# Prime Number Generation Using Genetic Algorithm

Arpit Goel[1], Anuradha Brijwal[2], Sakshi Gautam[3]
[1] Dept. Of Computer Science & Engineering, Himalayan School of Engineering & Technology, Swami Rama Himalayan University, Dehradun Uttarakhand (India)
[2] Dept. Of Computer Science & Engineering, Himalayan Institute of Technology, Dehradun Uttarakhand (India)
[3]Dept. of computer science ,JB Institute of Technology Dehradun Uttrakhand.
[1]arpitgoel29@gmail.com, [2]meetanubrijwal01@gmail.com , [3]1sakshigautam@gmail.com

**ABSTRACT-** In the modern cryptography, the problem of the generation of large primes is considered as an important issue to implement public key crypto-scheme such as RSA and so on. Since a great amount of computational resources is generally required to generate a large prime number, it takes considerable amount of resource especially in embedded systems. In general, the prime number generation is started from the random number generation. If the generated random number is passed a specified probabilistic primality test, the random number is tentatively considered as a prime number and applied to a public key crypto-system. In many primality tests, a number of modular exponentiations are usually required. Hence these algorithms are considerably slow.
Here we are introducing a new prime generation technique using GA which can be used to generate large primes in more efficient way.

**Keywords:** Prime Number, Genetic Algorithm, Primality Test.

## 1 INTRODUCTION

The subject of prime numbers has fascinated mathematicians for centuries. Some of the methods for finding prime numbers date to antiquity [1]. The properties of primes have been investigated for thousands of years [1]. The advent of digital computers and public-key cryptography has brought the subject of prime numbers into the mainstream and focused renewed attention on it[1].

The use of public-key cryptography is pervasive in the information protection and privacy arenas. Public key crypto algorithms utilize prime numbers extensively; indeed, prime numbers are an essential part of the major public key systems[1].

For centuries, the problem of validating prime numbers, the primality test, has posed a great challenge to both computer scientists and mathematicians[2]. The problem of identifying Prime and Composite numbers is known as one of the most important problems in arithmetic. Many security applications also involve large numbers: while it is easy to multiply prime numbers to get a product, the reverse process of recovering the primes is much more difficult[2].

## 2 BACKGROUND DETAILS

### 2.1 Prime Number

Any natural number n greater than 1 will be prime if it has no divisor other then 1 and itself. The number 2 is a prime, there being no candidate divisors between 1 and itself similarly the numbers 3, 5,7,11,13 and 17 are all prime.

One more definition of prime number can be given in terms of relative prime. A prime number M will be prime if it is relative prime to all other numbers (M-1) relative primes.

### 2.2 Prime Generation Problem

Today the big problem is big prime numbers is generation.

#### 2.2.1 Difficulty in Finding New Prime

The difficulty of generating new primes can be explained by prime density function. The prime density function shows that how many prime number will exist below number n. Prime density function (n) can be given by following function n=log(n). From this function, it can be seen that a very less comparisons will be required if we want to find all prime numbers less than a short number such that 500. This happens because short prime numbers have small gap in between them but as we approach for a longer number, problem becomes very difficult. This is due to larger gap for longer prime numbers. So in order to find a prime we just pick an odd number and apply a prime testing algorithm on it. This is the only way of generating new primes. There are various primality tests which are available to us for checking the primality of a number. Let us discuss some important tests.

#### 2.2.2 Prime Testing Algorithms

A primality test is an algorithm that determines whether an input number is prime or not. The primality tests differ from regular process of integer factorization. The primality test doesn’t necessarily give prime factors, while integer factorization does. As of 2009, factorization is a computationally difficult problem, whereas primality test is compare to factorization is easy. There are various type of tests that are available to check the primality of number, most of them are deterministic.

## 3 BASICS OF GENETIC ALGORITHM

Traditional random approach for searching optimal solution in a given solution space was very time consuming and not suitable for large application. In 1975 Holland proposed a better solution than traditional random search approach known as genetic algorithm[3].

A genetic algorithm (GA) is a search heuristic that mimic the process of natural evolution. This heuristic is mostly used to generate useful solution to optimization and search problems. GA is a part of evolutionary algorithm, which is used for generating solution for optimization problems using techniques inspired by natural evolution and have properties like inheritance, mutation, crossover and selection.
GA is based on the survival of the fittest i.e. it selects only best solutions from among all available solutions.

### 3.1 Basic Operations of Genetic Algorithm
GA basically consists of four operations:-

#### 3.1.1 Encoding-
Basic of GA structure is the use of encoding for representation the optimization problem's variables to form chromosome which represent a candidate solution for problem. The encoding mechanism mainly depends on the type of problem variables. For example if the problem is to find the optimal flow in transportation problem than the variables will be of real type and when the problem is of travelling salesman than the variables will be of binary type. So for both problems we have to use different-different encoding mechanism that will encode the variable into unique string.

Traditionally we use the binary bit representation for encoding the variables. And collection of these binary bits will become string of bit and represent the solution. But it has some drawback and not suitable for some particular application, so some other mechanisms have also been proposed such as Gray code representation. So we have number of options for choosing best encoding mechanism that will be best suited our problem.

#### 3.1.2 Fitness Function-

We have some objective function depend on the problem which we need to optimize for particular string representing the solution. Each objective function has different values. So to maintain uniformity we use fitness function to normalize the objective function so it will give the value in the range of 0 and 1[4]. The normalize value of objective function represents the fitness of string that will use by selection operator for evaluating the string of solution.

#### 3.1.3 Selection-
Selection is one of the most important operator or step of GA. It models the survival of fittest process of nature. Fittest will survive for next generation while weakest will be lost. There are different selection procedures or methods available. Each has its pros and cons. One of the basic selection procedure is proportionate selection. In the proportionate selection procedure a string with fitness value $f_i$ will be allocated $f_i/f_a$ offspring, where $f_a$ is the average fitness of population. String with fitness value greater than average fitness of population will be allocated more than one offspring, While string with fitness value less than average fitness of population will be allocated less than one offspring. This procedure will results in fractional allocation number value. And finally we will require an integer value, so we have some method to convert these fractional values into integer value. One of the methods is Roulette Wheel Selection Scheme.

#### 3.1.4 Crossover-
This is also a crucial operation of GA. Pairs are picked randomly from population to be subjected for crossover and then crossover will be performed on that pairs and new children will be generated. There are number of crossover methods available which use different strategies for performing crossover like single point crossover, two point crossover etc., for example like in single point crossover suppose the length of string is l, it can randomly choose an crossover point of the value between 1 to l-1. Now swaps the value of both string beyond the crossover point and it will result in generation of two new offspring. But one thing should be noticed here that the crossover is not always effective. So for this after choosing pair of string, we only apply crossover if the randomly generated variable between range 0 and 1 will be greater than $p_c$, where $p_c$ is the crossover rate.

#### 3.1.5 Mutation-
After crossover mutation will be performed. Mutation can be per-formed by flipping the binary bit of string. Like the crossover probability $p_c$, in the crossover which control the crossover, here we have probability of mutation (mutation rate) $p_t$, which gives the probability for change in bit. Mutation of bits are independent i.e. change in a bit will not affect the probability of change in another bit.
One important point is here that GA mostly treats mutation as secondary operator[3]. Primary operator is crossover. Mutation is mainly used for regenerating lost string of solution. For example suppose all available strings into solution converged to 0 at certain position and optimal

solution contain 1 at that position. So only by mutation we can get optimal solution by flipping that position bit.

## 4 PROPOSED ALGORITHM

Although traditional genetic algorithm mimic the process of natural selection and genetic and are very successful in solving optimization problems, now there are more questions which are to be answered. Can we apply genetic algorithms to those problems like (classification problem) in which we do not have any mathematical function to guide the search? Till now we have not designed any genetic algorithm for this type of problems. To design a genetic algorithm for those problems, let us review the literature of genetic algorithm with a different view. Genetic algorithm are used to maximize the value of certain objective function for any problem, Actually this type of function measures the features of a problem (by calculating the objective value at each sample point) and assigns some value to it. We can assume that with the help of this objective function we are assigning some membership value to the characteristics of a problem and genetic algorithm guide the search to find those solutions which have larger value of it. Selection, crossover and mutation operator guides the approach to get appropriate solution. But consider a case where we cannot assign an objective function (which provides a numerical value at a certain point) for the measurement of characteristics of a problem. We can use genetic algorithm for finding different solutions in a classification problem but in that case we have to design each operator differently. While solving a classification problem with genetic algorithm we will use these operators.

### 4.1 Selection

We will design a test function and always select those strings that qualify the selection criteria. Since all selected solutions are equally good so we will not replicate any value during selection operator. Taken an example in prime generating problem, prime testing function is the required test function. According to this function we will select only those values which are prime.

### 4.2 Crossover Operator

In normal crossover operator, we do not bother about the nature of generated child, we simply choose a cross site and generate two children. In case of classification problems as we do not have any mathematical function to guide our search so we will generate only those children which have some probability to pass the test. To generate child, we will apply some mathematical function. Here we will select only those mathematical function which generate child that strengthen our criteria for selection or in other way we can say that we will choose a mathematical function only when we can define the characteristic of generated child and the characteristics of generated child has some properties that has good probability to make him qualify for the selection.

For example, in our problem we need to generate new numbers that have a good probability of being prime. But how we judge that the generated child has a good probability of qualifying the primality test? To generate such type of child we will use the second definition of prime stated in above introduction. We will take two numbers that are prime and generate a child which is relative prime to its parents. In proposed algorithm, we have designed a function that will generate only one child, and this child will be relative prime to the parents involved in the crossover. One more strong argument that supports our idea comes from the theory of natural genetics. As we are using prime numbers in crossover so definitely generated child must inherit have some characteristics of prime numbers. The function which we used in our algorithm is shown below.

```
Proposed Algorithm: Crossover function
   Input: Two prime integers X and Y
   Output: An integer
1 If  Y > X
 2 Max   Y and Min X
 3 Else Max X and Min Y
 4 End If
 5 Convert Max on base Min
 6 Reverse the digits of generated number and generate a new number Z on base n
 7 Again convert Z back to decimal to get D
 8 return D
```

Our proposed algorithm does not require mutation operator because our selection operator is very efficient in maintaining the diversity among solutions.

## 5 EXPERIMENTAL RESULT AND ANALYSIS

In this section we will discuss the working of our algorithm. In our experiment we choose first sixty prime numbers as a set of initial population. We will apply some prime testing algorithm to generate first sixty primes. For example we can choose 2,3,5,7,11,13,17,.... up to first 60 prime numbers as initial population set. We can also increase the size of our initial population set. Now we can apply crossover operator to generate new primes. To apply crossover operator (proposed algorithm) we first choose any two prime numbers (generated randomly) from the given set. Let these prime numbers are 7 and 11. Then we will implement our crossover function. Since 7 are less than 11 so we will convert 11 on the base of 7. This will give us $(14)_7$. We will reverse these digits and generate a new number $(41)_7$. Now we will convert this number to original base (that is decimal). According to above example the decimal value of this number will be 29 which is defiantly a prime number. All new primes are important for us and we need bigger numbers so we will apply this algorithm as much as we can. In this case our set size is 60. Using this set at most ${}^{60}C_2$ combination can be produced, so we can apply crossover function up to 1770 times. We will store all new primes to our initial set. While storing, we discard all repeated prime numbers. For the next iteration our population set size will increase and that will be the input for next generation. We have applied the algorithm to generate new primes although we were not able to produce all primes but bigger primes were generated very efficient time. Also by using proposed algorithm the chances of generated new primes has also increased.

Various graphs presented below show the strength of our algorithm.

First graph represent the total number of crossover performed using proposed algorithm and total number of prime generated.

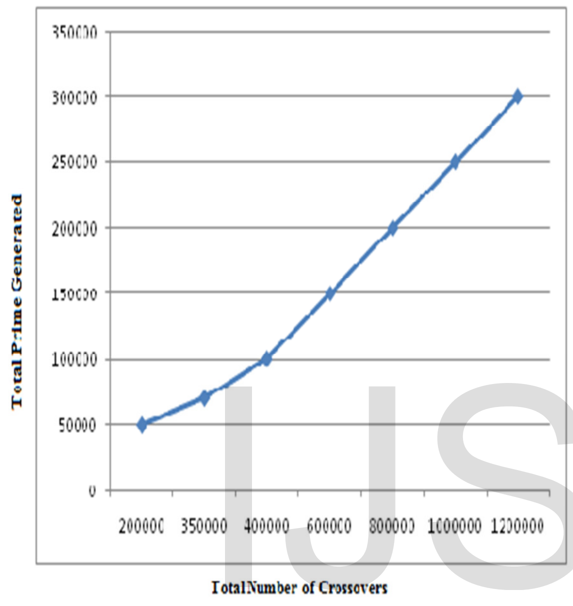

Figure 1: Total Prime Generated Vs. Total Crossover

Second graph shows the relationship between the total number of primes found and new prime generated.

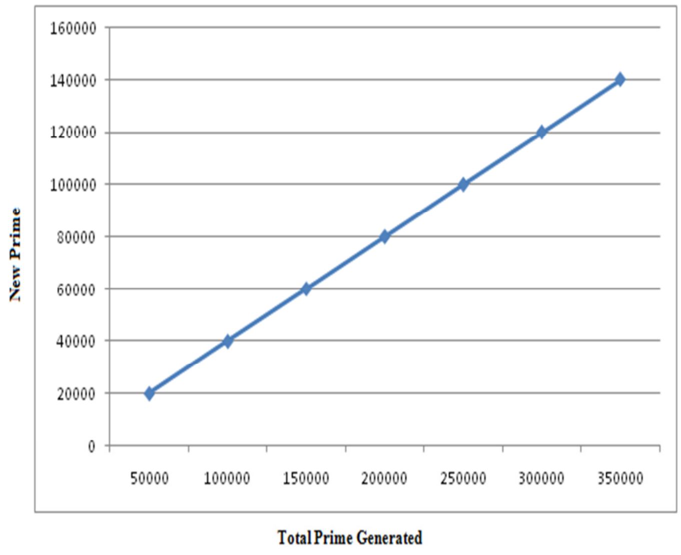

Figure 2: Total Prime Generated Vs. New Prime

Similarly third graph shows the relationship between total numbers of crossover performed and total new prime generated.

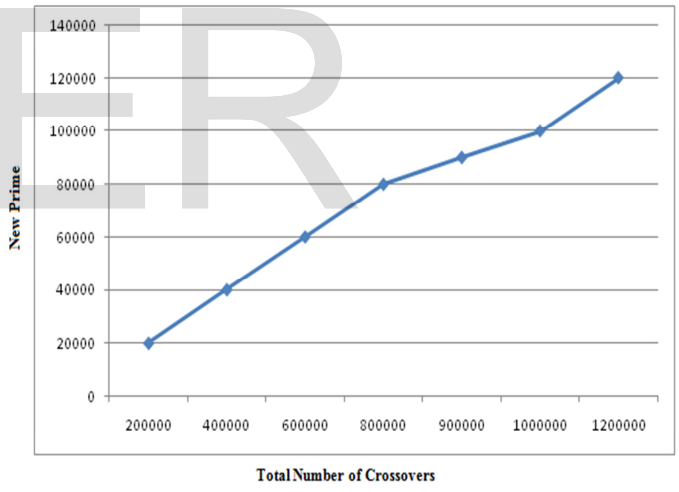

Figure 3: New Prime Generated Vs. Total Crossover

## 6 CONCLUSION & FUTURE SCOPE

We have proposed a new algorithm by which we can generate new primes in very less time. Various graphs drawn above show the strength of our proposed algorithm. It is well known that as the size of prime numbers increases, the problem of generating new prime become more difficult. Using our proposed algorithm we can generate new bigger primes in reasonable time. Proposed algorithm can be used with public key encryption techniques, digital signatures, public key distribution etc. Our proposed Algorithm can be further studied to reduce its memory complexity, so as to make it useful for small portable devices having lesser memory comparatively.

## REFERENCES

[1] Jerry Crow, “PRIME NUMBERS IN PUBLIC KEY CRYPTOGRAPHY AN INTRODUCTION”, SANS Institute 2002.

[2] Ray C.C. Cheung and Ashley Brown, “A SCALABLE SYSTEM-ON-A-CHIP ARCHITECTURE FOR PRIME NUMBER VALIDATION”, academia.edu, Nov 2005.

[3] Goldberg, E. David and John H. Holland, “Genetic algorithms and machine learning”, Machine learning, vol. 3.2, pp. 95-99, Springer, 1988.

[4] Jin. Yaochu,“A comprehensive survey of fitness approximation in evolutionary computation”, Soft Computing, vol. 9, pp. 3-12, 2005.

[5] Randy L. Haupt, “Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors”, In Antennas and Propagation Society International Symposium, IEEE, vol. 2, pp. 1034-1037, 2000.

[6] Al-Madi, Nailah and Simone A. Ludwig, “Adaptive genetic programming applied to classification in data mining”, In Nature and Biologically Inspired Computing, Fourth World Congress on IEEEs, pp. 79-85, 2012.

[7] R. L. Rivest, A. shamir and L. Adleman, “A method of obtaining digital signatures and public key crytosystems”, CACM, vol. 21, iss. 2, pp. 120-126, 1978.

[8] L. C. Guillou and J. J. Quisquater, “A practicle zero knowledge protol fitted to security microprocessor minimizing both transmission and memory”, Advances in cryptology-EUROCRYPT’88, LCNS, vol. 330, pp. 123-128, Berlin: Springer–Verlag, 1988.

[9] Domagoj Jakobovi and Marin Golub, “Adaptive genetic algorithm”, 1999.

[10] M. Joye and P. Paillier, “Fast generation of prime numbers on portable devices: An update”, CHES 2006, LNCS 4249, pp. 160-173, 2006.

[11] M. Joye, P. Paillier, and S. Vaudenay, “Efficient generation of prime numbers”, CHES 2000, LNCS 1965, pp. 340-354, 2000.

[12] D. Boneh and M. Franklin, “Efficient generation of shared RSA keys”, CRYPTO97, LNCS 1294, pp. 425-439, 1997.

[13] J. Brandt and I. Damgaard, “On generation of probable primes by incremental search”, CRYPTO92, LNCS 740, pp. 358-370, 1993.

[14] W. Bosma and P. P. Van der Hulst, “Faster primality testing”,EUROCRYPT89