Optimization of problem using Binary Coded Genetic Algorithm (BCGA)

ME674 Coding Assignment-2 Report

Babalesh Kumar

Roll No: 214103007

Department of Mechanical Engineering,

Indian Institute of Technology Guwahati,

Assam, India-781039

1. Question:

Use a binary-coded GA to minimize the function f $(X_1, X_2) = X_1 + X_2 - 2X_1^2 - X_2^2 + X_1X_2$, in the range of $0.0 \le X_1, X_2 \le 0.5$. using a random population of size N=6, a single point crossover with probability p_C = 1.0, and assume 5 bits for each variable.

2. Procedure for Binary Coded Genetic Algorithm

a. A population of size N=6 is randomly initialized containing 6 strings of length 10 bits.5 bits for each variable
b. Then decoded values s1 and s2 is then calculated.
c. Then real values x1 and x2 are calculated and fitness values are also calculated using following formula.

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{\ell_i} - 1} DV(\mathbf{s}^i),$$

d. Then the minimization problem is converted into maximization problem using fitness function of F(X) = -f(x)
e. Then reproduction is carried out through tournament selection of size 3 and winners are obtained from reproduction.
f. Then for Crossover, Single Point Crossover method is carried out with probability of p_C = 1.0.
g. Then Mutation is carried out over the population with mutation probability of pm =0.05.
h. Then this process is carried out for 1000 generations.

3. Results

After 1000 number of generations the population obtained is as follows:

1. 0 0 0 1 1 0 0 0 0 0
2. 0 0 0 1 0 0 0 0 0 0
3. 0 0 0 1 1 0 0 0 0 1
4. 0 1 0 0 1 0 0 0 0 0
5. 0 0 0 1 1 0 0 0 0 1
6. 0 0 0 1 1 0 0 0 0 1

- The final x1, x2 and fitness values are as follows:

Sr. No.	X1	X2	Fitness Value
1	0.016129	0.016129	-0.031738
2	0.000000	0.000000	-0.000000
3	0.016129	0.016129	-0.031738
4	0.016129	0.000000	-0.015609
5	0.016129	0.016129	-0.031738
6	0.016129	0.016129	-0.031738

- Final Solution is:
  X1 = 0.00000
  X2 = 0.00000

C Programming Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

float fun(float x1, float x2)
{
	float F;
	F = (x1) + (x2) - (2*pow(x1,2)) - (pow(x2,2)) + (x1*x2);
	return -F;
}

int main()
{
	float fun(float , float);
	int i,j,S[20][20],r1,r2,mp[20][20],ch[15][15]; /*r1,r2 are random numbers for creating mating pool*/
	int a,b,c,d,e,f,co1, co2,co3,rn,count = 1;	/*mp - mating pool, ch- childern solution*/
	float c1[10],c2[10],pb = 0.05, pm[15][15],s1[10],s2[10];/*c1,c2-decoded values of solution*/
	float x1min = 0, x1max = 0.5, x2min = 0, x2max = 0.5,x1[10],x2[10],fi[10]; /*s1,s2- */
	srand((unsigned)time(NULL));

	/*generating random solution set*/
	for(i=1;i<=6;i++)
	{
		for(j=1;j<=10;j++)
		{
			S[i][j] = rand() % 2;
			printf("%d ",S[i][j]);
		}
		printf("\n");
	}
	printf("\n");

	while (count<1000)
	{
		/*decoding the values of x1 and x2*/
	for(i=1;i<=6;i++)
	{
	    c1[i] = 0;
	    c2[i] = 0;
		for(j=1;j<=10;j++)
		{
		    if(j<=5)
		    {
			c1[i] = c1[i] + (pow(2, (5-j))*S[i][j]);
		    }
		    else
```

```
            {
                c2[i] = c2[i] + (pow(2, (10-j))*S[i][j]);
            }
        }
}
/*actual values of x1 and x2*/
for(i=1;i<=6;i++)
{
     x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*c1[i]);
     x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*c2[i]);
}
/*calculating fitness value of each solution*/
for(i=1;i<=6;i++)
{
    fi[i] = fun(x1[i],x2[i]);
    printf("\n%f",fi[i]);
}
/*tounament selection for mating pool*/
for(i=1;i<=6;i++)
{
    r1 = (rand() %(6 - 1 + 1)) + 1;
    r2 = (rand() %(6 - 1 + 1)) + 1;
    printf(" \nfor r-%d  are r1-%d, r2-%d and the winner is ",i,r1,r2);
    if(fi[r1]>fi[r2])
    {
        for(j=1;j<=10;j++)
        {
            mp[i][j] = S[r1][j];
        }
        printf("%d",r1);
    }
    else
    {
        for(j=1;j<=10;j++)
        {
            mp[i][j] = S[r2][j];
        }
        printf("%d",r2);
    }
}

/*mating pool*/
printf("\nMating pool as follows\n");
for(i=1;i<=6;i++)
{
      for(j=1;j<=10;j++)
      {
            printf("%d ",mp[i][j]);
      }
      printf("\n");
}
/*single point crossover*/
a = (rand() %(6 - 1 + 1)) + 1;
      b = (rand() %(6 - 1 + 1)) + 1;
```

```
        while(b==a)
        {
              b=(rand() %(6 - 1 + 1)) + 1;
        }
        c = (rand() %(6 - 1 + 1)) + 1;
        while(c==a || c==b)
        {
              c=(rand() %(6 - 1 + 1)) + 1;
        }
        d = (rand() %(6 - 1 + 1)) + 1;
        while(d==a || d==b || d==c)
        {
              d=(rand() %(6 - 1 + 1)) + 1;
        }
  e = (rand() %(6 - 1 + 1)) + 1;
  while(e==a || e==b || e==c || e==d)
        {
              e=(rand() %(6 - 1 + 1)) + 1;
        }
  f = (rand() %(6 - 1 + 1)) + 1;
  while(f==a || f==b || f==c || f==d || f==e)
        {
              f=(rand() %(6 - 1 + 1)) + 1;
        }
        printf("a=%d, b=%d, c= %d, d= %d, e= %d and f=%d\n",a,b,c,d,e,f);

  printf("\nPairs are a,b\n");
    co1 = (rand() %(9 - 1 + 1)) + 1;
    printf("Crossover at co1 = %d\n",co1);
    for(j=1;j<=co1;j++)
    {
        ch[a][j] = mp[a][j];
        ch[b][j] = mp[b][j];
    }
    for(j=co1+1;j<=10;j++)
    {
        ch[a][j] = mp[b][j];
        ch[b][j] = mp[a][j];
    }
    printf("\nPairs are c,d\n");
    co2 = (rand() %(9 - 1 + 1)) + 1;
  printf("Crossover at co2 = %d \n ",co2);
  for(j=1;j<=co2;j++)
  {
        ch[c][j] = mp[c][j];
        ch[d][j] = mp[d][j];
        }
        for(j=co2+1;j<=10;j++)
    {
        ch[c][j] = mp[d][j];
        ch[d][j] = mp[c][j];
        }
        printf("\nPairs are e,f\n");
    co3 = (rand() %(9 - 1 + 1)) + 1;
```

```
printf("Crossover at co3 = %d\n",co3);
for(j=1;j<=co3;j++)
{
      ch[e][j] = mp[e][j];
      ch[f][j] = mp[f][j];
      }
      for(j=co3+1;j<=10;j++)
      {
         ch[e][j] = mp[f][j];
         ch[f][j] = mp[e][j];
  }

  printf("childern solution\n\n");

  for(i=1;i<=6;i++)
{
      for(j=1;j<=10;j++)
      {
            printf("%d ",ch[i][j]);
      }
      printf("\n");
}

/*generating propbability for each solution*/
printf("\n");
for(i=1;i<=6;i++)
{
      for(j=1;j<=10;j++)
      {
            rn = (rand() %(100 - 0 + 1)) + 0;
            pm[i][j] = (float)rn/100;
            printf("%f ",pm[i][j]);
            if(pm[i][j]<=0.05)
            {
                if(ch[i][j] == 0)
                {
                    ch[i][j] = 1;
                }
                else
                {
                    ch[i][j] = 0;
                }
            }
      }

      printf("\n");
}
/*making child values as solution pool for next iteration*/
for(i=1;i<=6;i++)
{
      for(j=1;j<=10;j++)
      {
            S[i][j] = ch[i][j];
      }
```

```
}
```

```
        count++;
        printf("count = %d\n",count);
        }
        /*Decoding the values of x1 and x2*/
        for(i=1;i<=6;i++)
        {
            c1[i] = 0;
            c2[i] = 0;
              for(j=1;j<=10;j++)
              {
                  if(j<=5)
                  {
                      c1[i] = s1[i] + (pow(2, (5-j))*S[i][j]);
                  }
                  else
                  {
                      c2[i] = s2[i] + (pow(2, (10-j))*S[i][j]);
                  }
              }


              printf("%f  %f\n",c1[i],c2[i]);
        }
        /*actual x1 and x2 values from decoded values within the range*/
        for(i=1;i<=6;i++)
        {
            x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*c1[i]);
            x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*c2[i]);
            printf("%f %f\n",x1[i],x2[i]);
        }

        /*fitness values for each solution*/
        for(i=1;i<=6;i++)
        {
            fi[i] = fun(x1[i],x2[i]);
            printf("\n%f",fi[i]);
        }
        return 0;
}
```