# Analysis of Effect of Varying Crossover Points on Simple Genetic Algorithm (SGA)

Pradeep Kanchan
Dept. of Computer Science and Engineering
NMAM Institute of Technology
Nitte, India

Rashmi Adyapady R
Dept. of Computer Science and Engineering
NMAM Institute of Technology
Nitte, India

## ABSTRACT
The genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. A genetic algorithm is a search method that can be used for both solving problems and modeling evolutionary systems. The concept of the proposed paper is taken from simple genetic algorithm implementation using integer arrays for storage of binary strings as a basic ingredient. The Simple genetic algorithm (SGA) evaluates a group of binary strings and it performs crossover and mutation operation, which is the most important operation of genetic algorithm. SGA is successful if the final average fitness value is more than the initial average fitness value after crossover and mutation. This proposed paper deals with varying crossover points and observing its effect on SGA. Basically, the crossover point is varied from 1 to n (where n<=2) and observe its effect on both initial and final average fitness value. The probabilities of crossover and mutation are also varied. Then the proposed paper, Analysis of effect of varying crossover points on simple genetic algorithm is compared with the simple genetic algorithm implementation using integer arrays for storage of binary strings. Experimental results show that the proposed scheme significantly improves the performance of genetic algorithm.

## Keywords
Binary strings, fitness function, crossover, mutation, crossover probability, mutation probability.

## 1. INTRODUCTION
Genetic algorithm (GA) is a heuristic technique used to solve optimization problem. Optimization attempts to find the best solutions for a given problem. GA's belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution.

The basic techniques of the GA's follow the principles given by Charles Darwin "survival of the fittest". The GA aims to produce 'offspring' better than the parents. In every generation, a new set of strings is created from the fittest of the old; a new part is used for good measure. Genetic Algorithms (GAs) are search algorithms based on the mechanics of the natural selection process. The basic concept is that the strong ones tend to survive and weak ones are left out.

GA's have the ability to create an initial population of feasible solutions. Each feasible solution is encoded as a chromosome, and each chromosome is given a measure of fitness using fitness function. The fitness of a chromosome determines its ability to survive and produce offspring. A finite population of chromosomes is maintained. Choosing a population size too small increases the risk of converging prematurely to local minima i.e., being stuck within neighboring set of strings. A larger population has a greater chance of finding the global optimum at the expense of more CPU time.

Genetic algorithm involves three types of operators: *Selection*, *crossover*, and *mutation*.

*Selection:* This operator selects chromosomes in the population for reproduction based on their fitness. The fitter the chromosome, the more times it is likely to be selected to reproduce relatively from the large number of initial population.

*Crossover:* After the completion of selection process, crossover operator randomly chooses a position and exchanges the bits between two chromosomes (parent chromosomes) to create two offspring (new chromosomes). The crossover operator roughly imitates biological recombination between two single chromosome organisms.

*Mutation:* Mutation is performed after crossover operation by randomly choosing a chromosome in the new generation to mutate. This operator randomly chooses a position and flips some of the bits in the chromosome. Mutation can occur at each bit position in a string with some probability.

The following steps are repeated until a solution is found:

1. Initially a large population of random chromosome is created. Let's say there are N chromosomes in the initial population.
2. Assign fitness to each chromosome using fitness function criteria. $F(x) = x^2$
3. Select two fittest members (pair of parent chromosomes) from the current population.
4. Perform crossover and mutation.
5. Repeat step 3, 4 until a new population of N members has been created.

The SGA is said to be successful if the final average fitness of the population after mating (crossover) is better than the initial average fitness. Probability and randomness are also essential parts of GA.

## 2. LITERATURE SURVEY
### 2.1 SGA Implementation using Integer Arrays for Storage of Binary Strings
–Pradeep Kanchan, Rio D'souza

Abstract—The Simple Genetic Algorithm evaluates a group of binary strings based on their fitness, performs crossover and mutation on them and tries to generate a group having maximum fitness. The usual method used for SGA implementation was by using character arrays for storage of binary strings. In this paper, the authors have implemented the SGA using integer arrays for storage of binary strings. Then, the initial average fitness is compared with the final average fitness so that the working of SGA can be verified.

The SGA implementation given by the authors:

i. Initially puts the chromosome into a pool to be evaluated.
ii. Finds the decimal equivalent of the chromosome and calculates its fitness based on the fitness function criteria. $F(x)=x^2$
iii. Selects the chromosomes to be mated, and performs crossover and mutation operation. Until the term condition is not reached. If the new generation matches the acceptance criteria they are added to the generation.

Specific Contribution—In this paper, the authors implement the SGA algorithm using integer arrays. Generally, SGA was implemented using character strings for representation of chromosomes. But, this method had some disadvantages because the storage space was limited. Also, this approach works for bit strings of small length but not for bit strings of larger length.

## 2.2 Evolving a Computer Program to Generate Random Numbers using the Genetic Programming Paradigm

–John R Koza

Abstract—This paper gives an idea that it is possible to use genetic programming paradigm to breed a computer program to perform the task of converting a sequence of consecutive integers into pseudo-random bits with near maximal entropy using a randomizer.

Steps given by author for breeding a randomizer:

i. To identify the set of terminals. The terminals in the genetic programming paradigm correspond to the input being genetically bred to the computer program.
ii. To identify a sufficient set of functions for the problem.
iii. To identify fitness function for evaluating how good a given computer program is at solving the problem at hand.
iv. Selecting the values of certain parameters. The important parameter here is population.
v. Criterion for terminating a run and accepting a result.

Specific Contribution—The goal of this paper was to genetically breed a computer program to convert a sequence of consecutive integers into a sequence of random binary digits. This paper gives an idea of how to go about generating random numbers. The Author gives the implementation of SGA which makes use of random numbers (random bits 0 and 1) for the creation of a chromosome. Usually SGA involves both crossover and mutation but in this paper mutation is not done by the author. This paper stops the SGA when 51 generations are done.

## 2.3 Solving ISP Problem by using Genetic Algorithm

–Fozia Hanif Khan, Nasiruddin Khan, Syed Inayatulla, Shaikh Tajuddin Nizami

Abstract— The purpose of this study was to propose a new representation method of chromosomes using binary matrix and to use a new fittest criteria as a method for finding the optimal solution for TSP. In this paper the authors are introducing new fittest criteria for crossing over, and have applied the algorithm on symmetric as well as asymmetric TSP, also presented asymmetric problem in a new and different way.

Steps of algorithm given by the authors:

i. Randomly create the initial population of individual strings of the given TSP problem and create a matrix representation of each.
ii. Fitness is assigned to each individual in the population using fitness criteria measure. The selection criteria depend upon the value of strings if it is close to 1.
iii. By applying crossover and mutation operation new offspring population of strings are created from the two existing strings in the parent population.
iv. If required the resultant off-springs are mutated.
v. New offspring is called as parent population and step (iii) and (iv) are continued until a single offspring which will be an optimal or near optimal solution to the problem is got.

Specific Contribution—In this paper authors have given a procedure for solving TSP by using genetic algorithm. The authors provided the fittest criteria and applied the algorithm for symmetric as well as asymmetric TSP which is used to solve the problem easily.

# 3. PROBLEM STATEMENT

The first step will be to implement the SGA algorithm. In the paper [1], the effect of mutation and crossover is studied, but the crossover points are not varied by the authors. This paper deals with varying crossover points on SGA with crossover and mutation probabilities. Basically, crossover points are varied from 1 to n (where n<=2) and its effect on both the initial and final average fitness value is observed. Hence the effect of varying crossover points on SGA will be studied.

# 4. PROPOSED METHODOLOGY

## 4.1 Algorithm

1. Generate random population;
2. Evaluate initial fitness using fitness function;
3. Evaluate initial average fitness;
4. While (generation<max generation and CP<1 and MP>0)
5. selection of two individuals.
6. N-Point crossover(); //for N=1, 2
7. Mutation();
8. CP=CP+0.01;
9. MP=MP-0.001;
10. Form new population;
11. End while
12. Show max average fitness;

In the above algorithm, CP stands for crossover probability and MP stands for mutation probability.

## 4.2 Initial Design

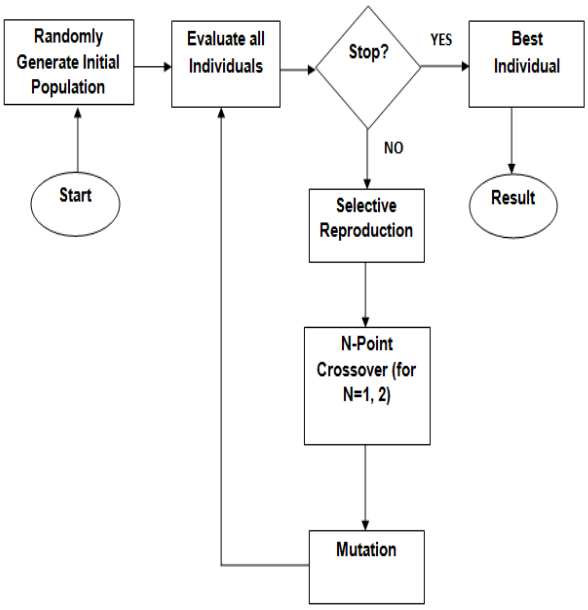


**Figure 1: Block Diagram of working of SGA**

## 5. COMPARISON OF BOTH ALGORITHM

For comparison and analysis, we used SGA implementation using integer arrays for storage of binary strings and Analysis of effect of varying crossover points (one point and two point crossover) on simple genetic algorithm.

**Table 1. Result of Paper [1]**

| Runs | SGA without probability variation | | | |
|---|---|---|---|---|
| | Pop size | Max gen | Initial Avg Fitness | Final Avg Fitness |
| 1 | 50 | 10 | 279.7 | 897.8 |
| 2 | 100 | 10 | 306.33 | 882.51 |
| 3 | 200 | 50 | 318.275 | 941.345 |
| 4 | 200 | 100 | 296.245 | 929.495 |
| 5 | 300 | 100 | 313.7833 | 928.7933 |

**Table 2. Result for One Point Crossover**

| Runs | Analysis of effect of varying crossover points on SGA with probability variation for one point crossover | | | |
|---|---|---|---|---|
| | Pop size | Max gen | Initial Avg Fitness | Final Avg Fitness |
| 1 | 50 | 10 | 7265.98 | 20401.66 |
| 2 | 100 | 10 | 12185.85 | 37354.31 |
| 3 | 200 | 50 | 29123.06 | 88209 |
| 4 | 200 | 100 | 28270.36 | 88209 |
| 5 | 300 | 100 | 53423.89 | 158404 |

**Table 3. Result for Two Point Crossover**

| Runs | Analysis of effect of varying crossover points on SGA with probability variation for two point crossover | | | |
|---|---|---|---|---|
| | Pop size | Max gen | Initial Avg Fitness | Final Avg Fitness |
| 1 | 50 | 10 | 6136.1 | 21073.18 |
| 2 | 100 | 10 | 12573.27 | 37721.41 |
| 3 | 200 | 50 | 29272.01 | 89401 |
| 4 | 200 | 100 | 29045.33 | 89401 |
| 5 | 300 | 100 | 56491.15 | 159201 |

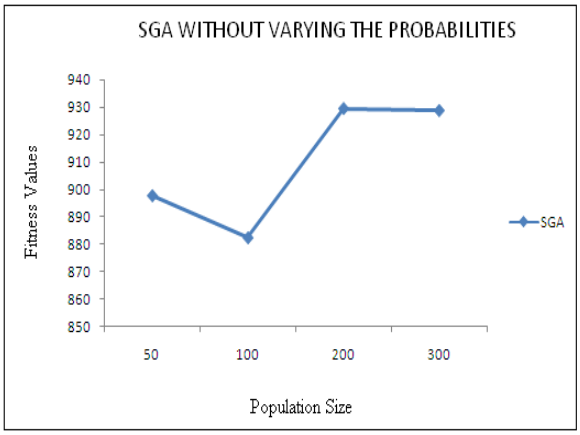


**Figure 2: SGA without variation in probabilities**

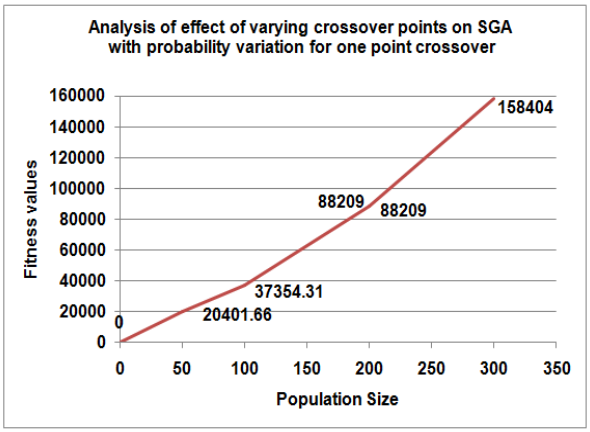


**Figure 3: Analysis of effect of varying crossover points on SGA with probability variation for One Point Crossover**

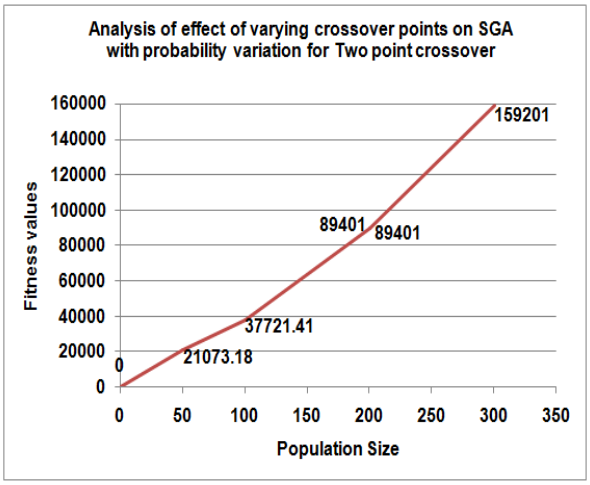


**Figure 4: Analysis of effect of varying crossover points on SGA with probability variation for Two Point Crossover**

We observe from all the above graphs that the Analysis of effect of varying crossover points on SGA with variations in probabilities for both One Point Crossover and Two Point Crossover works more efficiently as compared to the SGA without variations in probabilities.

## 6. CONCLUSION

In this paper, we have placed more emphasis on varying crossover points on simple genetic algorithm. Basically, the crossover point is varied from 1 to n (where n<=2) and its effect on both the initial and final average fitness is analyzed. Then, Analysis of effect of varying crossover points on simple genetic algorithm is compared with the paper [1].

In the paper [1], the crossover points are not varied by the authors. In the proposed paper, the crossover points are varied such as one point crossover and two point crossover with crossover and mutation probabilities and its effect on the final average fitness is analyzed. We have seen that the fitness increases as crossover probability increases and mutation probability decreases. We have come to the conclusion that the final average fitness value after the crossover (one point and two point) is more than the initial average fitness value. Experimental results show that the proposed scheme significantly improves the performance of genetic algorithm and its better than paper [1].

## 7. ACKNOWLEDGMENT

I would like to thank my project guide Mr. Pradeep Kanchan. I extend my thanks to my H.O.D, Mr. Prasanna Kumar H. R. and Principal, Dr. Niranjan N. Chiplunkar. I thank the almighty, my parents, and friends for their constant encouragement.

## 8. REFERENCES

[1] Kanchan, Pradeep, D'souza Rio, "SGA implementation using integer arrays for storage of binary strings", International Conference on Machine Learning and Computing (ICMLC), 2010.

[2] Koza, John R., "Evolving a computer program to generate random numbers using the Genetic Programming paradigm", Proceedings of the Fourth International Conference on Genetic Algorithms, 1991.

[3] Khan, Fozia Hanif, Khan, Nasiruddin, Inayatulla, Syed, Nizami, Shaikh Tajuddin, "Solving ISP problem by using Genetic Algorithm", International Journal of Basic & Applied Sciences IJBAS-IJENS, Vol:09 No:10.

[4] Koza, John R., "Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems", Proceedings of the 2nd International IEEE Conference on tools for Artificial Intelligence, 1990.

[5] Mitchell, Melanie, "An Introduction to Genetic Algorithms", 1998 :Prentice Hall.

[6] Goldberg, David E., "Genetic Algorithms in search, Optimization and Machine Learning", 1989 :Addison-Wesley.

[7] Davis, L., "Genetic algorithms and simulated annealing", 1987 :Pittman.

[8] Holland, J. H., "Adaptation in natural and artificial systems", 1975 :University of Michigan Press.