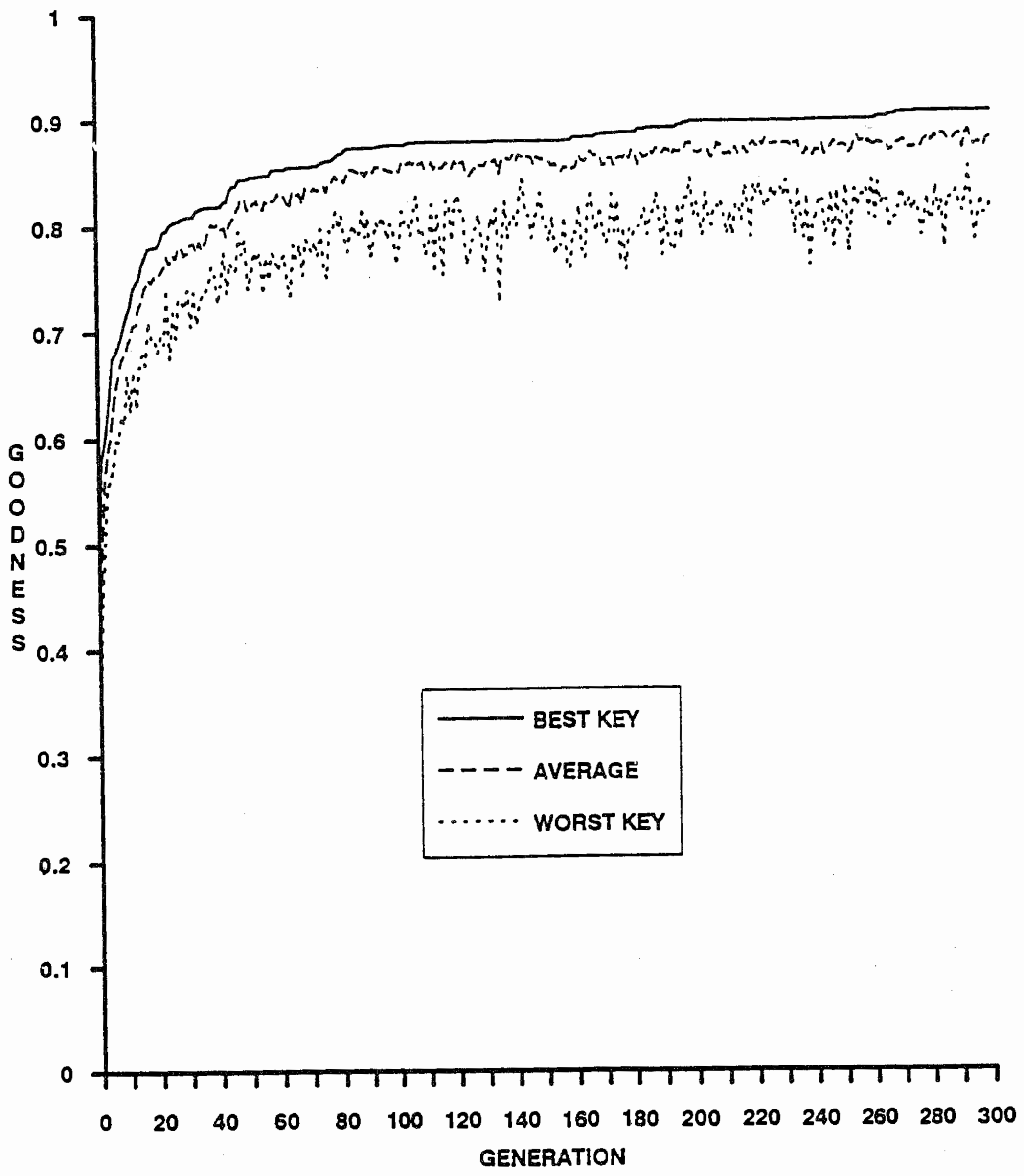

Figure 7. Ten(10) key gene pool with a small mutation rate (0.05).

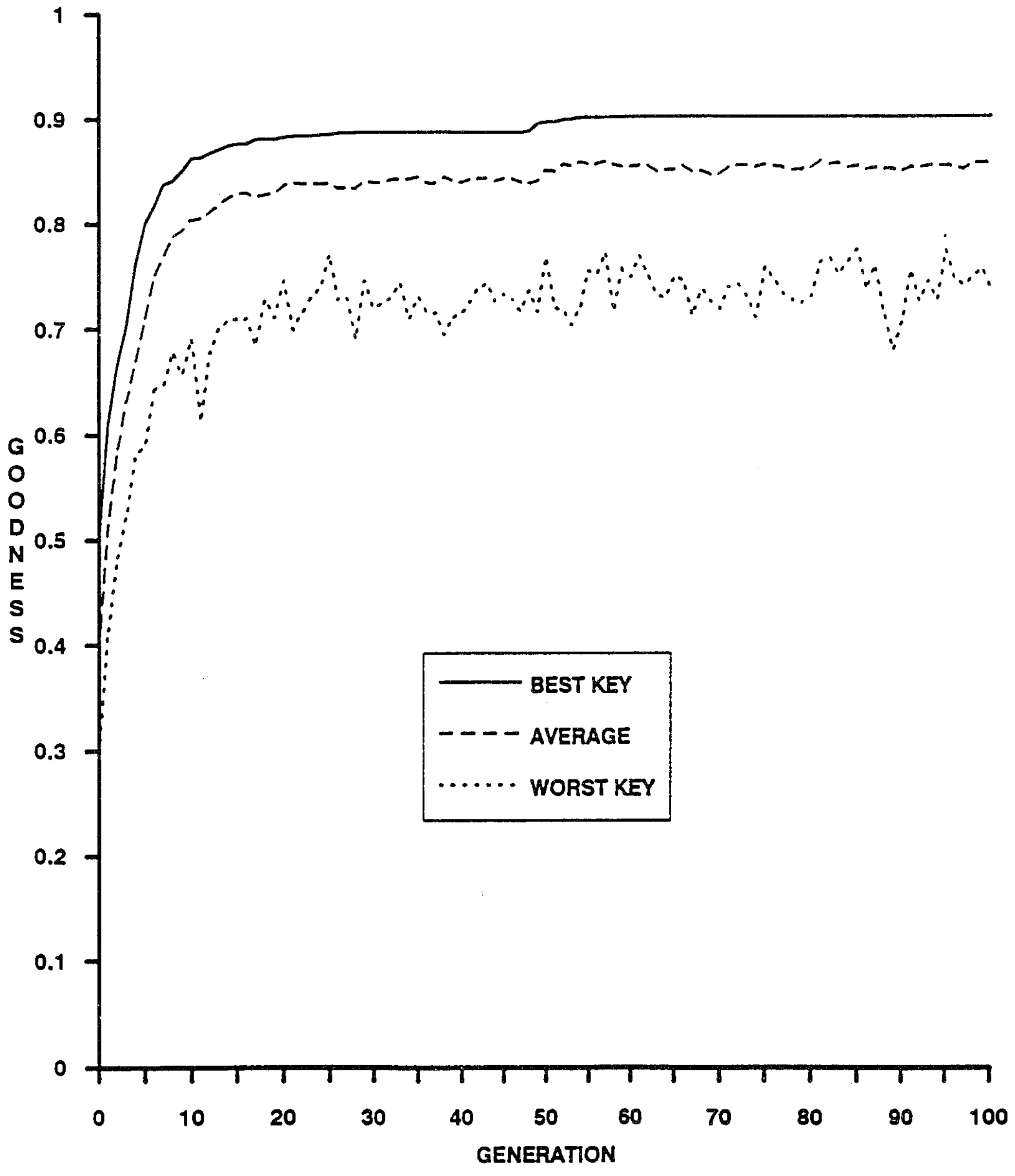

Figure 8. Fifty (50) key gene pool with a small mutation rate (0.05).

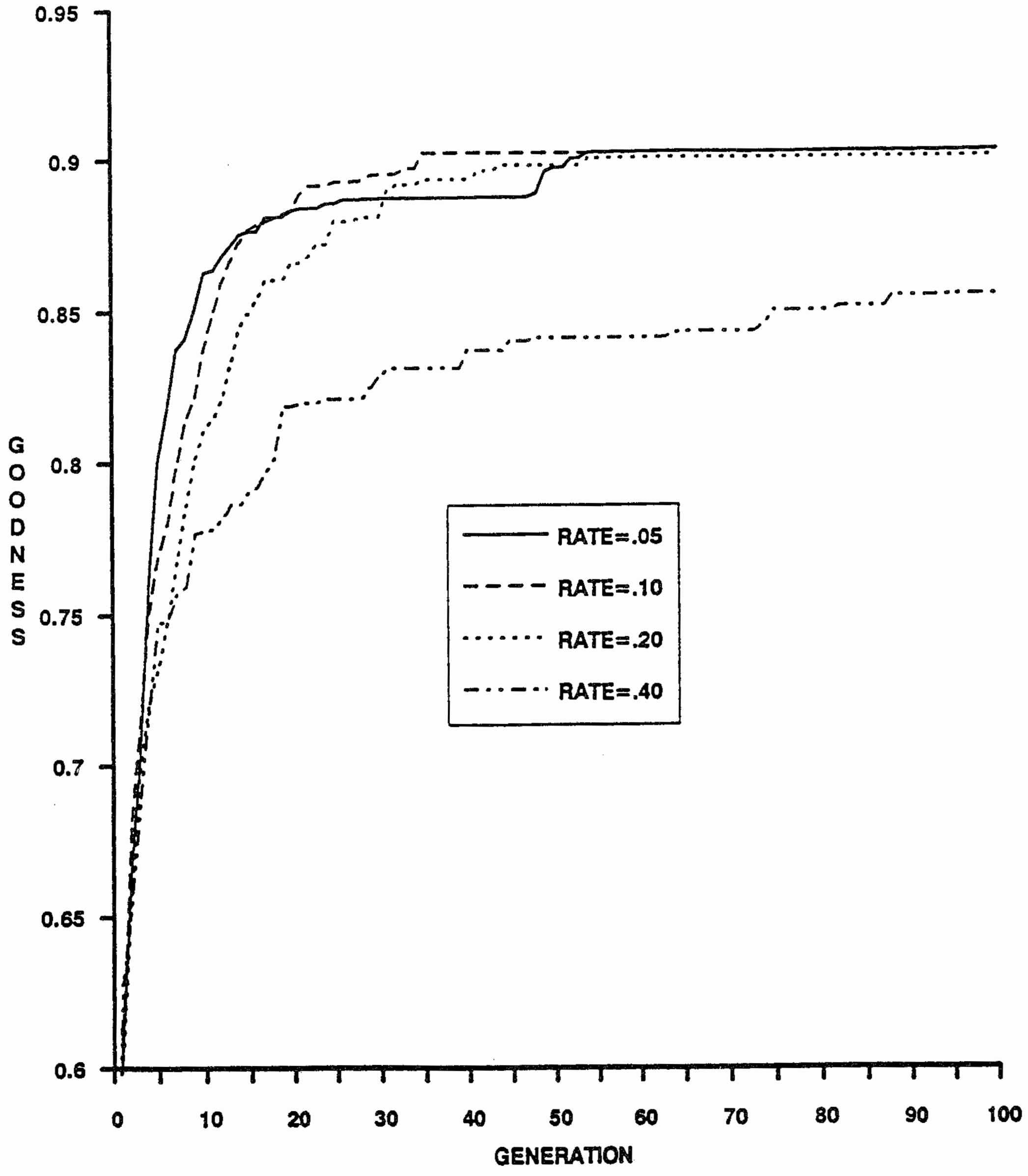

Figure 9. Fifty (50) key gene pool with a four mutation rates.

```
 4. ETAOSIFRWDLUCHMNGPYBVKXQZJ
 5. ETAOSINRHLUDCMWGFPYBVKXJQZ
 6. ETADSINJHLOUCMWGFPYBRKVXQZ
 7. ETAOSINRHLDUCMWGPFYBVKQXZJ
 8. ETAOSINRHLDUCMWGPFYBVKQXZJ
 9. ETAOSINRHLDUCMWGPFYBVKXQZJ
10. ETADSINROHLUCMWGFPYBVKXQJZ
```

The average fitness is now at 0.8486. Key 9 turns out to have the highest fitness and on examination is the exact key. The result is that the exact key was found after examining only 1000 keys in the key space. This is less than 4×10^{-23} of the entire space. This effect was found in other runs as well. Figure 7 shows a graph of the best, average, and worst key fitness across generations in a problem with a 10 key gene pool and a small mutation rate (probability of a mutation is 0.05). Notice that in the first 80 generations the best key improves quickly. There is also a steady improvement in the average fitness of all 10 keys in the gene pool. There is even a trend for the worst key in the gene pool to improve across generations.

Increasing the size of the gene pool can reduce the number of generations required to find an acceptable key. Figure 8 is a graph of the fitness across generations for a problem with 50 keys in each gene pool and a 0.05 mutation rate. Notice that the major improvement occurs in the first 20 generations for the best, average, and worst keys. It is interesting to note that the overall search effect is the same in both cases. Whether you search with 10 keys over a 100 generations or 50 keys over 20 generations, they both involve looking at about 1000 keys.

The effect of the mutation rate was also explored. Figure 9 shows the result of four experiments with the same problem. The gene pool consisted of 50 keys. Four different mutation rates were selected - 0.05, 0.10, 0.20, and 0.40. The highest mutation rate clearly hampered the search effort. In this case, too many keys were affected by the high mutation rate which clouded the search direction. On the other hand, low mutation rates (rates below 0.20) seemed to give better results. They provided enough variability to prevent the system from becoming stuck in a local minimum but not so much that they hampered the overall search effort.

5.0 CONCLUSIONS

Our original goal was clearly met. The GA proved highly successful in finding the key for a simple substitution cipher. It is not believed, however, that this approach will always find the exact key. It is evident that in a short run, it will come close enough to the exact key that a visual inspection of the resulting plaintext could be used to determine any misplaced letters. Given that this is a promising algorithm, there are several open avenues for further research. For example, it may be possible to improve the evaluation function using a form of interval analysis such as that suggested by Anderson [1]. It may also be possible to include a trigram analysis in the evaluation function. Variations on the crossover and mutation procedures may significantly affect the behavior of the algorithm. In addition, other more complicated ciphers may be analyzed. These concepts are currently under evaluation.

REFERENCES

1. Anderson, Roland. 1986. Improving the machine recognition of vowels in simple substitution ciphers. *Cryptologia*. 10: 10-22.

2. Goldberg, David. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading MA: Addison-Wesley.

3. Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor MI: University of Michigan Press.

4. Liepins, G. E. and M. R. Hilliard. 1989. Genetic Algorithms: Foundations and Applications. *Annals of Operations Research*. 21: 31-58.

5. Rawlins, G.. 1991. *Foundations of Genetic Algorithms*. San Mateo CA: Morgan Kaufmann Publishers.

BIOGRAPHICAL SKETCH

Dr. Richard Spillman is a professor of computer science and engineering at Pacific Lutheran University (PLU). His research interests are in cryptography and artificial intelligence.

Mark Janssen and Bob Nelson are undergraduate students in the computer science department at PLU.

Martin Kepner is a graduate students in the computer science department.