Chapter 11: Evolutionary Programming

Computational Intelligence: Second Edition

Contents

Compact Overview

- Originated from the research of L.J. Fogel in 1962 on using simulated evolution to develop artificial intelligence
- Differs substantially from GA and GP, in that evolutionary programming (EP) emphasizes the development of behavioral models and not genetic models
- EP is derived from the simulation of adaptive behavior in evolution
- EP considers phenotypic evolution
- EP iteratively applies two evolutionary operators:
  - Variation through application of mutation operators
  - Selection

Basic Evolutionary Programming Algorithm: Algorithm 11.1

$t = 0$, initialize strategy parameters;
Create and initialize the population, $\mathcal{C}(0)$, of n_s individuals;
Evaluate the fitness, $f(\mathbf{x}_i(t))$, of each individual;
while *stopping condition(s) not true* **do**
  for *each individual,* $\mathbf{x}_i(t) \in \mathcal{C}(t)$ **do**
    Create an offspring, $\mathbf{x}_i^{'}(t)$, by applying the mutation operator;
    Evaluate the fitness, $f(\mathbf{x}_i^{'}(t))$;
    Add $\mathbf{x}_i^{'}(t)$ to the set of offspring, $\mathcal{C}^{'}(t)$;
  end
  Select the new population, $\mathcal{C}(t+1)$, from $\mathcal{C}(t) \cup \mathcal{C}^{'}(t)$;
  $t = t + 1$;
end

Basic Components

- The main components of an EP:
  - Initialization
  - Evaluation
    - Fitness function measures the "behavioral error" of an individual with respect to the environment of that individual
    - provides an absolute fitness measure of how well the problem is solved
    - Survival in EP is usually based on a relative fitness measure
    - A score is computed to quantify how well an individual compares with a randomly selected group of competing individuals
    - Individuals that survive to the next generation are selected based on this relative fitness
    - The search process in EP is therefore driven by a relative fitness measure, and not an absolute fitness measure

Basic Components (cont)

- The main components of an EP (cont):
  - Mutation as the only source of variation
  - Selection
    - Main purpose to select new population
    - A competitive process where parents and offspring compete to survive
- Behaviors of individuals are influenced by strategy parameters

Mutation Operators

- Mutation is the only means of introducing variation in an EP population
- In general, mutation is defined as

$$x'_{ij}(t) = x_{ij}(t) + \Delta x_{ij}(t)$$

  $\mathbf{x}'_i(t)$ is the offspring and $\Delta \mathbf{x}_i(t)$ is the mutational step size
- Mutational step size:
  - Noise sampled from some probability distribution
  - Deviation of noise is determined by a strategy parameter, σ_{ij}
  - Generally, the step size is calculated as

$$\Delta x_{ij}(t) = \Phi(\sigma_{ij}(t))\eta_{ij}(t)$$

    $\Phi : \mathbb{R} \rightarrow \mathbb{R}$ scales the contribution of the noise

Categories of EP Algorithms

- Based on the characteristics of the scaling function, Φ, the following categories of EP algorithms:
  - **non-adaptive** EP, in which case $\Phi(\sigma) = \sigma$. In other words, the deviations in step sizes remain static.
  - **dynamic** EP, where the deviations in step sizes change over time using some deterministic function, Φ, usually a function of the fitness of individuals.
  - **self-adaptive** EP, in which case deviations in step sizes change dynamically. The best values for σ_{ij} are learned in parallel with the decision variables, x_{ij}.

Strategy Parameters

- The deviations, σ_{ij}, are referred to as strategy parameters
- Each individual has its own strategy parameters
- An individual is represented as the tuple,

$$\chi_i(t) = (\mathbf{x}_i(t), \sigma_i(t))$$

- Correlation coefficients between components of the individual have also been used as strategy parameters

Mutation Distributions: Uniform

$$\eta_{ij}(t) \sim U(x_{min,j}, x_{max,j})$$

- $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ provide lower and upper bounds for the values of η_{ij}
- Note that

  $$E[\eta_{ij}] = 0$$

  to prevent any bias induced by the noise
- Alternative uniform mutation:

  $$\Delta x_{ij}(t) = U(0,1)(\hat{y}_j(t) - x_{ij}(t))$$

  directing all individuals to make random movements towards the best individual

Mutation Distributions: Gaussian

$$\eta_{ij}(t) \sim N(0, \sigma_{ij}(t))$$

- The Gaussian density function:

$$f_G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/(2\sigma^2)}$$

where σ is the deviation of the distribution

Mutation Distributions: Cauchy

$$\eta_{ij}(t) \sim C(0, \nu)$$

- ν is the scale parameter
- Cauchy density function centered at the origin

$$f_C(x) = \frac{1}{\pi} \frac{\nu}{\nu + x^2}$$

  for $\nu > 0$
- Distribution function

$$F_C(x) = \frac{1}{2} + \frac{1}{\pi} \arctan(\frac{x}{\nu})$$

- Cauchy distribution has wider tails than the Gaussian distribution, producing more, larger mutations

Mutation Distributions: Lévy

$$\eta_{ij}(t) \sim L(\nu)$$

- Lévy probability function, centered around the origin

$$F_{L,\nu,\gamma}(x) = \frac{1}{\pi}\int_0^\infty e^{-\gamma q^\nu}\cos(qx)dq$$

- $\gamma > 0$ is the scaling factor
- $0 < \nu < 2$ controls the shape of the distribution
- If $\nu = 1$, the Cauchy distribution is obtained
- If $\nu = 2$, the Gaussian distribution is obtained
- For $|x| >> 1$, the Lévy density function can be approximated by

$$f_L(x) \propto x^{-(\nu+1)}$$

Mutation Distributions: Exponential

$$\eta_{ij}(t) \sim E(0, \xi)$$

- Density function of the double exponential probability distribution

$$\eta_{ij}(t) \sim E(0, \xi)$$

- $\xi > 0$ controls the variance (which is equal to $\frac{2}{\xi^2}$)
- random numbers can be calculated as follows:

$$x = \begin{cases} \frac{1}{\xi}\ln(2y) & \text{if } y \leq 0.5 \\ -\frac{1}{\xi}\ln(2(1-y)) & \text{if } y > 0.5 \end{cases}$$

where $y \sim U(0, 1)$.

Mutation Distributions: Chaos

$$\eta_{ij}(t) \sim R(0,1)$$

- $R(0,1)$ represents a chaotic sequence within the space $(-1,1)$
- The chaotic sequence can be generated using

$$x_{t+1} = \sin(2/x_t)x_t, \quad t = 0, 1 \ldots$$

Mutation Distributions: Combined

- Mean mutation operator (MMO)

$$\eta_{ij}(t) = \eta_{N,ij}(t) + \eta_{C,ij}(t)$$

where

$$\begin{aligned} \eta_{N,ij} &\sim N(0,1) \\ \eta_{C,ij} &\sim C(0,1) \end{aligned}$$

- Generates more very small and large mutations compared to the Gaussian distribution
- Generates more very small and small mutations compared to the Cauchy distribution
- Generally, produces larger mutations than the Gaussian distribution, and smaller mutations than the Cauchy distribution

Selection Operators

- The selection operator is used to create the new population
- New population is selected from parents and their offspring
- Competition to survive is based on a relative fitness measure
- EP notation:
  - μ indicates the number of parent individuals
  - λ indicates the number of offspring
- Selection consists of two steps:
  - Calculate score, or relative fitness
  - Select new population

Selection Operators: Compute Relative Fitness

- Define $\mathcal{P}(t) = \mathcal{C}(t) \cup \mathcal{C}'(t)$ to be the competition pool
- Let $\mathbf{u}_i(t) \in \mathcal{P}(t), i = 1, \ldots, \mu + \lambda$ denote an individual in the competition pool
- For each $\mathbf{u}_i(t) \in \mathcal{P}(t)$, randomly select a group of competitors, excluding $\mathbf{u}_i(t) \in \mathcal{P}(t)$
- Calculate the score

$$s_i(t) = \sum_{l=1}^{n_{\mathcal{P}}} s_{il}(t)$$

where

$$s_{il}(t) = \begin{cases} 1 & \text{if } f(\mathbf{u}_i(t)) < f(\mathbf{u}_l(t)) \\ 0 & \text{otherwise} \end{cases}$$

Selection Operators: Select New Population

- **Elitism:** the best μ individuals from $\mathcal{P}(t)$ are selected
- **Tournament** selection: the best μ individuals are stochastically selected using tournament selection.
- **Proportional** selection: each individual is assigned a probability of being selected:

$$p_s(\mathbf{u}_i(t)) = \frac{s_i(t)}{\sum_{l=1}^{2\mu} s_l(t)} \tag{1}$$

  Use Roulette-wheel selection to select the μ survivors
- **Nonlinear ranking** selection: Individuals are sorted in ascending order of score and then ranked

$$p_s\left(\mathbf{u}_{(2\mu-i)}(t)\right) = \frac{i}{\sum_{l=1}^{2\mu} l} \tag{2}$$

  Use Roulette-wheel selection to select survivors

Selection Operators: Select New Population

- Direct competition between parent and offspring:
  - Create a number of offspring from the parent
  - Select the best offspring
  - The offspring, $\mathbf{x}_i^{'}(t)$, survives to the next generation if $f(\mathbf{x}_i^{'}(t)) < f(\mathbf{x}_i(t))$ or if

    $$e^{(-(f(\mathbf{x}_i^{'}(t))-f(\mathbf{x}_i(t)))/\tau(t))} > U(0,1) \tag{3}$$

    where τ is the temperature coefficient, with $\tau(t) = \gamma\tau(t-1), 0 < \gamma < 1$; otherwise the parent survives
  - The offspring has a chance of surviving even if it has a worse fitness than the parent
  - This reduces selection pressure and improves exploration

Introduction to Strategy Parameters

- Mutational step sizes are dependent on strategy parameters
- Usually each component of each individual has its own strategy parameter
- However, a single strategy parameter can also be associated with a single individual
- A single strategy parameter limits degrees of freedom in addressing the exploration–exploitation trade-off

Static Strategy Parameters

- Values of deviations are fixed, using a linear strategy parameter:
$$\Phi(\sigma_{ij}(t)) = \sigma_{ij}(t) = \sigma_{ij}$$
where σ_{ij} is a small value
- Offspring are created as
$$x'_{ij}(t) = x_{ij}(t) + \Delta x_{ij}(t)$$
with $\Delta x_{ij}(t) = N_{ij}(0, \sigma_{ij})$
- A too small value for σ_{ij} limits exploration and slows down convergence
- A too large value for σ_{ij} limits exploitation and the ability to fine-tune a solution

Dynamic Strategy Parameters

- Having strategy parameters proportional to fitness

$$\sigma_{ij}(t) = \sigma_i(t) = \gamma f(\mathbf{x}_i(t))$$

$\gamma \in (0, 1]$

- Offspring is generated using

$$\begin{aligned} x'_{ij}(t) &= x_{ij}(t) + N(0, \sigma_i(t)) \\ &= x_{ij}(t) + \sigma_i(t) N(0, 1) \end{aligned}$$

- Fitness proportional to error wrt best solution

$$\sigma_{ij}(t) = \sigma_i(t) = |f(\hat{\mathbf{y}}) - f(\mathbf{x}_i)|$$

$\hat{\mathbf{y}}$ is the most fit individual

Self-Adaptation

- Major issues with regards to strategy parameters:
  - Amount of mutational noise to be added
  - Severity of noise, i.e. step sizes
- Usually best practice wrt these issues is problem dependent
- A solution is to self-adapt strategy parameters during the search process

Self-Adaptation (cont)

- Additive methods:

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta\sigma_{ij}(t)N_{ij}(0,1)$$

η is the learning rate

- In the first application of this approach, $\eta = 1/6$
- If $\sigma_{ij}(t) \leq 0$, then $\sigma_{ij}(t) = \gamma$, where γ is a small positive constant (typically, $\gamma = 0.001$) to ensure positive, non-zero deviations
- An alternative:

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \sqrt{f_\sigma(\sigma_{ij}(t))}N_{ij}(0,1)$$

where

$$f_\sigma(a) = \begin{cases} a & \text{if } a > 0 \\ \gamma & \text{if } a \leq 0 \end{cases}$$

Self-Adaptation (cont)

- Multiplicative methods:

$$\sigma_{ij}(t+1) = \sigma(0)(\lambda_1 e^{-\lambda_2 \frac{t}{n_t}} + \lambda_3)$$

λ_1, λ_2 and λ_3 are control parameters

- Lognormal methods:

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) e^{(\tau N_i(0,1) + \tau' N_{ij}(0,1))}$$

where

$$\tau' = \frac{1}{\sqrt{2\sqrt{n_x}}}, \ \tau = \frac{1}{\sqrt{2 n_x}}$$

  - Offspring is generated using

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij}(t) N_{ij}(0,1)$$

Self-Adaptation (cont)

- Undesirable property of self-adaptation:
  - Stagnation due to strategy parameters converging too fast
  - Deviations becomes too small too fast, limiting exploration
  - Search then stagnates until strategy parameters grow sufficiently large due to random variation
- Solution:
  - Impose a lower bound on values of σ_{ij}
  - Use dynamic lower bounds

$$\sigma_{min}(t+1) = \sigma_{min}(t)\left(\frac{n_m(t)}{\xi}\right)$$

$\sigma_{min}(t)$ is the lower bound at generation t
$\xi \in [0.25, 0.45]$ is the reference rate
$n_m(t)$ is the number of successful consecutive mutations

Classical Evolutionary Programming (CEP)

- EP with Gaussian mutation
- Uses lognormal self-adaptation
- Produces offspring using

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij}(t)N_{ij}(0,1)$$

- Elitism selection is used to select new population from parents and offspring

Fast Evolutionary Programming (FEP)

- Mutational noise is sampled from the Cauchy distribution with $\nu = 1$
- Uses lognormal self-adaptation
- Offspring is generated using

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij}(t)C_{ij}(0,1)$$

- Elitism selection is used to select new population from parents and offspring
- The wider tails of the Cauchy distribution provide larger step sizes
- Therefore, faster convergence and better exploration

Improved FEP

- FEP showed that step sizes may be too large for proper exploitation
- CEP showed a better ability to fine-tune solutions
- The improved FEP:
  - For each parent, IFEP generates two offspring
  - One offspring is generated using Gaussian mutation, and the other using Cauchy mutation
  - The best offspring is chosen as the surviving offspring

Exponential Evolutionary Programming

- Uses the double exponential probability distribution to sample mutational noise
- Offspring are generated using

$$x'_{ij}(t) = x_{ij}(t) + \sigma_{ij}(t)\frac{1}{\xi}E_{ij}(0,1)$$

  σ_{ij} is self-adapted
- The variance of the distribution is controlled by ξ
  - The smaller the value of ξ, the greater the variance
  - Larger values of ξ result in smaller step sizes
  - To ensure initial exploration and later exploitation, ξ can be initialized to a small value that increases with time

Accelerated Evolutionary Programming

- Individuals are represented as

$$\chi_i(t) = (\mathbf{x}_i(t), \rho_i(t), a_i(t))$$

  $\rho_{ij} \in \{-1, 1\}, j = 1, \ldots, n_x$ gives the search direction
  a_i represents the age of the individual
- Age is used to force wider exploration if offspring are worse than their parents
- Offspring generation consists of two steps
  - Update age parameters and search directions

$$a_i(t) = \begin{cases} 1 & \text{if } f(\mathbf{x}_i(t)) < f(\mathbf{x}_i(t-1)) \\ a_i(t-1)+1 & \text{otherwise} \end{cases}$$

$$\rho_{ij}(t) = \begin{cases} \mathrm{sign}(x_{ij}(t) - x_{ij}(t-1)) & \text{if } f(\mathbf{x}_i(t)) < f(\mathbf{x}_i(t-1)) \\ \rho_{ij}(t-1) & \text{otherwise} \end{cases}$$

Accelerated Evolutionary Programming (cont)

- Offspring generation (cont)
  - If the fitness does not improve, increase in age cause larger step sizes
    - If $a_i(t) = 1$

$$\begin{aligned} \sigma_i(t) &= \gamma_1 f(\mathbf{x}_i(t)) \\ x'_{ij}(t) &= x_{ij}(t) + \rho_{ij}(t)|N(0, \sigma_i(t))| \end{aligned}$$

    - If $a_i(t) > 1$

$$\begin{aligned} \sigma_i(t) &= \gamma_2 f(\mathbf{x}_i(t)) a_i(t) \\ x'_{ij}(t) &= x_{ij}(t) + N(0, \sigma_i(t)) \end{aligned}$$

- Selection: offspring competes directly with its parent using absolute fitness

Momentum Evolutionary Programming

- Offspring is generated as follows:

$$x'_{ij}(t) = x_{ij}(t) + \eta \Delta x_{ij}(t) + \alpha \tilde{x}_{ij}(t)$$

where

$$\begin{aligned} \Delta x_{ij}(t) &= (\hat{y}_j(t) - x_{ij}(t))|N_{ij}(0,1)| \\ \tilde{x}_{ij}(t) &= \eta \rho_i(t) \Delta x_{ij}(t-1) + \alpha \tilde{x}_{ij}(t-1) \end{aligned}$$

with $\eta > 0$ the learning rate, $\alpha > 0$ the momentum rate, and

$$\rho_i(t) = \begin{cases} 1 & \text{if } f(\mathbf{x}'_i(t-1)) < f(\mathbf{x}_i(t-1)) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Evolutionary Programming with Local Search

- To improve exploitation ability, by adding hill-climbing to generate offspring
- While a better fitness can be obtained, hill-climbing is applied to each offspring:

$$x'_{ij}(t) = x'_{ij}(t) - \eta_i(t)\frac{\partial f}{\partial x_{ij}(t)}$$

- The learning rate is calculated using

$$\eta_i(t) = \frac{\sum_{j=1}^{n_x} \frac{\partial f}{\partial x_{ij}(t)}}{\sum_{h=1}^{n_x} \sum_{j=1}^{n_x} \frac{\partial^2 f}{\partial x_{ih}(t)\partial x_{ij}(t)} \frac{\partial f}{\partial x_{ih}(t)} \frac{\partial f}{\partial x_{ij}(t)}}$$

Evolutionary Programming Hybrid with PSO

$$x_{ij}(t+1) = x_{ij}(t) + \nu_{ij}(t) + \sigma_i N_{ij}(0,1)$$

- Classical EP mutation operator is used
- Can use any other EP mutation operator