

Optimization of problem using Binary Coded Genetic Algorithm (BCGA)

ME674 Coding Assignment-2 Report

K. Sai Vishnu Vardhan Reddy

Roll No : 214103010



**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 the range 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 population of size $N=6$ is randomly initialized containing 6 strings of length 10 bits. 5 bits for each variable
- Then decoded values s_1 and s_2 is then calculated.
- Then real values x_1 and x_2 are calculated and fitness values are also calculated using following formula.

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

- Then the minimization problem is converted into maximization problem using fitness function of $F(X) = -f(x)$
- Then reproduction is carried out through tournament selection of size 3 and winners are obtained from reproduction.
- Then for Crossover, Elitism method is used in which best 2 solutions are kept same and Single Point Crossover method is carried out for 4 remaining solutions with probability of $p_c = 1.0$.
- Then Mutation is carried out over the population with mutation probability of $p_m = 0.05$.
- Then this process is carried out for 10000 generations.

3. Results

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

- 0 0 0 0 0 0 0 0 0 0
- 0 1 0 0 1 0 0 0 1 0
- 0 0 0 1 1 0 0 0 1 0
- 1 0 0 1 1 0 0 1 1 0
- 0 0 0 0 1 0 0 0 1 0
- 0 1 0 1 0 0 0 0 1 0

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

Sr. No.	X1	X2	Fitness Value
1	0.00000	0.00000	0.00000
2	0.14516	0.03226	-0.13892
3	0.04839	0.03226	-0.07648
4	0.30645	0.09677	-0.23569
5	0.01613	0.03226	-0.04735
6	0.16129	0.03226	-0.14568

- Final Solution is:
X1 = 0.00000
X2 = 0.00000

```

1: #include<stdio.h>
2: #include<stdlib.h>
3: #include<conio.h>
4: #include<math.h>
5: #include<time.h>
6:
7: double FN(double x1, double x2)
8: {
9:     double F;
10:    F = x1 + x2 - (2*x1*x1) - (x2*x2) + (x1*x2);
11:    return -F;
12: }
13:
14: int main()
15: {
16:     //Initialization
17:     double FN(double , double);
18:     int S[7][11],i,j;
19:     int Gen = 1;
20:     srand((unsigned)time(NULL));
21:
22:     //Generating some random Number
23:     for(i=1;i<=6;i++)
24:     {
25:         for(j=1;j<=10;j++)
26:         {
27:             S[i][j] = rand() % 2;
28:
29:         }
30:
31:     }
32:
33:     //Decoding the values
34:     do
35:     {
36:
37:         double D1[7],D2[7];
38:         for(i=1;i<=6;i++)
39:         {
40:             D1[i] = 0;
41:             D2[i] = 0;
42:             for(j=1;j<=10;j++)
43:             {
44:                 if(j<=5)
45:                 {
46:                     D1[i] = D1[i] + (pow(2, (5-j))*S[i][j]);
47:                 }
48:                 else
49:                 {
50:                     D2[i] = D2[i] + (pow(2, (10-j))*S[i][j]);
51:                 }
52:             }
53:
54:
55:         }

```

```

56:
57: //Calculating x1,x2
58: double x1min = 0.0, x1max = 0.5, x2min = 0, x2max = 0.5,x1[7],x2[7];
59: for(i=1;i<=6;i++)
60: {
61:     x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*D1[i]);
62:     x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*D2[i]);
63:
64: }
65:
66: //Finding the fitness values for each solution
67: double f[7];
68: for(i=1;i<=6;i++)
69: {
70:     f[i] = FN(x1[i],x2[i]);
71:
72: }
73:
74: //Using tournament selection to choose mating pool
75: int rn1,rn2,rn3;
76: int MS[7][11];
77: double max = f[1];
78:
79: //Directly taking a best solution
80: for(i=1;i<=6;i++)
81: {
82:     if(f[i]>max)
83:     {
84:         max = f[i];
85:     }
86: }
87:
88:
89:
90: for(i=1;i<=6;i++)
91: {
92:     if(f[i] == max)
93:     {
94:         for(j=1;j<=10;j++)
95:         {
96:             MS[1][j] = S[i][j];
97:         }
98:     }
99: }
100:
101:
102: for(i=2;i<=6;i++)
103: {
104:     rn1 = (rand() %(6 - 3 + 1)) + 3;
105:     rn2 = (rand() %(6 - 3 + 1)) + 3;
106:     rn3 = (rand() %(6 - 3 + 1)) + 3;
107:     if(f[rn1]>f[rn2])
108:     {
109:         if(f[rn1]>f[rn3])
110:         {

```

```

111:         for(j=1;j<=10;j++)
112:         {
113:             MS[i][j] = S[rn1][j];
114:         }
115:     }
116:     }
117:     else
118:     {
119:         for(j=1;j<=10;j++)
120:         {
121:             MS[i][j] = S[rn3][j];
122:         }
123:     }
124: }
125: }
126: else
127: {
128:     if(f[rn2]>f[rn3])
129:     {
130:         for(j=1;j<=10;j++)
131:         {
132:             MS[i][j] = S[rn2][j];
133:         }
134:     }
135:     }
136:     else
137:     {
138:         for(j=1;j<=10;j++)
139:         {
140:             MS[i][j] = S[rn3][j];
141:         }
142:     }
143: }
144: }
145: }
146:
147:
148: for(i=1;i<=6;i++)
149: {
150:     for(j=1;j<=10;j++)
151:     {
152:
153:     }
154: }
155:
156:
157: //Single point crossover
158: int CH[7][11], co1, co2, RNC;
159: RNC = (rand() % (3 - 1 + 1)) + 1;
160:
161: for(j=1;j<=10;j++)
162: {
163:     CH[1][j] = MS[1][j];
164:     CH[2][j] = MS[2][j];
165: }

```

```

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

```

```

221:     if(RNC == 3)
222:     {
223:         co1 = (rand() %(9 - 1 + 1)) + 1;
224:         for(j=1;j<=co1;j++)
225:         {
226:             CH[3][j] = MS[3][j];
227:             CH[6][j] = MS[6][j];
228:         }
229:         for(j=co1+1;j<=10;j++)
230:         {
231:             CH[3][j] = MS[6][j];
232:             CH[6][j] = MS[3][j];
233:         }
234:
235:         co2 = (rand() %(9 - 1 + 1)) + 1;
236:         for(j=1;j<=co2;j++)
237:         {
238:             CH[4][j] = MS[4][j];
239:             CH[5][j] = MS[5][j];
240:         }
241:         for(j=co2+1;j<=10;j++)
242:         {
243:             CH[4][j] = MS[5][j];
244:             CH[5][j] = MS[4][j];
245:         }
246:     }
247:
248:
249:
250:     //Now mutation considering Pm = 0.05
251:     double Pm = 0.05, Pb[7][11];
252:     int RNm;
253:     for(i=2;i<=6;i++)
254:     {
255:         for(j=1;j<=10;j++)
256:         {
257:             RNm = (rand() %(100 - 0 + 1)) + 0;
258:             Pb[i][j] = (double)RNm/100;
259:             if(Pb[i][j]<=0.05)
260:             {
261:                 if(CH[i][j] == 0)
262:                 {
263:                     CH[i][j] = 1;
264:                 }
265:                 else
266:                 {
267:                     CH[i][j] = 0;
268:                 }
269:             }
270:         }
271:     }
272: }
273:
274:
275:

```



```

276:
277:     // Restoring the child values back to S pool for next iteration
278:     for(i=1;i<=6;i++)
279:     {
280:         for(j=1;j<=10;j++)
281:         {
282:             S[i][j] = CH[i][j];
283:         }
284:     }
285:
286:     Gen = Gen + 1;
287: }while(Gen<=10000);
288:
289: printf("Final Values\n");
290: //Decoding the values of x1 and x2
291:
292: for(i=1;i<=6;i++)
293: {
294:     for(j=1;j<=10;j++)
295:     {
296:         printf("%d ",S[i][j]);
297:     }
298:     printf("\n");
299: }
300:
301: double D1[7],D2[7];
302: for(i=1;i<=6;i++)
303: {
304:     D1[i] = 0;
305:     D2[i] = 0;
306:     for(j=1;j<=10;j++)
307:     {
308:         if(j<=5)
309:         {
310:             D1[i] = D1[i] + (pow(2, (5-j))*S[i][j]);
311:         }
312:         else
313:         {
314:             D2[i] = D2[i] + (pow(2, (10-j))*S[i][j]);
315:         }
316:     }
317: }
318:
319:
320: //Finding the actual x1 and x2
321: double x1min = 0.0, x1max = 0.5, x2min = 0, x2max = 0.5,x1[7],x2[7];
322: for(i=1;i<=6;i++)
323: {
324:     x1[i] = x1min + (((x1max - x1min)/((pow(2, 5))-1))*D1[i]);
325:     x2[i] = x2min + (((x2max - x2min)/((pow(2, 5))-1))*D2[i]);
326:     printf("X1 = %1f and X2 = %1f\n",x1[i],x2[i]);
327: }
328:
329: //Finding the fitness values for each solution
330: double f[7];

```

```

331: printf("\nRespective fitness values\n");
332: for(i=1;i<=6;i++)
333: {
334:     f[i] = FN(x1[i],x2[i]);
335:     printf("F1 = %lf\n",f[i]);
336: }
337:
338: //Printing the final values
339: double max = f[1];
340: for(i=1;i<=6;i++)
341: {
342:     if(f[i]>max)
343:     {
344:         max = f[i];
345:     }
346: }
347:
348: printf("\n\n Final Soution is as follows\n");
349: for(i=1;i<=6;i++)
350: {
351:     if(f[i] == max)
352:     {
353:         printf("X1 = %lf , X2 = %lf\n",x1[i],x2[i]);
354:     }
355: }
356:
357:
358: return 0;
359: }

```