Scaling in Genetic Algorithms

Sushil J. Louis
Dept. of Computer Science
University of Nevada, Reno
sushil@cse.unr.edu

September 4, 2019

Motivation

We want to maintain an even selection pressure throughout the genetic algorithm's processing.

- At the beginning of the GA run, there may be a very high fitness individual i, that biases search towards i.
- Near the end of a run, when the population is converging, there may also not be much seperation among individuals in the population.

Neither is desirable. Thus we may want to scale the fitness so that selection pressure remains the same throughout the run. Let us formulate the problem in the following way:

- One useful scaling procedure is linear scaling where we want to scale the fitness of each individual in the population such that the scaled fitness is linearly related to the unscaled fitness.
$$f' = af + b$$
- We want to maintain a certain relationship between the maximum fitness individual in the population and the average population fitness. Let this be expressed by the following constraint equations:
$$f'_{max} = f_{avg} * C_s$$
$$f'_{avg} = f_{avg}$$
where f' is the scaled maximum fitness, f_{avg} is the average fitness of the population, and C_s is a scaling constant that specifies the expected number of copies of the best individual in the next generation. Increasing C_s will increase selection "pressure" (bias towards best individual and quicker convergence), decreasing C_s will decrease selection pressure.

We can calculate the linear coefficients a and b from the constraint equations above. However, it is possible for the scaled fitness to go below 0. What do we do? We map f'_{min} to 0. The pascal code for doing this is given in the book on page 79.

```
#include "type.h"

void FindCoeffs(IPTR pop, Population *p);

void Scalepop(IPTR pop, Population *p)
{
  /* linearly scale the population */
  IPTR pj;
  int i;

  FindCoeffs(pop, p);

  p->scaledSumFitness = 0.0;
  for(i = 0; i < p->popsize; i++){
    pj = &pop[i];
    pj->scaledFitness = p->scaleConstA * pj->fitness + p->scaleConstB;
    p->scaledSumFitness += pj->scaledFitness;
  }
}

void FindCoeffs(IPTR pop, Population *p)
{
  /* find coeffs scale_constA and scale_constB for linear scaling according to
     f_scaled = scale_constA * f_raw + scale_constB */

  double d;

  if(p->min > (p->scaleFactor * p->avg - p->max)/
     (p->scaleFactor - 1.0)) { /* if nonnegative smin */
    d = p->max - p->avg;
    p->scaleConstA = (p->scaleFactor - 1.0) * p->avg / d;
    p->scaleConstB = p->avg * (p->max - (p->scaleFactor * p->avg))/d;
  } else {  /* if smin becomes negative on scaling */
    d = p->avg - p->min;
    p->scaleConstA = p->avg/d;
    p->scaleConstB = -p->min * p->avg/d;
  }
  if(d < 0.00001 && d > -0.00001) { /* if converged */
    p->scaleConstA = 1.0;
    p->scaleConstB = 0.0;
  }
}
```