Evolutionary Programming

Presented by:
Faramarz Safi (Ph.D.)
Department of Computer Engineering
Islamic Azad University, Najafabad Branch

Chapter 5

EP quick overview

- Developed: USA in the 1960's
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: machine learning tasks by finite state machines
  - contemporary EP: (numerical) optimization
- Attributed features:
  - very open framework: any representation and mutation op's OK
  - crossbred with ES (contemporary EP)
  - consequently: hard to say what "standard" EP is
- Special:
  - no recombination
  - self-adaptation of parameters standard (contemporary EP)

EP technical summary tableau

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic
Survivor selection	Probabilistic $(\mu+\mu)$
Specialty	Self-adaptation of mutation step sizes (in meta-EP)

Historical EP perspective Introductory Example

- EP aimed at achieving intelligence
- Intelligence was viewed as adaptive behaviour
- Prediction of the environment was considered a prerequisite to adaptive behaviour
- Thus: capability to predict is key to intelligence

Prediction by finite state machines

- Finite state machine (FSM):
  - States S
  - Inputs I
  - Outputs O
  - Transition function $\delta : S \times I \rightarrow S \times O$
  - Transforms input stream into output stream
- Can be used for predictions, e.g. to predict next input symbol in a sequence

Prediction by finite state machines

- In the classical example of EP, predictors were evolved in the form of finite state machines.
- A finite state machine (FSM) is a transducer that can be stimulated by a finite alphabet of input symbols and can respond in a finite alphabet of output symbols.
- It consists of a number of states S and a number of state transitions. The state transitions define the working of the FSM:
- depending on the current state and the current input symbol, they define an output symbol and the next state to go to.

FSM example

- An example three-state FSM consider the FSM with:
  - $S = \{A, B, C\}$
  - $I = \{0, 1\}$
  - $O = \{a, b, c\}$
  - δ given by a diagram

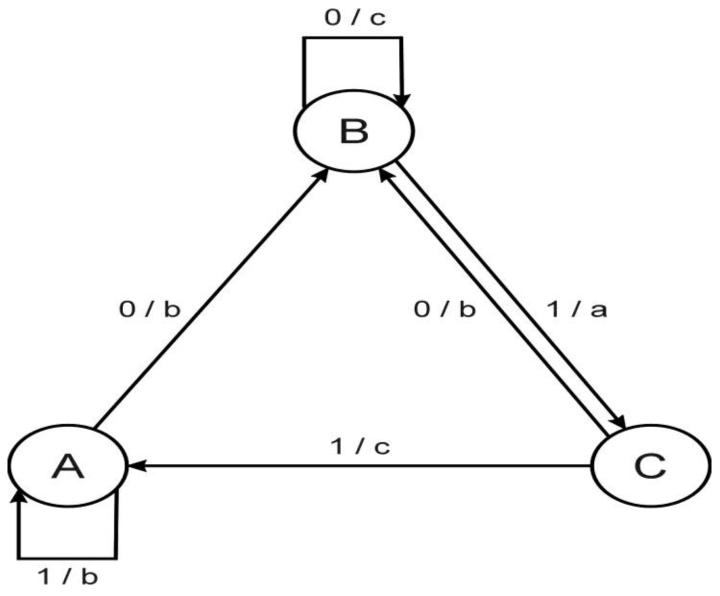

Fig. 5.1. Example of a finite state machine consisting of three states: A, B, and C. The input alphabet is $I = \{0, 1\}$, and the output alphabet is $O = \{a, b, c\}$. The FSM's transition function $\delta : S \times I \rightarrow S \times O$ that transforms the input stream to the output stream is specified by the arrows and their labels indicating the input/output of the given transition

FSM as predictor

- A simple prediction task to be learned by an FSM is to guess the following input symbol in an input stream. That is, considering n inputs, predict the *(n+1)th* one, and articulate this prediction by the *nth* output symbol.
- In this case, the performance of an FSM is measured by the percentage of inputs where $input_{n+1} = output_n$. Clearly, this requires the input alphabet and the output alphabet to be the same.
- Consider the following FSM:
  - Task: predict next input
  - Quality: % of $in_{(i+1)} = out_i$
  - Given initial state C
  - Input sequence 011101
  - Leads to output 110111
  - Quality: 3 out of 5

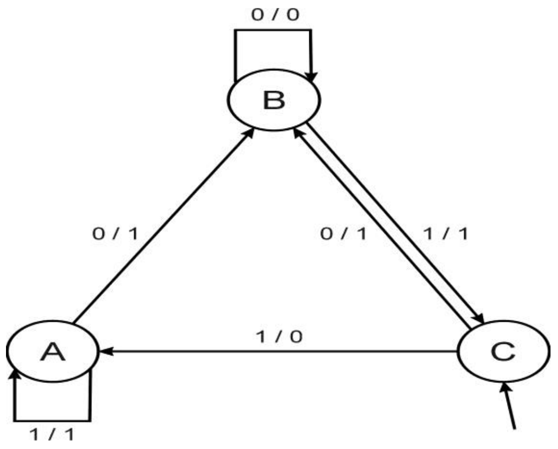

Fig. 5.2. Finite state machine as a predictor. The initial state of this FSM is C, and the given input string is 011101. The FSM's response is the output string 110111. On this string, its prediction performance is 60% (inputs 2, 3, and 6 correctly predicted)

Introductory example: evolving FSMs to predict primes

- Fogel et al. [156] describe an experiment where predictors were evolved to tell whether the next input (being an integer) in a sequence is a prime or not.
- For this task FSMs were used as individuals with the input alphabet $I = \mathbb{N}$ and output alphabet O = {0,1}.
- The fitness of an FSM was defined as its prediction accuracy on the input sequence of consecutive integers 1,2,3,... (minus some penalty for containing too many states).
- Many technical details of this application are hardly traceable today, but [145] and personal communication give some details. Parent selection does not take place, but each FSM in the given population is mutated once to generate one offspring.

Introductory example: evolving FSMs to predict primes

- P(n) = 1 if n is prime, 0 otherwise
- I = ***N*** = {1,2,3,…, n, …}
- O = {0,1}
- Correct prediction: $out_i = P(in_{(i+1)})$
- Fitness function:
  - 1 point for correct prediction of next input
  - 0 point for incorrect prediction
  - Penalty for “too much” states

Introductory example: evolving FSMs to predict primes

- Parent selection: each FSM is mutated once
- Mutation operators (one selected randomly):
  - Change an output symbol
  - Change a state transition (i.e. redirect edge, or change the next stage)
  - Add a state
  - Delete a state
  - Change the initial state
- Survivor selection: ($\mu+\mu$)

Introductory example: evolving FSMs to predict primes

A choice from these mutation operators is made randomly with a uniform distribution. Recombination (crossover) is not used, and after having created μ offspring from a population of μ FSMs, the top 50% of their union is saved as the next generation.

The results obtained with this system show that after 202 input symbols the best FSM is a very opportunistic one, having only one state and always guessing "no" (output 0). Given the sparcity of primes, this strategy is good enough for accuracies above 81%. Using more sophisticated setups these outcomes could be improved.

However, the main point was not perfect accuracy (which is theoretically impossible), but the empirical proof that a simulated evolutionary process is able to create good solutions for an intelligent task.

For historical reasons EP has been long associated with prediction tasks and the use of finite state machines as their representation. However, since the 1990s, EP variants for optimization of real valued parameter vectors have become more frequent and even positioned as “standard” EP [22, 31].

Modern EP

Today the EP community typically considers EP as a very open framework in terms of representation and mutation operators.

- No predefined fixed representation in general, but derived from the problem to be solved.
- Thus: no predefined mutation (must match representation)
- Often applies self-adaptation of mutation parameters
- In the sequel, we present ***one EP variant***, **not the canonical EP**.

Representation

EP is used for many different applications, and takes a very pragmatic approach of choosing the representation based on the problem's features.

- For continuous parameter optimization
- Chromosomes consist of two parts:
  - Object variables: $x_1,\ldots,x_n$
  - Mutation step sizes: $\sigma_1,\ldots,\sigma_n$
- Adding strategy parameters to the individuals. Similar to ES: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \rangle$

Mutation

There is no single EP mutation operator:

- Mutatatin transforms chromosomes $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n \rangle$ into $\langle x'_1,\ldots, x'_n, \sigma_1',\ldots,\sigma_n' \rangle$
- $\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1))$
- $x'_i = x_i + \sigma_i' \cdot N_i(0,1)$
- N(0,1) denotes the outcome of a random drawing from a Gaussian distribution with zero mean and standard deviation 1< and with $\alpha \approx 0.2$>
- A boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$ to prevent standard deviations very close to zero.
- Other mutation variants proposed & tried:
  - Lognormal scheme as in ES
  - Using variance instead of standard deviation
  - Mutate σ-last
  - Other distributions, e.g, Cauchy instead of Gaussian

Recombination

- The issue of recombination in EP can be handled very briefly since it is not used. In the beginning, recombination of FSMs was proposed, based on a majority vote mechanism, **but this was never incorporated into the EP algorithm.**
- Rationale: a point in the search space is not viewed as an individual of some species, but as the abstraction of a species itself. As a consequence, recombination does not make sense as it cannot be applied to different species (cross over between different species).
- Technically, of course, it is possible to design and apply variation operators merging information from two or more individuals.
- Much historical debate “mutation vs. crossover”, and pragmatic approaches seems to prevail today

Parent selection

- The question of selecting parents to create offspring is almost nonissue for EP, and this distinguishes it from the other EA dialects.
- In EP, every member of the population creates exactly one offspring via mutation.
- In this way, it differs from GAs and GP, where selective pressure based on fitness is applied at this stage.
- It also differs from ES, since the choice of parents in EP is deterministic, whereas in ES it is stochastic.
- For instance, in ES, each parent takes part in on average λ/μ offspring creation events, but possibly in none for EP.
- Thus, each individual creates one child by mutation.
- Thus:
  - Deterministic
  - Not biased by fitness

Survivor selection

- The selection operator is generally ($\mu+\mu$) selection. P(t): μ parents, P'(t): μ offspring
- **Pairwise competitions in round-robin format** involving both parent and offspring populations:
  - Each solution x from $P(t) \cup P'(t)$ is evaluated against q other randomly chosen solutions
  - For each comparison, a "win" is assigned if x is better than its opponent.
  - The μ solutions with the greatest number of wins are retained to be parents of the next generation
- Parameter q allows tuning selection pressure, typically q = 10 is recommended.
- It is worth noting that this variant of selection allows for less-fit solutions to survive into the generation if they had a lucky draw of opponents.
- As the value of q increases this chance becomes more and unlikely, until in the limit the mechanism becomes deterministic $\mu+\mu$ as in the case of evolution strategies.

Example application: the Ackley function (Bäck et al ’93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$$

- Representation:
  - $-30 < x_i < 30$ (coincidence of 30’s!)
  - 30 variances as step sizes
- Mutation with changing object variables first !
- Population size $\mu = 200$, selection with q = 10
- Termination : after 200000 fitness evaluations
- Results: average best solution is $1.4 \cdot 10^{-2}$

Example application: the Ackley function (Bäck et al ’93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

- Representation:
  - $-30 < x_i < 30$ (coincidence of 30’s!)
  - 30 variances as step sizes
- Mutation with changing object variables first !
- Population size $\mu = 200$, selection with q = 10
- Termination : after 200,000 fitness evaluations
- Results: average best solution is $1.4 \cdot 10^{-2}$

Example application: evolving checkers players (Fogel'02)

Human checker players regularly compete against each other in a variety of tournaments (often Internet-hosted), and there is a standard scheme for rating a player according to their results.

In order to play the game, the program evaluates the future value of possible moves. It does this by calculating the likely board state if that move is made, using an iterative approach that looks a given distance ("ply") into the future.

A board state is assigned a value by a neural network, whose output is taken as the "worth" of the board position from the perspective of the player who had just moved.

The neural network thus defines a "strategy" for playing the game, and it is this strategy that is evolved with EP.

Example application: evolving checkers players (Fogel'02)

- Neural nets for evaluating future values of moves are evolved
- NNs have fixed structure with 5046 weights, these are evolved + one weight for "kings"
- Representation:
  - vector of 5046 real numbers for object variables (weights)
  - vector of 5046 real numbers for σ's
- Mutation:
  - Gaussian, lognormal scheme with σ-first
  - Plus special mechanism for the kings' weight
- The authors used a population size of 15, with a tournament size q = 5. When programs played against each other they scored +1, 0, -2 points for a win, draw, and loss, respectively. The 30 solutions were ranked according to their scores over the 5 games, then the best 15 became the next generation.

Example application: evolving checkers players (Fogel’02)

- Programs (with NN inside) play against other programs, no human trainer or hard-wired intelligence.
- After 840 generation (6 months!) best strategy was tested against humans via Internet.
- Program earned “expert class” ranking outperforming 99.61% of all rated players