Adaptation in Genetic Algorithms

Ghodrat Moghadampour
Vaasa University of Applied Sciences
Vaasa
Finland

Outline

- Evolutionary Algorithms
- Genetic Algorithms
- Parameters
- Adaptation
- Mutation Operator
- Adaptation for mutation operator
- Intelligent mutation operators

Motivation

- In the real world, there are numerous hard problems, which cannot be solved with conventional techniques within reasonable time, like optimization problems:

$$\frac{1}{4000}\sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1$$

- Conventional techniques require rigid assumptions, like convexity, linearity, differentiability, explicitly defined objectives and so on.

Evolutionary Algorithms

- It is generally accepted that any evolutionary algorithm must have five basic components:
  1. a genetic representation of a number of solutions to the problem
  2. a way to create an initial population of solutions
  3. an evaluation function for rating solutions in terms of their “fitness”
  4. “genetic” operators that alter the genetic composition of offspring during reproduction
  5. values for the parameters, e.g. population size, probabilities of applying genetic operators

Purpose of Evolutionary Algorithms

- Classical optimization problems are more efficient in solving linear, quadratic, strongly convex, unimodal, separable and many other special problems.
- On the other hand, EAs do not give up so early when discontinuous, nondifferentiable, multimodal, noisy and otherwise unconventional response surfaces are involved.
- EAs show inefficiency on the classes of simple problems, but the effectiveness or robustness of them extends to a broader field of applications.

Genetic Algorithms

A simple GA works as follows:

1. Start with a randomly generated population of *n* individuals
2. Calculate the fitness *f(x)* of each individual in the population
3. Repeat the following steps until a new population is created:
   i. Select a pair of parent from the current population

Genetic Algorithms

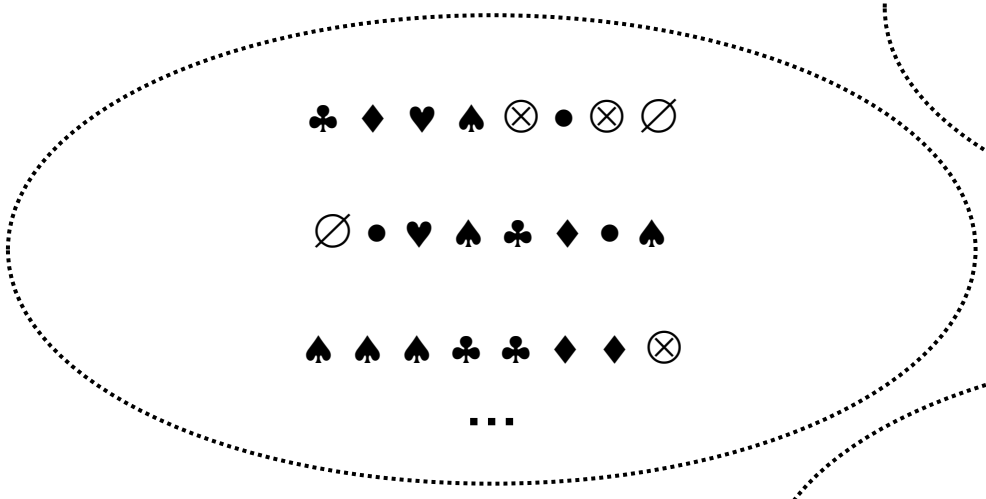
ii. Cross over the pair with crossover probability *Pc* at a randomly chosen point to form two offspring

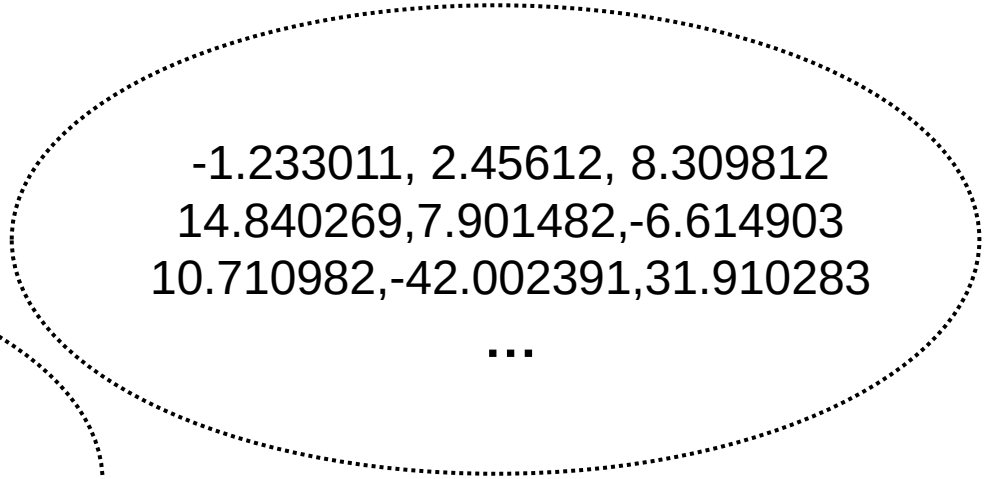
iii. Mutate the two offspring at each locus with probability *Pm* and place the resulting individuals in the new population

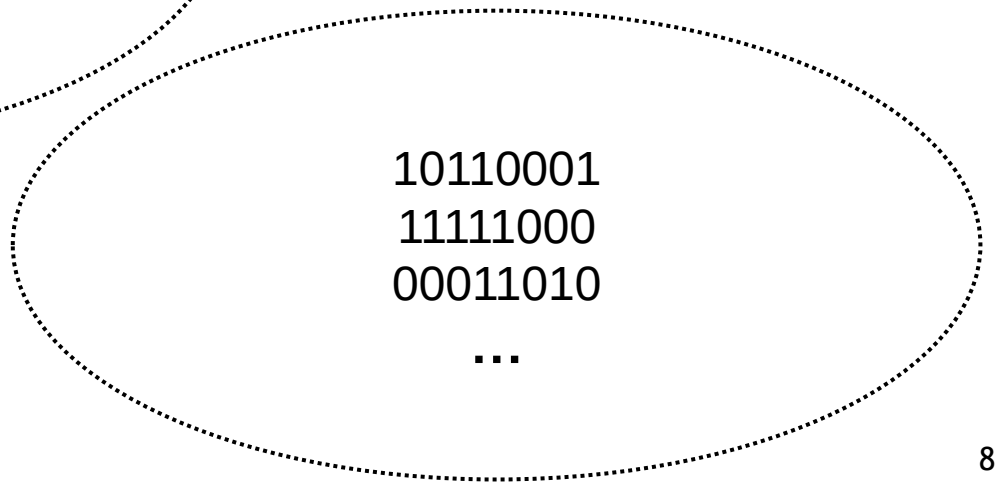
4. Replace the current population with the new population

5. While the termination condition is false go to step 2.

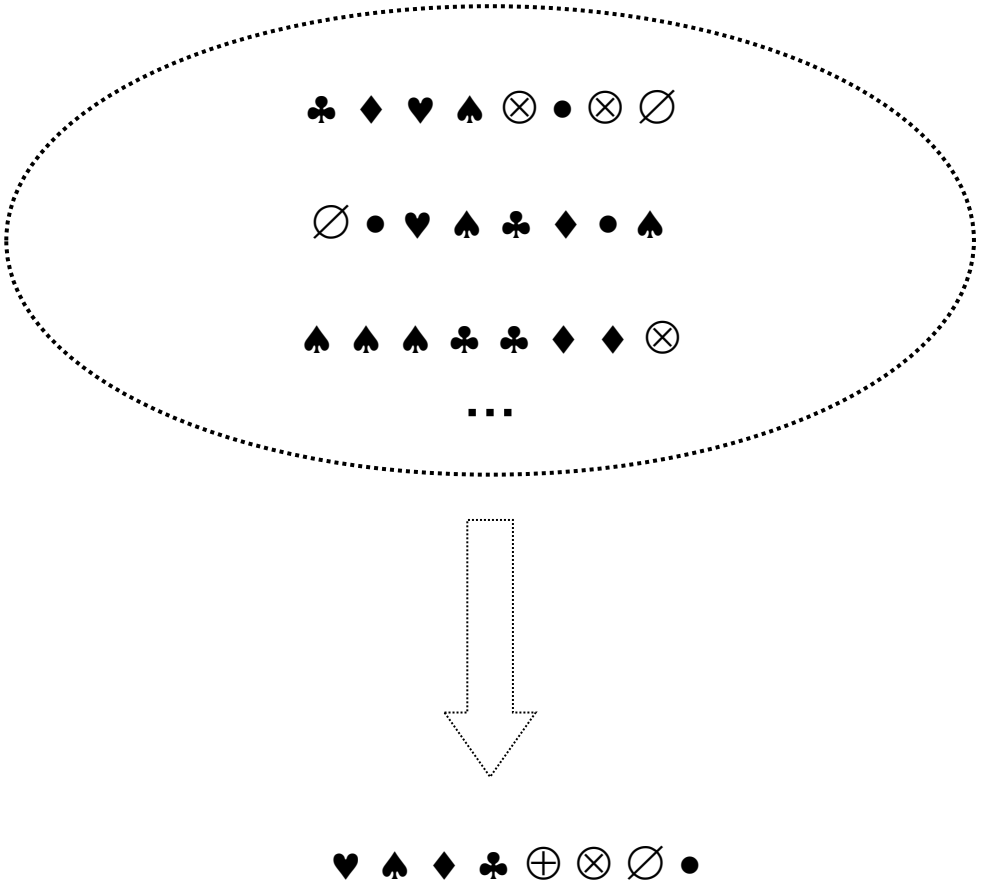
Representation







Evolutionary Process



Parameters in Evolutionary Algorithms

- Evolutionary algorithms are affected by more parameters than optimization methods typically.
- This is at the same time a source of their robustness as well as a source of frustration in designing them.
- Adaptation can be used not only for finding solutions to a given problem, but also for tuning genetic algorithms to the particular problem

Adaptation

- Adaptation can be applied to problems as well as to evolutionary processes.
  - In the first case, adaptation modifies some components of genetic algorithms to provide an appropriate form of the algorithm, which meets the nature of the given problem.
  - These components could be any of representation, crossover, mutation and selection.

Adaptation

- In the second case, adaptation suggests a way to tune the parameters of the changing configuration of genetic algorithms while solving the problem.
- Some of such parameters are:
  - population size and structure, like subpopulations
  - genome representation (floating point, binary, parse tree, matrix), precision and length
  - crossover type (arithmetic, -point, etc.), the number of crossover points and probability
  - mutation type (uniform, Gaussian, etc.), mutation variance and probability
  - selection type (tournament, proportional, etc.), tournament size.

Optimal Parameters

- The challenge is that optimal parameters of an EA are problem dependent and there is a large set of possible EA settings.
- The No-Free-Lunch theorem implies that no set of parameters for an EA is superior on all problems.
- Finding the right parameter values is a time-consuming task and it has been the subject of many researches.

Parameter Setting Methods

- The main criteria for classifying parameter setting methods are:

1) what is changed:
   - representation
   - evaluation function
   - variation operators (mutation and recombination)
   - selection
   - replacement
   - population

Parameter Setting Methods

2) How the change is made:

- deterministic (or fixed) parameter control (parameter tuning) in which the parameter-altering transformations takes no input variables related to the progress of search method
- adaptive (also called explicitly adaptive) parameter control in which there is some form of feedback from the search
- self-adaptive (implicitly adaptive) parameter control in which the parameters to be adapted are encoded into the chromosomes and undergo mutation and recombination

Mutation

- Mutation is a bit reversal event that occurs with small probabilities per bit.

Binary Mutation:

1 0 0 1 1 1 0 0 0 0 0 0 1 1 ⇨ 10.95

$-50 \leq x \leq 50$

0 0 0 1 1 1 0 0 0 0 0 0 1 1 0 ⇨ -39.05

Real Value Mutation:

6.290351 ⇨ 6.290102

Mutation Operators

- Efforts to tune the mutation probability have resulted to different values and hence leaving practitioners in ambiguity.
- As results of tuning “optimal” mutation rate, the best rate found to be Pm=0.001 (De Jong 1975), Pm=0.01(Grefenstette 1986), 0.005≤Pm≤0.01 (Schaffer et al. 1989) and Pm=1/L (Mühlenbein 1992), where L is the length of the bit string (Michalewicz et al. 2004).

Adaptation for Mutation Rate

- Controlling the mutation rate in bit-flip mutation (Fogarty 1989; Ursem 2003):

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t}$$

Adaptation for Mutation Rate

- A theoretically optimal schedule for deterministically changing *Pm* for the counting-ones function is presented in (Hesser & Männer 1991; Eiben et al. 1999; Michalewicz et al. 2004):

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \times \frac{\exp(\frac{-\gamma t}{2})}{\lambda\sqrt{L}}$$

Adaptation for Mutation Rate

- An optimal schedule for decreasing the mutation rate as a function of the distance to the optimum is defined in (Bäck 1992 a; Eiben et al. 1999; Michalewicz et al. 2004) in the following way:

$$p_m(f(x)) \approx \frac{1}{2(f(x)+1)-L}$$

Adaptation for Mutation Rate

- Controlling the variance in Gaussian mutation is very critical in successful application of real-encoded EAs.
- The standard approach for doing this is to set the variance of the mutation according to a monotonic decreasing function depending on the generation number.

Adaptation for Mutation Rate

- Gaussian mutation of a real-encoded variable is usually performed according to:

$$x_i' = x_i + N(0, \sigma_i(t))$$

- The mutation variance is traditionally set using either a linear or an exponentially decreasing function such as:

$$\sigma_i(t) = 1/\sqrt{1+t}$$

Adaptation for Mutation Rate

- The mutation rate *Pm* of GAs can also be self-adapted by adding the rate of mutating , coded in bits, to every individual.
- Then the new is used to mutate the individual's object variables.
- This is based on the idea that better *Pm* rates will produce better offspring and then hitchhike on their improved children to new generations, while bad rates will die out

Adaptation for Mutation Rate

- Mutating a floating-point object variable in a self-adaptive way may happen in the following way:

$$x_i' = x_i + \sigma_i N(0,1)$$

- where the mean step sizes can be modified for instance lognormally:

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1))$$

Intelligent Mutation Operators

- **Problem with the classical implementation of binary mutation**: it is difficult to control effect or to restrict changes caused by multiple point mutation or the crossover operator within certain limits
- **Solution**: implement the genetic operators intelligently so that the resulting modifications on the binary string will cause changes in the real values within the desired limits

Mutation

Binary Mutation:

1 0 0 1 1 1 0 0 0 0 0 0 1 1 ⇨ 10.95

$-50 \leq x \leq 50$

0 0 0 1 1 1 0 0 0 0 0 0 1 1 0 ⇨ -39.05

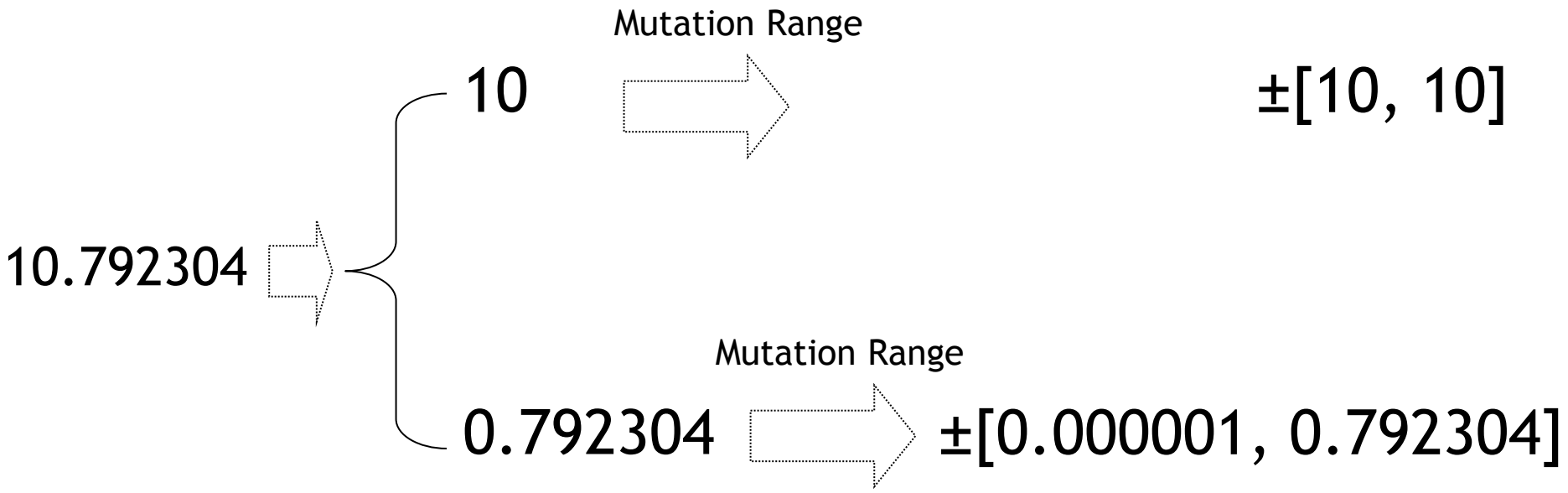
Real Value Mutation:

6.290351 ⇨ 6.290102

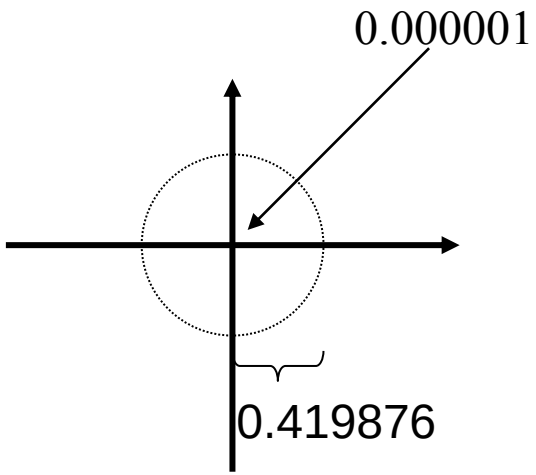
Intelligent Mutation Operators

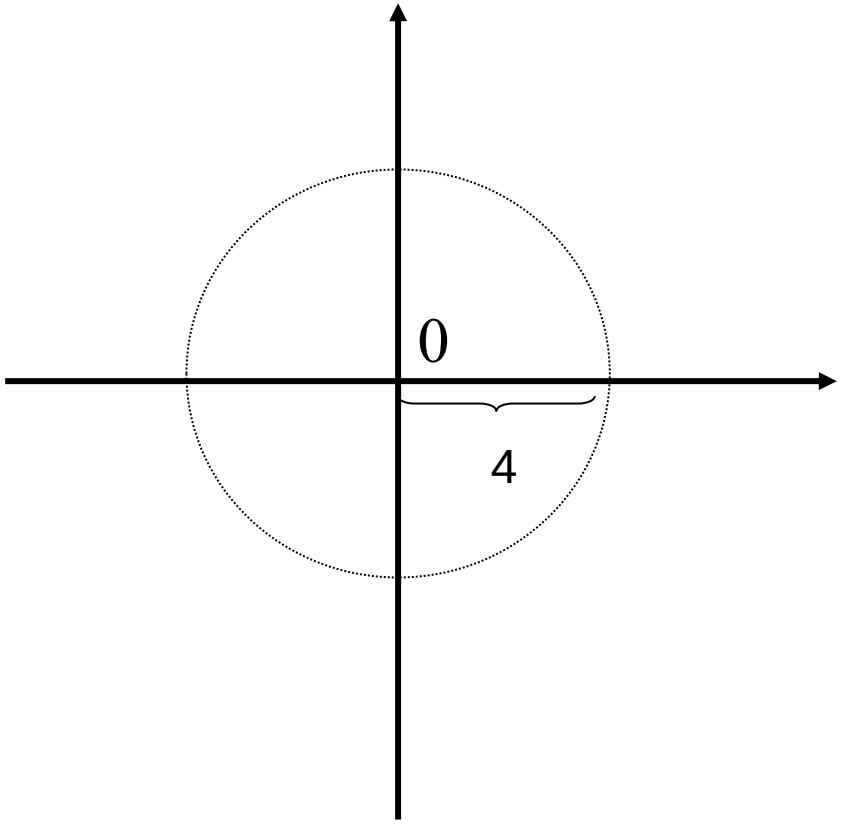
- Changes of different magnitudes are required at different stages of the evolutionary process:
- Modifying variables with integer values. The bounds for the absolute values of such changes are at least 1 and at most the integer part of the real value representation of the variable.
- Modifying variables with values from the range 0<x<1. The lower bound for the absolute value of such changes is determined by the required precision of the real value presentation of the variable, like 10-6. The upper bound for the absolute value of such changes is determined by decimal part of the variable.

Intelligent Mutation Operators



Intelligent Mutation Operators





Experimentation & Conclusions

- These operators were tested on 44 test problems in 2200 runs.
- Experimentation showed that the most efficient operators are the integer mutation and the decimal mutation operators, which were able to improve the population fitness values the most.

Thank you!

☺