Project 1 - Genetic Algorithm

Tao Zhang
CS 472

February 20, 2014

Abstract

My Genetic Algorithm works well for most functinos, except Rastrigin and Schwefel function. I tested all the crossover methods, except "Permutation". Among the crossover methods, I find that 1 point crossover works the best for most of the functions.

1 Description for Genetic Algorithm

Generally, my Genetic Algorithm used Steady-State type that:

- generate a population
- have enough loops
- for each loop
  1. tournament select two parents
  2. crossover (mainly by 1 point crossover)
  3. mutate 2 children
  4. replace two loser (tournament selection) by two children
- finally check the fitness of each individual and find the best

1.1 Fitness function: just math, no need to explain

- Sphere:

```
double evaluate(double x[X_I])
{
        double z = 0;
        double tmp;
        int i;
        for(i = 0 ; i < X_I; i++ )
        {
                tmp = x[i];
                z += tmp*tmp;
        }

        return z;
}
```

- Rosenbrock

```
double evaluate(double x[X_I])
{
        double z = 0;
        int i;
        for(i = 0; i < X_I - 1; i++) {
```

```
                z += 100*pow(x[i+1]-pow(x[i],2) , 2) + pow(x[i] - 1, 2);
        }

        return z;
}
```

- Rastrigin

```
double evaluate(double x[X_I])
{
        double z = 10*X_I;
        int i;
        for(i = 0; i < X_I; i++){
                z += pow(x[i] , 2) - 10*cos(2*M_PI*x[i]);
        }
        return z;
}
```

- Schwefel

```
double evaluate(double x[X_I])
{
        double z = 418.9829*X_I;
        int i;
        for(i = 0; i < X_I; i++){
                z += x[i]*sin(sqrt(abs(x[i])));
        }
        return z;
}
```

- Ackley

```
double evaluate(double x[X_I])
{
        double z = 20+exp(1);
        double a = 0;
        double b = 0;
        int i;
        for(i = 0; i<X_I;i++){
                a += pow(x[i] , 2);
                b += cos(2*M_PI*x[i]);
        }
        z = z - 20*exp(-0.2*sqrt(a/X_I)) - exp(b/X_I);

        return z;
}
```

- Griewangk

```
double evaluate(double x[X_I])
{
        double z = 1;
        double a = 0;
        double b = 1;
```

```
        int i;
        for(i = 0; i < X_I; i++) {
                a += pow(x[i],2) / 4000.00;
                b *= cos(x[i]/(i+1));
        }
        z = z + a - b;

        return z;
}
```

1.2 Generate initial random solutions

```
// Generate a population
for 0 to Population step 1
  // Generate each individual
  for 0 to 30 step 1
    // Take random values in range for each vector
  // Calculate each individuals fitness, put into an array
```

1.3 Crossover

Once we get the parents, we will genearte two children by crossover. I did 4 crossover methods, listed below:

- one point crossover

```
void crossover(double father[X_I],
                double mother[X_I], double children[2][X_I])
{
        int cut = rand()%X_I;   // get a random cut/point
        int i;
        for(i = 0; i < cut; i++){       // keep the front half
                children[0][i] = father[i];
                children[1][i] = mother[i];
        }
        for( ; i < X_I; i++){           // swap/cross the back half
                children[1][i] = father[i];
                children[0][i] = mother[i];
        }
}
```

- two points crossover

```
void crossover(double father[X_I],
                double mother[X_I], double children[2][X_I])
{
        // get two random cuts/points
        int cut1 = rand()%X_I;
        int cut2 = rand()%(X_I - cut1);
        int i;
```

```
        for(i = 0; i < cut1; i++){
                children[0][i] = father[i];
                children[1][i] = mother[i];
        }
        for( ; i < cut1+cut2; i++){     // only swap the middle
                children[1][i] = father[i];
                children[0][i] = mother[i];
        }
        for( ; i < X_I; i++){
                children[0][i] = father[i];
                children[1][i] = mother[i];
        }
}
```

- uniform crossover

```
void crossover(double father[X_I],
                double mother[X_I], double children[2][X_I])
{
        double tmp= 0;
        int P = 0;
        int i;
        for(i = 0; i < X_I; i++) {
                // for each value, we have 30% chance to swap that "gene"
                if((P = rand()%100) < 30) {
                        tmp = children[0][i];
                        children[0][i] = children[1][i];
                        children[1][i] = tmp;
                }
        }
}
```

- arithmetic

```
void crossover(double father[X_I],
                double mother[X_I], double children[2][X_I])
{
        double tmp = 0;
        int i;
        for(i = 0; i < X_I; i++) {
                // get the avg of two values and get two same children
                tmp = (children[0][i] + children[1][i]) / 2;
                children[0][i] = tmp;
                children[1][i] = tmp;
        }
}
```

1.4 Mutation

Mutate the children by "creep". For each vecter in child, we have 50percent chance to mutate the value by +/- a small value (0.001 for Rosenbrock; 0.01 for Sphere, Rastrigin; 0.1 for Ackley; 1 for Schwefel, Griewangk).

```
void mutate(double child[X_I])
{
  for 0 to 30 step by 1
    if rand()%10 < 5  // 50% chance
      // 50% + and 50% -
      child[i] += (rand()%10 < 5 ? -1 : 1)*INTERVAL;
}
```

1.5 GeneticAlgorithem

The major algorithm part. It will finally return the best fitness, also pass the reference of the best individual (solution) so far.

```
        /* Steady-State */
double GeneticAlgorithm(double resultA[X_I])
{
  // Generate a population
  // Calculate each individuals fitness, put into an array

  while(count < population*100) // as Dr. Soul suggested
  {
    // select father by Tournament
    // select mother by Tournament (diff from father)

        // crossover father and mother to get two children

    // Mutate both children

    // select two losers by Tournament
    // Replace two losers by two children
  }

  /* Finally, check the fitness array */
  bestEval <- fitness[0](inital);
  for 0 to population step 1
    if fitness[i] < bestEval
      bestEval <- fitness[i];
      best = i  // record the position;

  // copy best solution to resultA

  return bestEval;
}
```

1.6 Selection

This is a tournament selection function which select a winner/loser (depend on the index) from a sample of N random individuals (N is also a random number). Finally the function returns a int which is the index of that individual among population array.

```
        /* Tournament selection */
int selection(int good_poor, double fitness[POP])
/* good_poor is the index to determe we are select a winner or loser*/
{
  // get a random winner and calc his fitness
  winner <- rand()%POP;
  winner_fitness <- fitness[winner];

  // generate a random N
  N <- rand()%POP ;
  // loop to get the best
  for 0 to N step 1
    temp <- rand%POP
    if(good_poor == GOOD && fitness[temp] < winner_fitness){
    // this is for winner
      winner_fitness <- fitness[temp];
      winner <- temp;
    }else if(good_poor == POOR && fitness[temp] > winner_fitness){
    // this is for loser
      winner_fitness <- fitness[temp];
      winner <- temp;
    }

    return winner; // the index among population
}
```

2 Results

2.1 Sphere

Crossover	1point	2points	uniform	arithmetic
Population	50	50	50	50
Mutation Interval	0.01	0.01	0.01	0.01
Running Times(*POP)	100	100	100	100
1st Best Fitness	187.526800	148.298600	142.681100	148.242200
1st Avg Fitness	258.182810	259.975670	258.766226	265.070316
Final Best Fitness	0.002500	0.001200	2.557200	1.217325
Final Avg Fitness	0.003112	0.001756	5.150842	2.173404

2.2 Rosenbrock

Crossover	1point	2points	uniform	arithmetic
Population	100	100	100	100
Mutation Interval	0.001	0.001	0.001	0.001
Running Times(*POP)	2000	2000	2000	2000
1st Best Fitness	6633.415331	6537.239168	6105.194947	5716.116572
1st Avg Fitness	14865.619920	14462.804456	14392.547199	14755.587222
Final Best Fitness	0.192033	4.180114	159.677838	141.739331
Final Avg Fitness	0.196815	4.183143	209.094239	148.872279

2.3 Rastrigin

My genetic algorithm doesn't work for Rastrigin function. The best result I get will stay (little change) at some point, even I add more running times. But the fitness function I write should be correct. I have no idea about it.

2.4 Schwefel

For this function, I get the same trouble as Rastrigin function.

2.5 Ackley

Crossover	1point	2points	uniform	arithmetic
Population	50	50	50	50
Mutation Interval	1	1	1	1
Running Times(*POP)	1000	1000	1000	1000
1st Best Fitness	18.924080	18.885888	18.775362	18.762705
1st Avg Fitness	19.334254	19.379073	19.375065	19.308400
Final Best Fitness	1.711187	1.841785	2.637531	2.467152
Final Avg Fitness	2.319135	2.409010	14.853659	15.595730

2.6 Griewangk

Crossover	1point	2points	uniform	arithmetic
Population	50	50	50	50
Mutation Interval	1	1	1	1
Running Times(*POP)	100	100	100	100
1st Best Fitness	533.451750	612.249000	594.165750	518.436000
1st Avg Fitness	929.957840	915.618320	906.391795	900.314490
Final Best Fitness	0.277369	2.207000	6.057750	4.453250
Final Avg Fitness	0.391046	2.354080	11.263200	10.335357

3 Conclusion

My Genetic Algorithm should be successful, but still need more researches to figure out the trouble I get on Rastrigin and Schwefel functions.