

## **ANALYSIS OF GENETIC ALGORITHM ON KAUFFMAN'S NK LANDSCAPE**

**17593**

**ARAVIND RAJESH**

## Table of Contents

<b>1 GENETIC ALGORITHM.....</b>	<b>3</b>
<b>1.1 Limbs of Genetic algorithm.....</b>	<b>4</b>
<b>2 WORKING OF GENETIC ALGORITHM.....</b>	<b>5</b>
<b>3 NK LANDSCAPE MODEL.....</b>	<b>5</b>
<b>4 WORKING OF ALGORITHM.....</b>	<b>7</b>
<b>5 DISCUSSIONS.....</b>	<b>8</b>
<b>6 CONCLUSIONS.....</b>	<b>11</b>

## APPENDIX

Abstract: The main goal of the project is to get a better understanding of genetic algorithm, its performance with various parameters and its usefulness in optimization problem when compared with other learning algorithms. Performance of genetic algorithm is tested on a simple optimization problem which is called NK landscape developed by Kauffman. It offers tunable landscape with dynamically alterable problems size and epistasis. A series of simulation is run with variety of population size and epistasis, for a greater understanding of relationship of landscape with algorithm. We compared the performance of genetic algorithm with two variants ,one with mutation only and other with crossover and mutation .We study their ability such as speed, efficiency and ability to attain optimal goal on these landscape and what is the comparative performance of these two variants are discussed here. Finally it is shown “crossover with mutation” variant underperforms mutation only on a NK landscape with greater problems size and epistasis, due to deprived genetic diversity population in complex landscapes.

## 1 GENETIC ALGORITHM

Genetic algorithm fall under the group of evolutionary algorithms, they are typical replication of the evolutionary process in living things as described by Charles Darwin. They find usage in engineering, simulation, computers as well in other fields. Genetic algorithms have been extensively used for optimization task problems of complex nature, path /solution finding problems with randomness; The evolutionary search strategies can be used with genetic algorithm for out producing result when compared with other learning algorithms. Genetic algorithms design implementation involves replication of evolutionary process in nature like mutation, selection, crossover and inheritance.

For example a list of population is replaced with a set of new offspring whose characteristics are directly inherited from parents at the same time maintain a group of random individuals. The perseverance of the best solution in each generation and passed on to next is the main criterion of Evolutionary algorithms.

The use of genetic algorithm for a variety of test function and test problems has already been attempted by many researchers for studying their performance(Merz & Freisleben 1998).Genetic algorithms are well suited to be tested on test function problem to evaluate their performance . Variety of problems that makes Genetic Algorithm to underperform are epistasis, problem size etc. The tunable rugged landscape provided by NK landscape model( Kauffman & Weinberger 1989; Kauffman & Levin 1987) provides a suitable problem for testing the performance of genetic algorithm. Kauffman's NK landscape model is one such problem with characteristics that make the problem hard for any optimization algorithm. Epistasis is one such characteristics that allows us to test the efficiency of Genetic algorithm (Davidor,1991 ,cited in Merz & Freisleben 1998).(Aguirre & Tanaka 2003) In biology, the term epistasis is described as the effect of one gene, mask or influence the effect of another gene. Such interaction can be seen in genomic level of organism. Whereas in genetic algorithm term epistasis describes the nonlinearity in fitness function, which alternates depending on the values of interacting bits. The size of the problem is another important factor when considering evolutionary algorithms, as we have only little information on their performance for problems of larger problems size and complex landscapes. Recently several study being conducted on the performance of evolutionary algorithm for a variety of problem scape to analyze and observe the best evolutionary strategies on these landscapes. NK landscape provides such landscape of increased evaluation on performance of Genetic Algorithm.

### 1.1 Limbs of Genetic Algorithm

#### 1.1.1 GENES

Chromosomes are a set of genes grouped together. Genes can be understood as a molecular element of information in heredity. In terms of genetic algorithm individual genes are considered as set of individual's .For any genetic algorithms the initial set of population called genes is represented individuals binary strings. These form the candidate solution for the given problem task. These initial set of genes are further mutated, selected and inherited for subsequent generation until a required goal is reached.

**1.1.2 MUTATION**

Mutation is generally considered as a mistake while copying the best characters from parent to the children. In other words it is a random change of value of the gene in the population

**1.1.3 SELECTION**

Selecting the best parent out the population is the key in genetic algorithm. In nature this process is called as Survival of the fittest. In genetic algorithm selection process is always accompanied with fitness function which allows selecting the best, in a way fitness function allows the algorithm to implicitly point toward the goal. Each individual from population is checked for the fitness function and the best is chosen for the next generation. Thus moving towards the goal in each step.

**1.1.4 Crossover**

Crossover is the process of combining blocks of information from parents and, forms an entirely new chromosome [children] at the same time holding the best information intact. Crossover has the primary role of combining blocks parents chromosomes to create a variety in the fourth coming generation .There are many different types of crossovers e.g. single point crossover or multipoint crossover.

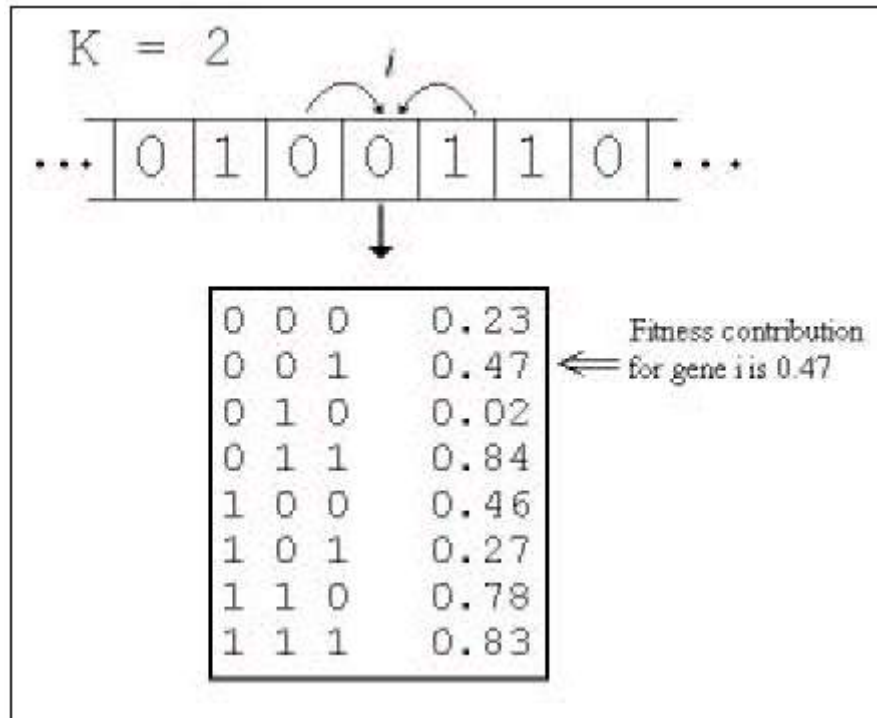
The best individual will undergo a crossover with other best individual selected from the population .Thus the new population created from crossover averages to be better.

**2 WORKING OF GENETIC ALGORITHM**

Genetic algorithms are initialized with required set of chromosomes which forms the initial population. Each chromosome is an individual bit string and is populated with individuals called genes or bits .This acts as a candidate solution of the problem. Each individual is evaluated with a fitness function described by the user .The fitness function allows the algorithm to choose the individual that best adapt (fit) to that environment. The best individual is then applied to evolutionary strategies or functions such as mutation or crossover with certain randomness, further the next generation until the goal is reached.

**3 NK LANDSCAPE MODEL**

NK landscape model (Kauffman & Levin 1987) is introduced as an adaptive walk on rugged landscapes. NK landscapes are well known examples for test function generators with randomly generated fitness function and non-linearity. NK landscape model can be visualized with an agent seeking to move in a rugged landscape from fitness valley to fitness peak. NK landscape is initialized as a bit string and is characterized by two parameter N for the overall number of genes (genes represented in bits) each agent can have and K for the neighborhood size. For each bit K defines the number of bits it interacts with this effect is called as epistasis. Epistasis is the interdependency of bit's fitness value and each bit acts in gene has an effect on the total fitness. For example one bit may improve the fitness by itself but may decrease the fitness contributed by other bits with which it interacts(Aguirre & Tanaka 2003,p.2270).



**Fig 1.** Epistatic link between neighbors for  $k=2$  and fitness table .Here each gene is linked to two other genes each gene has a fitness contribution of its own. Adapted from(Kauffman 1993,p 42),Figure2.2

The main advantage of NK landscape is the, variably tunable ruggedness of landscape with the parameter  $K$ . For  $K=0$  there is no epistasis-relation between each other individual genes. The overall fitness contribution is by itself. Thus resulting landscape is smooth with one peak .When this goes to  $K=(N-1)$  where each individual epistatically interact with every other individual and thus the resulting landscape is rugged. The fitness  $f$  of any gene ( $d$ ) of length  $N$  is given by adding all the fitness values each gene. i.e.  $f(d)=d_1 \dots \dots d_N$

$$f(d) = \frac{1}{N} \sum_{i=1}^N f_i(d_i, d_{i1}, \dots, d_{iK})$$

(3a) adopted from (Vidgen et al. 2008)

Where  $f_i$  is the fitness contribution of gene at point  $i$ . The function  $f_i : P^{K+1}$  provides the fitness function for the gene  $d_i$  and  $d_{i1} \dots \dots d_{iK}$  are the  $K$  bits interacting with  $d_i$ . Here the value of  $P=2$ , which represent the binary string of gene population (0 or 1). When  $K=2$  so  $2^{2+1} = 8$  bits forms a fitness table list .The land scape can be a visualizes as a  $N$  dimensional hyper cube where each point in landscape can be identified as a  $N$  bit string e.g.: 00001, 00010 and  $P$  act as a digit base(for binary  $p=2$ )( Kauffman 1993,pp.40-50).  $P^N$  Gives the total number of coordinate points on the landscape.

There are two ways pointed by Kauffman by which each gene can be epistatically linked to its neighbors. **A.** to select the individual to the left and right of the bits selected, if the bit selected is at either of the edge then selection of neighbors is carried in wrap around sequence random –  $K$  neighbors are selected in random among  $N$  bit strings. **B.** all the neighbors are selected in a random order.

Works of (Aguirre & Tanaka 2003; Merz & Freisleben 1998)(Pelikan et al. 2009) and many others provides a detailed study of performance of evolutionary algorithms on NK landscape which acts as a benchmark for comparison.

#### 4 WORKING OF ALGORITHM

This section is used to explain working of the algorithm in steps.

The agent chooses a random location from a set of initial population of bit length  $N$ . Total search space population is calculated from  $P^N$  for e.g. here  $P=2$  (always a binary string for representing genes) and  $N=8$ ,  $2^8 = 256$  states of initial population is generated (from 00000000 to 11111111).

Links between genes are generated in a random fashion, for a particular gene at its locus  $i$  the neighbors are chosen in random fashion. The number of combination for genes are  $2^{K+1}$ , for each combination fitness score is assigned in random fashion drawn from a uniform distribution between 0.0 to 1.0. Fitness of a gene at  $i^{th}$  locus is the fitness contribution of that gene and all other  $K$  epistatically connected gene. In order for the agent to move or travel through the landscape, the method of fitter dynamics (Vidgen et al. 2008) where the fitness of the new location is compared with the current location if it is greater, the agent moves otherwise remains in same location until a fitter location is found.

In each generation a certain number of offspring's are generated (here size of offspring is 30 for each generation). The best fit parent is chosen for the population of next generation, with a small rate of mutation probability, the best fit parent are mutated for the next generation in first variant and crossover and mutation function is used for the second variant of algorithm. The mutation probability decides the survival of the parent information. Higher mutation rate imply that more chances of getting mutated and vice versa. From the parent chromosome, the gene at a location pointed by the randomly generated index is chosen. The bit at that location is flipped, i.e. if the bit is 0 (then flip to 1), if 1 (flip to 0). All offspring's are evaluated the changes that result in better solution are retained as best fit parents and used for population next generation. Each test runs for a 1000 generations.

Parallely after each generation the best fit parent and corresponding best fitness, average fitness of each generation and worst fitness for each generation is recorded. For various different values of  $N$  and  $K$  1000 generation are run and the above said parameters are recorded.

A random landscape is chosen initially with that, agent initiate random walk with 30 offspring generated for each generation with added mutation to the best fit parent of previous generation. Average fitness of each generation is calculated for different  $N$  and  $K$  parameters and tabulated below

	N			
K	4	8	16	22
0	0.68	0.7	0.71	0.68
2	0.70	0.68	0.68	0.68
4	0.65	0.66	0.67	0.7
8		0.56	0.61	0.68
16			0.58	0.58

**Fig 4(a):** For various N and K, average fitness (random interaction) value for 1000 generation K parameters.

The table above obtained by our simulation is compared with the results obtained by Kauffman in (Kauffman 1993). We can see that results obtained have significant equivalence.

	N	
K	8	16
0	0.65	0.65
2	0.7	0.7
4	0.7	0.71
8	0.66	0.68
16		0.65

**Fig 4(b):** For various N and K values average fitness (with random interaction )(Kauffman 1993)

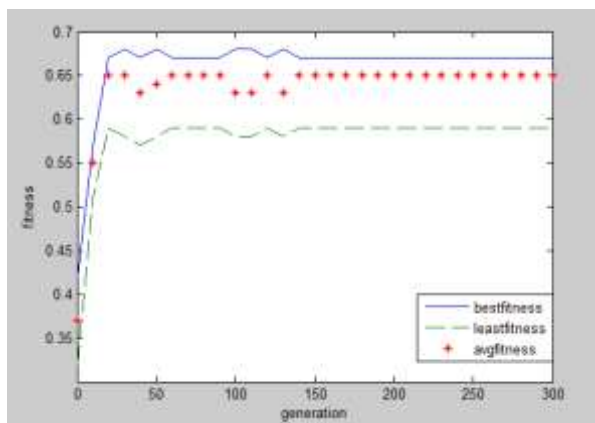
## 5 DISCUSSIONS

From the tables above in Fig4 (a) we are able to analyze that the average fitness obtained by genetic algorithm for a N decreases as K increases. Further for a fixed K value the N increases but the average fitness does not fall. Here we are able to understand how well a Genetic algorithm perform when the problem size is increased with tuning the parameter N. Increase in dimension of the search space does not influence the fitness with great effect rather genetic algorithm tends to reach less average fitness in landscape with high epistatic interaction. The algorithm usually buckles down in reaching fitter optimal value with highly rugged landscapes but relatively shows a good performance with increase of search space.

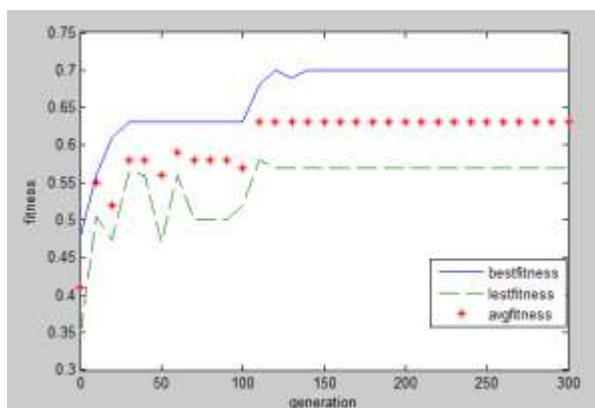
Individual fitness contribution bends the fitness landscape for finding the local optima. Thus the average fitness of the landscape decreases as shown in Fig 4(a). Recent works confirm the existence of positive correlation between K and number of peaks (Smith, RE & Smith, JE 1999) as cited in (Aguirre & Tanaka 2003) and the global optima increases for small K (Mathias et.al 2001) as cited in (Aguirre & Tanaka 2003). Thus we are able to show that as K increases the average fitness dwindles due to K value is directly related to the number of peaks. Mutation function in genetic algorithm has added edge to the performance when compared with its variant of crossover with mutation.



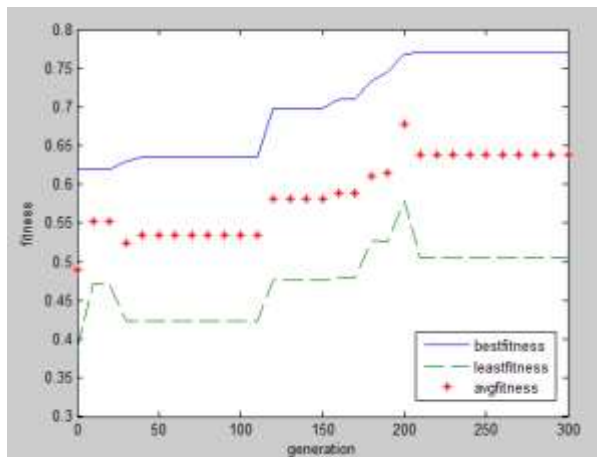
Further for a fixed K value the N increases but the average fitness does not fall, this behavior of algorithm for an increasing search space suggests that, the algorithm gets stuck in the local fit optimal value without traversing better fitness location. One of the primary reasons is variety of available population to search is restricted in some way. When the agent reaches the first location the fitness of that location is taken as the best, from there the neighboring location list is generated with mutation probability. This provides a new set of population for the agent to search through much fitter location. Reduced variety of population mutation function of the algorithm is the reason why large search space is often unseen and the agent stays in the local optimal region.



**Fig 5(a):** Graph showing best, least and avg fitness values for N=16 and K=0.



**Fig 5(b):** Graph showing best, least and avg fitness values for N=16 and K=4.



**Fig 5(c):** Graph showing best, least and avg fitness values for N=16 and K=8

With Crossover and mutation

All the Fig 5(a, b, c) shoes how the fitness converges across generation and stuck in optimal fitness.

To study the comparative performance of Genetic algorithm a variant with crossover and mutation is introduced. The crossover is a two point crossover. The threshold value for applying crossover is set 0.02. The interaction genes for each locus are selected randomly among N.

	N			
K	4	8	16	22
0	0.60	0.65	0.65	0.64
2	0.48	0.63	0.62	0.68
4	0.51	0.56	0.66	0.67
8		0.58	0.65	0.64
16			0.56	0.60

**Fig 5(d):** For various N and K, average fitness value for 1000 generation

Comparing the two tables 5d and 4a the performance of GA with two point crossover and mutation (GA 2PCM) is below the performance of GA with mutation (GA M) only. The relative difference in performance can be attributed to the increasing search space. For search space of mid-size i.e.  $N > 8$  and  $N < 17$  GA with mutation only perform well and above that as mentioned earlier the average fitness never fell below. But in the case of GA 2PCM average fitness fall with increase in search space. However as the search space and neighborhood size increases the effective performance for both GA variants drops. The main reason for this behavior is duplicates of offspring's are created in each generation, however more number of duplicates offspring's were spotted from each generation and the proportion of duplicates created increases as problem space increases for GA 2PCM than GA M. This creates a selective pressure on choosing the best fit offspring and gets stuck in local optimal fit individual. Mutation variant remains preferable over the other. And GA M seems to perform well in all cases and is more preferable for dimension of large search space.

## 6 CONCLUSION

Performance of genetic algorithms with different variants was tested on NK landscape. The two variants are Genetic algorithm with crossover and mutation and Genetic algorithm with mutation only. The performance of Genetic algorithm with mutation performed better than former for a broad class of problems cases. This is contrary to the assumption that GA 2PCM will perform better for the reason it has crossover and mutation which will produce a variety of offspring. The primary reason for not expected performance can be related due to the ineffective recombination and premature convergence of optimal solution. From the results we are able to draw that both algorithms perform considerable well on smaller search space and neighborhood size, but when it is extend to complex landscape and higher epistasis relative degradation of performance is noticed in GA 2PCM . The performance of GA with mutation stands preferable over GA 2PCM for higher search space

Thus the NK landscapes provided a good tunable problem space of different complexity and dimensionality allowing us to test the behavior and overall performance of genetic algorithm and functionality involved in GA. This helps us to get insight of how GA work on different problem space and how their effective performance can be tuned to suit different problems. How they can implemented for solving real world problem of different complexity.

The study on other parameter like selection , different types of crossover, genetic drift ,are other useful areas where one can study how their variation affect the fitness landscape try to overcome it . Correlation among variables and their impact is another area of exploration with GA.

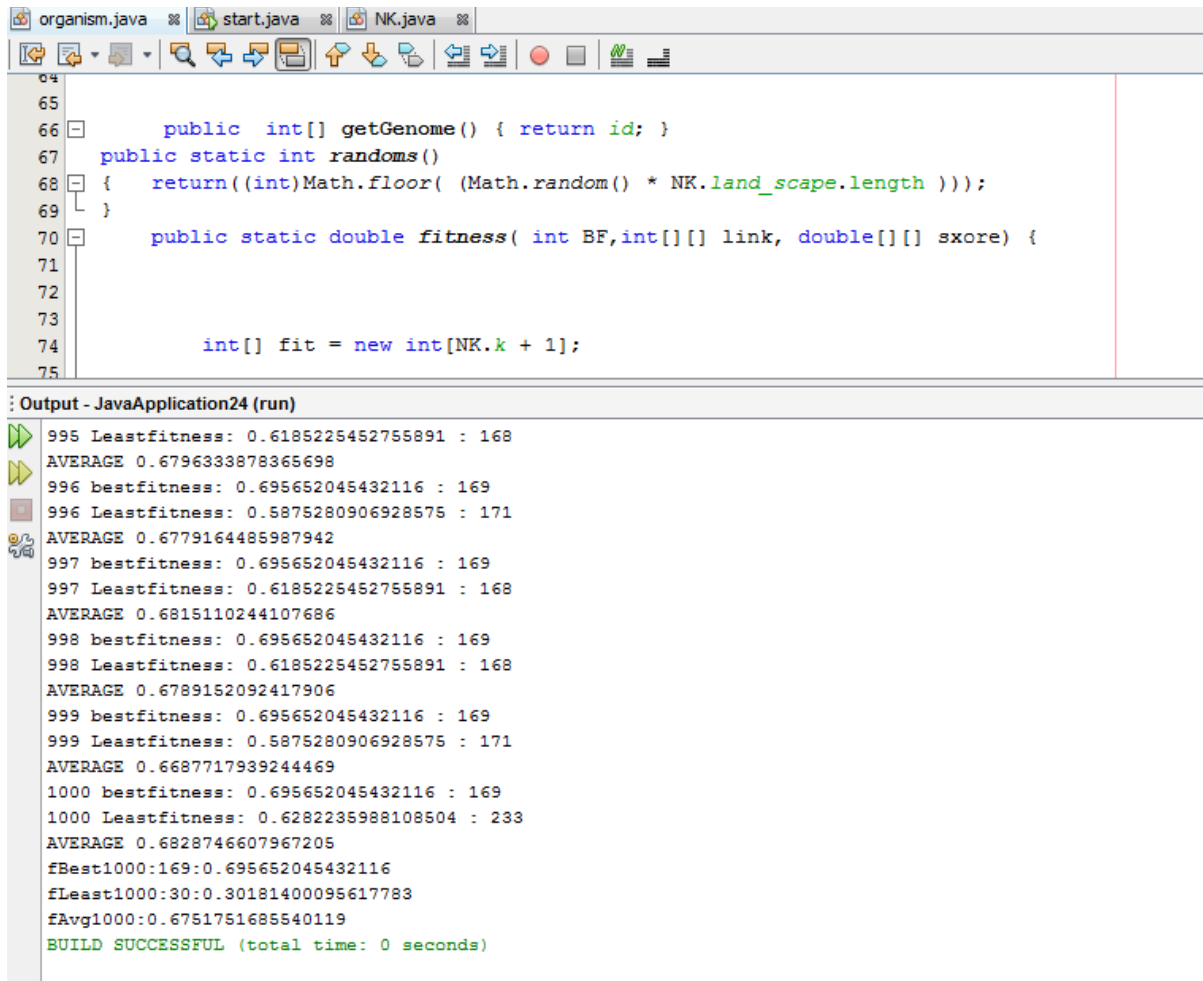
### Future works

Future works to be pursued in the direction of comparing different other evolutionary algorithms along with Genetic algorithm to find the effectiveness of each. Already research have been conducted and performance and effectiveness are of evolutionary algorithms are compared with local search algorithms.

Addition of neutrality to the NK landscape, and what difference it makes to the performance of evolutionary algorithm is a potential future works. Neutrality provides ridges in the NK landscape, which allows mutation to make no change to overall fitness contribution of the genome and help to understand further the relationship between different function of GA and complexity of problems.

## APPENDIX

### A.SCREEN SHOTS



The screenshot shows an IDE with three tabs: `organism.java`, `start.java`, and `NK.java`. The `start.java` tab is active, displaying the following code:

```

64
65
66     public int[] getGenome() { return id; }
67     public static int randoms()
68     { return((int)Math.floor( (Math.random() * NK.land_scape.length)));
69     }
70     public static double fitness( int BF,int[][] link, double[][] score) {
71
72
73
74         int[] fit = new int[NK.k + 1];
75

```

Below the code editor, the `Output - JavaApplication24 (run)` window shows the following output:

```

995 Leastfitness: 0.6185225452755891 : 168
AVERAGE 0.6796333878365698
996 bestfitness: 0.695652045432116 : 169
996 Leastfitness: 0.5875280906928575 : 171
AVERAGE 0.6779164485987942
997 bestfitness: 0.695652045432116 : 169
997 Leastfitness: 0.6185225452755891 : 168
AVERAGE 0.6815110244107686
998 bestfitness: 0.695652045432116 : 169
998 Leastfitness: 0.6185225452755891 : 168
AVERAGE 0.6789152092417906
999 bestfitness: 0.695652045432116 : 169
999 Leastfitness: 0.5875280906928575 : 171
AVERAGE 0.6687717939244469
1000 bestfitness: 0.695652045432116 : 169
1000 Leastfitness: 0.6282235988108504 : 233
AVERAGE 0.6828746607967205
fBest1000:169:0.695652045432116
fLeast1000:30:0.30181400095617783
fAvg1000:0.6751751685540119
BUILD SUCCESSFUL (total time: 0 seconds)

```

### B.CODE

#### Block (a): MAIN PROGRAM

```

public class start {

    public static void main(String[] args) { //main function

        NK ga = new NK();
        ga.run(8, 0);
    }
}

```

```

    }
}

```

### Block (b):NK.JAVA

```

import java.util.*;
import java.util.Map.Entry;

public class NK {

    public static double[] land_scape; // total number of bit strings
    public static int n; //param size
    public static int k; //param epistasis
    public static int P = 2;
    public static int Number_offspring = 30;

    public NK() {
    }

    public void run(int n_, int k_) {
        n = n_;
        k = k_;
        land_scape = new double[(int) Math.pow(P, n)];

        // fills neighborhood for each gene

        int[][] links = new int[n][k + 1];

        for (int i = 0; i < n; i++) {
            links[i][0] = i;
            for (int j = 1; j < k + 1; j++) {
                links[i][j] = (int) (Math.random() * n);
            }
        }

        // fitness table filled with scores
        int numscore = (int) Math.pow(2, k + 1);
        double[][] scores = new double[n][numscore];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < numscore; j++) {
                scores[i][j] = Math.random();
            }
        }
    }
}

```

```

int generation = 0;
int lorgs; //Individual organism
double temp;
double avgValue = 0.0;
double bestfitness;
double leastfitness;
LinkedList<Integer> individual = new LinkedList<Integer>(); // stores individual from each
generation
LinkedList<Double> AVG = new LinkedList<Double>(); // stores the average fitness of each
generation
HashMap<Integer, Double> MIN = new HashMap<Integer, Double>(); //stores the
minimum fitness of each generation
HashMap<Integer, Double> MAXI = new HashMap<Integer, Double>(); //stores the
maximum fitness of each generation
organism org = new organism();
organism[] offspring = new organism[Number_offspring]; // create & store children for
each generation
int orgs = binToInt(org.getGenome()); // hold the individual temporarily
while (generation < 1000) {
    temp = 0.0;
    individual.clear();
    generation++;
    for (int i = 0; i < Number_offspring; i++) {
        offspring[i] = new organism(intToBin(orgs));
        individual.add(binToInt(offspring[i].getGenome()));
    }
    // To find the bestfitness
    orgs = individual.get(0);
    bestfitness = organism.fitness(individual.get(0), links, scores);
    for (int i = 1; i < Number_offspring; i++) {
        if ((organism.fitness(individual.get(i), links, scores)) > bestfitness) {
            orgs = individual.get(i);
            bestfitness = organism.fitness(individual.get(i), links, scores);
        }
    }

    System.out.println(generation + " bestfitness: " + bestfitness + " : " + orgs);
    MAXI.put(orgs, bestfitness);

    //To find least fitness
    lorgs = individual.get(0);
    leastfitness = organism.fitness(individual.get(0), links, scores);
    for (int i = 1; i < Number_offspring; i++) {
        if (organism.fitness(individual.get(i), links, scores) < leastfitness) {
            lorgs = individual.get(i);
            leastfitness = organism.fitness(individual.get(i), links, scores);
        }
    }
    System.out.println(generation + " Leastfitness: " + leastfitness + " : " + lorgs);
    MIN.put(lorgs, leastfitness);

    // To find average fitness.

```

```

    for (int i = 0; i < Number_offspring; i++) {
        temp += (organism.fitness(individual.get(i), links, scores));

    }
    AVG.add(temp / Number_offspring);
    System.out.println("AVERAGE " + temp / Number_offspring);

}

//To find the bestfitness out of all generation
double maxValueInMap = (Collections.max(MAXI.values()));
for (Entry<Integer, Double> entry : MAXI.entrySet()) { // Itrate through hashmap
    if (entry.getValue() == maxValueInMap) {
        System.out.println("fBest" + generation + ":" + entry.getKey() + ":" + entry.getValue());
    }
}

//To find the leastfitness out of all generation
double minValueInMap = (Collections.min(MIN.values()));
for (Entry<Integer, Double> entry : MIN.entrySet()) { // Itrate through hashmap
    if (entry.getValue() == minValueInMap) {
        System.out.println("fLeast" + generation + ":" + entry.getKey() + ":" + entry.getValue());
    }
}

//To find the Averagefitness out of all generation
for (double avg : AVG) {
    avgValue += avg;
}
System.out.println("fAvg" + generation + ":" + (avgValue / generation));
}

//author @ Lazer, D &Friedman, A
//To convert Integer to Binary
public static int[] intToBin(int num) {
    int[] bin = new int[n];
    for (int i = n - 1; i >= 0; i--) { //start with the highest bit
        if (((1 << i) & num) != 0) { //bitshift 1 over and compare with the power of 2

            bin[n - 1 - i] = 1;
            //write things right to left
        } else {
            bin[n - 1 - i] = 0;
        }
    }
}

return bin;
}

//author @ Lazer, D &Friedman, A
//To convert Binary to Integer

public static int binToInt(int[] bin) {
    int t = bin.length; //should be k+1

```

```

    int num = 0;
    int coef = 0;
    for (int i = 0; i < t; i++) {
        coef = (int) Math.pow(2, (t - i - 1));
        num += bin[i] * coef;
    }
    return num;
}
}
}

```

### Block (c):ORGANIMS.JAVA

```

class organism {

    private static int[] id = new int[NK.n];
    private static int[] xd = new int[NK.n];
    static double value;
    double Mutation_rate = 0.7;
    double crossover_rate = 0.02;

    organism() { // creates organism with random value

        int Rnd = randoms();

        id = NK.intToBin(Rnd);

    }

    //Mutation and Crossover with mutation variant individuals are generated
    //a. use the code within comment lines for crossover with mutation variant other it is
    mutation variant
    organism(int[] rd) {
        /*if(crossover_rate<Math.random()) // remove the comment for crossover with mutation
        variant
        {
            for(int i=0;i<2;i++)
            {
                int randId = (int)( Math.random() * (NK.n) );
                rd[randId]=id[randId];

            }
        }*/
        // remove the comment for crossover with mutation variant

        //use only for mutation variant
        if (Mutation_rate < Math.random()) {
            int randId = (int) (Math.random() * (NK.n));
            if ((rd[randId]) != 1) {
                rd[randId] = 1;
            }
        }
    }
}

```



```

        } else {
            rd[randId] = 0;
        }
        id = rd;
    } else {
        id = rd;
    }

}

}

//returns genome
public int[] getGenome() {
    return id;
}

//returns a random integer
public static int randoms() {
    return ((int) Math.floor((Math.random() * NK.land_scape.length)));
}

//Calculates the fitness
//Param :int,int array,double array
//Param:returns double
public static double fitness(int BF, int[][] link, double[][] sxore) {

    int[] fit = new int[NK.k + 1];

    xd = NK.intToBin(BF);
    //System.out.println(Arrays.toString(id)+"I");

    double temp_score = 0;
    for (int g = 0; g < NK.n; g++) {
        for (int m = 0; m <= NK.k; m++) {
            fit[m] = xd[link[g][m]];

        }
        temp_score += sxore[g][NK.binToInt(fit)];
        value = temp_score / NK.n;
    }

    return value;
}
}

```

## BIBLIOGRAPHY

- Aguirre, H. & Tanaka, K., 2003. A study on the behavior of genetic algorithms on NK-landscapes: Effects of selection, drift, mutation, and recombination. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 86(9), pp.2270–2279. Available at: [http://search.ieice.org/bin/summary.php?id=e86-a\\_9\\_2270](http://search.ieice.org/bin/summary.php?id=e86-a_9_2270) [Accessed July 4, 2013].
- Kauffman, S A & Weinberger, E.D., 1989. The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2), pp.211–245. Available at: <http://www.ncbi.nlm.nih.gov/pubmed/2632988> [Accessed July 27, 2013].
- Kauffman, S A, 1993. *The Origins of Order: Self Organization and Selection in Evolution*, Oxford University Press, NewYork.
- Kauffman, A & Levin, S., 1987. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1), pp.11–45. Available at: <http://www.sciencedirect.com/science/article/pii/S0022519387800292> [Accessed July 13, 2013].
- Merz, P. & Freisleben, B., 1998. On the effectiveness of evolutionary search in high-dimensional NK-landscapes. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*. IEEE, pp. 741–745. Available at: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=700144> [Accessed July 4, 2013].
- Pelikan, M., Sastry, K. & Goldberg, D., 2009. Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 851-858. Available at: <http://dl.acm.org/citation.cfm?id=1570018> [Accessed July 6, 2013].
- Vidgen, R., Padget, J. & Mitchell, J., 2008. An extended, agent-based implementation of Kauffman's NK (C) model. *University of Bath*. Available at: [https://wiki.bath.ac.uk/download/attachments/15500198/NKC\\_Bath\\_overview.doc](https://wiki.bath.ac.uk/download/attachments/15500198/NKC_Bath_overview.doc) [Accessed July 5, 2013].

## Parts of code adapted from

Lazer, D & Friedman, A (2007) The Network Structure of Exploration and Exploitation, *Administrative Science Quarterly*, 52(4) [computer program] NK\_gen. Available at: <https://code.google.com/p/parallelproblemsolving/>