Optimization of problem using Binary Coded Genetic Algorithm (BCGA)

ME674 Coding Assignment-2 Report

Chetan Shantaram Nalavade

Roll No : 214103009

Department of Mechanical Engineering,

Indian Institute of Technology Guwahati,

Assam, India-781039

1. Problem Statement

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 therange of $0.0 \leq X_1, X_2 \leq 0.5$. 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 fitnessfunction of F(X) = -f(x)
e. Then reproduction is carried out through tournament selection of size 3 and winnersare obtained from reproduction.
f. Then for Crossover, Elitism method is used in which best 2 solutions are kept sameand Single Point Crossover method is carried out for 4 remaining solutions 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 10000 generations.

3. Results

After 10000 number of generations the population obtained is as follows:

1) 1 1 1 1 1 0 0 0 0 0
2) 1 1 1 1 0 1 0 0 1 0
3) 1 1 0 1 0 0 0 0 0 0
4) 1 1 1 1 0 1 0 0 0 0
5) 1 1 1 1 0 0 0 0 0 0
6) 1 1 1 0 1 0 0 1 0 0

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

Sr. No.	X1	X2	Fitness Value
1	0.50000	0.00000	0.00000
2	0.48387	0.29032	-0.36212
3	0.41936	0.00000	-0.06764
4	0.48387	0.25807	-0.33195
5	0.48387	0.00000	-0.01561
6	0.46774	0.06452	-0.12071

- Final Solution is:
  X1 = 0.50000
  X2 = 0.00000

- Code for this problem is mentioned from next page.

```c
// Defining library
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

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

int main()
{
    //Initialization
    double fun(double , double);
    int S[7][11],i,j;
    int Gen = 1;
    srand((unsigned)time(NULL));

    //Generating Random Number
    for(i=1;i<=6;i++)
    {
        for(j=1;j<=10;j++)
        {
            S[i][j] = rand() % 2;
            //printf("%d ",S[i][j]);
        }
        //printf("\n");
    }


    do
    {
        //Decoding the values of x1 and x2
        double D1[7],D2[7];
        for(i=1;i<=6;i++)
        {
            D1[i] = 0;
            D2[i] = 0;
            for(j=1;j<=10;j++)
            {
                if(j<=5)
                {
                    D1[i] = D1[i] + (pow(2, (5-j))*S[i][j]);
                }
                else
                {
                    D2[i] = D2[i] + (pow(2, (10-j))*S[i][j]);
                }
            }

            //printf("%lf  %lf\n",D1[i],D2[i]);
```

```
56:         }
57:
58:         //Finding the actual x1 and x2 values from decoded values within the range
59:         double x1min = 0.0, x1max = 0.5, x2min = 0, x2max = 0.5,x1[7],x2[7];
60:         for(i=1;i<=6;i++)
61:         {
62:             x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*D1[i]);
63:             x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*D2[i]);
64:             //printf("%lf  %lf\n",x1[i],x2[i]);
65:         }
66:
67:         //Finding the fitness values for each solution
68:         double f[7];
69:         for(i=1;i<=6;i++)
70:         {
71:             f[i] = fun(x1[i],x2[i]);
72:             //printf("\n%lf",f[i]);
73:         }
74:
75:         //Using tournament selection to choose mating pool
76:         int rn1,rn2,rn3;
77:         int MS[7][11];
78:         double max = f[1];
79:
80:         //elitism operation for best solution
81:         for(i=1;i<=6;i++)
82:         {
83:             if(f[i]>max)
84:             {
85:                 max = f[i];
86:             }
87:         }
88:
89:         //printf("\nMax = %lf and No is ",max);
90:
91:         for(i=1;i<=6;i++)
92:         {
93:             if(f[i] == max)
94:             {
95:                for(j=1;j<=10;j++)
96:                {
97:                    MS[1][j] = S[i][j];
98:                }
99:                //printf("%d\n",i);
100:            }
101:        }
102:
103:        for(i=2;i<=6;i++)
104:        {
105:            rn1 = (rand() %(6 - 3 + 1)) + 3;
106:            rn2 = (rand() %(6 - 3 + 1)) + 3;
107:            rn3 = (rand() %(6 - 3 + 1)) + 3;
108:            //printf(" \nfor %d rn are %d %d %d and the winner is ",i,rn1,rn2,rn3);
109:            if(f[rn1]>f[rn2])
110:            {
```

```
111:                 if(f[rn1]>f[rn3])
112:                 {
113:                     for(j=1;j<=10;j++)
114:                     {
115:                         MS[i][j] = S[rn1][j];
116:                     }
117:                     //printf("%d",rn1);
118:                 }
119:                 else
120:                 {
121:                     for(j=1;j<=10;j++)
122:                     {
123:                         MS[i][j] = S[rn3][j];
124:                     }
125:                     //printf("%d",rn3);
126:                 }
127:             }
128:             else
129:             {
130:                 if(f[rn2]>f[rn3])
131:                 {
132:                     for(j=1;j<=10;j++)
133:                     {
134:                         MS[i][j] = S[rn2][j];
135:                     }
136:                     //printf("%d",rn2);
137:                 }
138:                 else
139:                 {
140:                     for(j=1;j<=10;j++)
141:                     {
142:                         MS[i][j] = S[rn3][j];
143:                     }
144:                     //printf("%d",rn3);
145:                 }
146:             }
147:         }
148:
149:         //printing mating pool
150:         //printf("\nMating pool as follows\n");
151:         for(i=1;i<=6;i++)
152:         {
153:             for(j=1;j<=10;j++)
154:             {
155:                 //printf("%d ",MS[i][j]);
156:             }
157:             //printf("\n");
158:         }
159:
160:         //Now Doing single point crossover
161:         //finding crossover point and doing
162:         int CH[7][11], co1, co2, RNC;
163:         RNC = (rand() %(3 - 1 + 1)) + 1;
164:
165:         for(j=1;j<=10;j++)
```

```
166:             {
167:                 CH[1][j] = MS[1][j];
168:                 CH[2][j] = MS[2][j];
169:             }
170: 
171:         if(RNC == 1)
172:         {
173:             //printf("\nPairs are 3,4 and 5,6\n");
174:             co1 = (rand() %(9 - 1 + 1)) + 1;
175:             //printf("\nCrossover\nco1 = %d and ",co1);
176:             for(j=1;j<=co1;j++)
177:             {
178:                 CH[3][j] = MS[3][j];
179:                 CH[4][j] = MS[4][j];
180:             }
181:             for(j=co1+1;j<=10;j++)
182:             {
183:                 CH[3][j] = MS[4][j];
184:                 CH[4][j] = MS[3][j];
185:             }
186: 
187:             co2 = (rand() %(9 - 1 + 1)) + 1;
188:             //printf("co2 = %d\n",co2);
189:             for(j=1;j<=co2;j++)
190:             {
191:                 CH[5][j] = MS[5][j];
192:                 CH[6][j] = MS[6][j];
193:             }
194:             for(j=co2+1;j<=10;j++)
195:             {
196:                 CH[5][j] = MS[6][j];
197:                 CH[6][j] = MS[5][j];
198:             }
199:         }
200: 
201:         if(RNC == 2)
202:         {
203:             //printf("\nPairs are 3,5 and 4,6\n");
204:             co1 = (rand() %(9 - 1 + 1)) + 1;
205:             //printf("\nCrossover\nco1 = %d and ",co1);
206:             for(j=1;j<=co1;j++)
207:             {
208:                 CH[3][j] = MS[3][j];
209:                 CH[5][j] = MS[5][j];
210:             }
211:             for(j=co1+1;j<=10;j++)
212:             {
213:                 CH[3][j] = MS[5][j];
214:                 CH[5][j] = MS[3][j];
215:             }
216: 
217:             co2 = (rand() %(9 - 1 + 1)) + 1;
218:             //printf("co2 = %d\n",co2);
219:             for(j=1;j<=co2;j++)
220:             {
```

```
221:                 CH[4][j] = MS[4][j];
222:                 CH[6][j] = MS[6][j];
223:             }
224:             for(j=co2+1;j<=10;j++)
225:             {
226:                 CH[4][j] = MS[6][j];
227:                 CH[6][j] = MS[4][j];
228:             }
229:         }
230:
231:         if(RNC == 3)
232:         {
233:             //printf("\nPairs are 3,6 and 4,5\n");
234:             co1 = (rand() %(9 - 1 + 1)) + 1;
235:             //printf("\nCrossover\nco1 = %d and ",co1);
236:             for(j=1;j<=co1;j++)
237:             {
238:                 CH[3][j] = MS[3][j];
239:                 CH[6][j] = MS[6][j];
240:             }
241:             for(j=co1+1;j<=10;j++)
242:             {
243:                 CH[3][j] = MS[6][j];
244:                 CH[6][j] = MS[3][j];
245:             }
246:
247:             co2 = (rand() %(9 - 1 + 1)) + 1;
248:             //printf("co2 = %d\n",co2);
249:             for(j=1;j<=co2;j++)
250:             {
251:                 CH[4][j] = MS[4][j];
252:                 CH[5][j] = MS[5][j];
253:             }
254:             for(j=co2+1;j<=10;j++)
255:             {
256:                 CH[4][j] = MS[5][j];
257:                 CH[5][j] = MS[4][j];
258:             }
259:         }
260:
261:         for(i=1;i<=6;i++)
262:         {
263:             for(j=1;j<=10;j++)
264:             {
265:                 //printf("%d ",CH[i][j]);
266:             }
267:             //printf("\n");
268:         }
269:
270:         //Now mutation for each bit considering Pm = 0.05
271:
272:         //generating random probability for each bit of each solution
273:         double Pm = 0.05, Pb[7][11];
274:         int RNm;
275:         //printf("\n");
```

```
276:         for(i=2;i<=6;i++)
277:         {
278:             for(j=1;j<=10;j++)
279:             {
280:                 RNm = (rand() %(100 - 0 + 1)) + 0;
281:                 Pb[i][j] = (double)RNm/100;
282:                 //printf("%lf ",Pb[i][j]);
283:                 if(Pb[i][j]<=0.05)
284:                 {
285:                     if(CH[i][j] == 0)
286:                     {
287:                         CH[i][j] = 1;
288:                     }
289:                     else
290:                     {
291:                         CH[i][j] = 0;
292:                     }
293:                 }
294:             }
295:             //printf("\n");
296:         }
297: 
298:         //printf("\nAfter mutation\n");
299:         for(i=1;i<=6;i++)
300:         {
301:             for(j=1;j<=10;j++)
302:             {
303:                 //printf("%d ",CH[i][j]);
304:             }
305:             //printf("\n");
306:         }
307: 
308:         // Restoring the child values back to S pool for next iteration
309:         for(i=1;i<=6;i++)
310:         {
311:             for(j=1;j<=10;j++)
312:             {
313:                 S[i][j] = CH[i][j];
314:             }
315:         }
316: 
317:         Gen = Gen + 1;
318:         //printf("\nGen = %d\n",Gen);
319:     }while(Gen<=10000);
320: 
321:     printf("Final Values\n");
322:     //Decoding the values of x1 and x2
323: 
324:     for(i=1;i<=6;i++)
325:     {
326:         for(j=1;j<=10;j++)
327:         {
328:             printf("%d ",S[i][j]);
329:         }
330:         printf("\n");
```

```
331:     }
332:
333:     double D1[7],D2[7];
334:     for(i=1;i<=6;i++)
335:     {
336:         D1[i] = 0;
337:         D2[i] = 0;
338:         for(j=1;j<=10;j++)
339:         {
340:             if(j<=5)
341:             {
342:                 D1[i] = D1[i] + (pow(2, (5-j))*S[i][j]);
343:             }
344:             else
345:             {
346:                 D2[i] = D2[i] + (pow(2, (10-j))*S[i][j]);
347:             }
348:         }
349:
350:         //printf("%lf  %lf\n",D1[i],D2[i]);
351:     }
352:
353:     //Finding the actual x1 and x2 values from decoded values within the range
354:     double x1min = 0.0, x1max = 0.5, x2min = 0, x2max = 0.5,x1[7],x2[7];
355:     for(i=1;i<=6;i++)
356:     {
357:         x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*D1[i]);
358:         x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*D2[i]);
359:         printf("X1 = %lf and X2 = %lf\n",x1[i],x2[i]);
360:     }
361:
362:     //Finding the fitness values for each solution
363:     double f[7];
364:     printf("\nRespective fitness values\n");
365:     for(i=1;i<=6;i++)
366:     {
367:         f[i] = fun(x1[i],x2[i]);
368:         printf("F1 = %lf\n",f[i]);
369:     }
370:
371:     //Printing the final values
372:     double max = f[1];
373:     for(i=1;i<=6;i++)
374:     {
375:         if(f[i]>max)
376:         {
377:             max = f[i];
378:         }
379:     }
380:
381:     printf("\n\n Final Soultion is as follows\n");
382:     for(i=1;i<=6;i++)
383:     {
384:         if(f[i] == max)
385:         {
```

```
386:             printf("X1 = %lf , X2 = %lf\n",x1[i],x2[i]);
387:         }
388:     }
389:
390:
391:     return 0;
392: }
```