Evolutionary Computing Summary October 2017

Evolutionary Computing (Vrije Universiteit Amsterdam)

Jantine Flendrie

Evolutionary Computing Exam October 2017

A population is a multiset of individuals, repetitions are possible. It is the basic unit of evolution: the population is evolving, not the individual. There are two population management models:

1. Generational model: Each individual survives for only 1 generation. The entire set of parents is replaced by the offspring.
2. Steady-state: A few offspring are generated per generation, and only a few members of the population are replaced.

Generation Gap = λ/μ = proportion of population replaced.
Often used: the steady-state model with $\lambda=1$.

Premature Convergence is the most common problem with EA's. This happens when a population gets stuck in a single local optimum. All multimodal problems benefit from a diverse population, otherwise no adaptation possible.

Increasing population **diversity** by **genetic operators**:

- Mutation
- Recombination

→ Individual level
→ Push towards **novelty**

Decreasing **diversity** by **selection**

- Parent selection
- Survivor selection

→ Population level
→ Push towards **quality**

Selection Pressure: The degree in which fitter solutions are more likely to survive. When selection pressure is high, fitter solutions have a much greater survival probability, and less-fit solutions are correspondingly less likely to survive.

Measurements of Selection pressure:

- **Takeover time** $\tau^{¿}$ is a measure to quantify the selection pressure. The number of generations it takes until the application of selection completely fills the population with copies of the best individual.

$$\lambda$$
$$¿$$
$$\mu$$
$$¿¿$$
$$¿$$
$$¿$$
$$\ln ¿$$
$$\tau^{¿} = \frac{\ln \lambda}{¿}$$

- Fisher's Exact test
- Kendall's Tau-b

Most common representation of genomes

1. Binary
2. Integer
3. Permutation: Arranging some objects in a certain order. Example: TSP, scheduling, 8 queens
4. Real-valued or Floating-point
5. Tree

Initialization usually done at random.

1. Parent selection

Parent selection is the selection from current generation to take part in mating. Parent selection is **usually probabilistic**.

Fitness-proportionate selection
- Can cause premature convergence
- At the end of runs: loss of selection pressure

Rank-based selection: Removes the problems of fitness-proportionate selection by basing the selection probabilities on relative rather than absolute fitness. Rank population according to fitness and then base selection probabilities on rank.
- Linear: $P(i)=\frac{(2-s)}{\mu}+\frac{2i(s-1)}{\mu(\mu-1)}$, $1<s\leq 2, i$ = rank
- Exponential: $P(i)=\frac{1-e^{-i}}{c}$, c determined according to population size, i = rank

Tournament selection: Pick k members at random then select the best of these, repeat to select more individuals.
- Could be a bottleneck especially on structured or very large populations
- Relies on external fitness function which might not exist: e.g. evolving game players
- Higher k increases selection pressure
- Picking without replacement increases selection pressure

Uniform selection: $P(i)=\frac{1}{\mu}$
- Parents are selected by uniform random distribution.
- Uniform parent selection is unbiased – every individual has the same probability to be selected
- When working with extremely large populations, over-selection can be used.

Stochastic universal sampling: same as fitness proportionate selection, both with roulette wheel.

2. Survivor selection

Elitism: Always keep best or keep N best
Genitor: Delete worst
Age-based selection
Round-robin tournament
- Compare each individual with q randomly chosen other individuals
- The μ solutions with the greatest number of wins are retained to be next generation
- Typically, q = 10

(μ,λ) -Selection: Based on set of children only, so $\lambda>\mu$, and choose best μ

μ + λ -Selection: Based on set of children and parents, and choose best μ

What to choose?
Unimodal problem:
- The higher the better so a fitness proportionate to choose parents
- Never delete good solutions (elitism, (μ+λ))

Multimodal problem:
Reduce the focus on better solutions (tournament with lower size) to avoid premature convergence.

Changing optimum:
Good solutions might not be good in the new generation (μ,λ)

3. Variation

Variation is at the individual level and pushes towards novelty**.**

Mutation is **exploitative**: optimising within a promising area. Using information. Small diversions near the parent. Introduces new information. To hit the optimum you often need a 'lucky' mutation

- Binary Representation
  * Bit flipping: Alter each gene independent with probability p_m
- Integer Representation
  * Random resetting
  * Creep mutation: add small value to each gene with probability p_m
- Permutation Representation
  * Swap mutation: Swap two randomly chosen alleles
  * Insert mutation: Pick two allele values at random and move the second to follow the first, shifting the rest. Preserves most of the order and adjacency information.
  * Scramble mutation: Pick subset of genes random. Randomly rearrange the alleles.
  * Inversion mutation: Pick two allele values at random and invert substring between them. Preserves most adjacency info, but disruptive of order.
- Real-valued Representation
  * Uniform mutation: one allele x_i' drawn randomly from (LB_i, UB_i)
  * Non-uniform mutation: see all types below
    - Mostly probabilistic. Makes small changes to value. Most common to add random deviate to each variable separately taken from N(0, σ).
- Tree Representation
  * Most common: replace randomly chosen subtree by randomly generated tree.
    This mutation has two parameters:
    - Probability p_m to choose mutation, this is advised to be very small.
    - Probability to choose an internal point as the root of the subtree to be replaced

    The size of the child can exceed the size of the parent.

Crossover is **explorative**: discovering promising areas in the search space. Gaining information. Big jump in between 2 parents. Crossover does not change the allele frequencies of the population.

- Binary or Integer Representation
  * 1-point crossover (positional bias)
  * n-point crossover (still some positional bias)
  * uniform crossover (independent of position)
- Permutation Representation
  * Order 1 crossover: Choose an arbitrary part from P1 and copy to the first child. Copy the numbers that are not in the first part, to the first child: starting right from cut point of the copied part and using the order of P2. Wrap around at the end. For second child roles revised.
  * PMX crossover: Choose an arbitrary part from P1 and copy to the first child. Starting from the first crossover point look for elements in that segment of P2 that have not been copied. For each of these i look in the offspring to see what element j has been copied in its place from P1 and place i into the position occupied j in P2. If the place occupied by j in P2 has already been filled in the offspring by an element k, put i in the position occupied by k in P2. Having dealt with the elements from the crossover segment, the remaining positions in this offspring can be filled from P2, and the second child is created analogously with the parental roles reversed.
  * Cycle crossover: Make a cycle of alleles from P1. Put the alleles of the cycle in the first child on the positions they have in the P1. Take next cycle from P2.
  * Edge recombination: List the edges which are present in both parents for each element. Then choose random element and place in child. Look to it edges, choose one with common edge, otherwise choose one with shortest list, otherwise random.

- Real-valued Representation
  * Discrete, each allele comes from one of its parents
    - N-point crossover
    - Uniform crossover
  * Intermediate, create children between parents
    - Arithmetic crossover: $z_i = \alpha x_i + (1-\alpha) y_i$
      - Uniform arithmetic crossover: α is constant
      - Single arithmetic crossover: single random gene
      - Simple arithmetic crossover: random gene, and genes after this
      - Whole arithmetic crossover: all genes
    - Blend cross over: $d_i = y_i - x_i$, if $x_i < y_i$
      Then $z_i = [x_i - \alpha d_i, y_i + \alpha d_i]$ is the range within the allele will lie
  * Multi-parent
    - Diagonal crossover for n parents (choose n-1 crossover points)
    - Arithmetic crossover for n parents (i^{th} allele in child is average of parents' i^{th} alleles. Creates centre of mass as child. Odd in genetic algorithms, but long used in evolution strategies.
- Tree Representation
  * Most common: Exchange two randomly chosen subtrees among the parents
    Again two parameters.
    - Probability p_c to choose crossover, this is advised to be very small.
    - Probability to choose an internal point within each parent as crossover point
    The size of the offspring can exceed that of the parents

Evolving Mutation Operators

Different mutation strategies may be appropriate in different stages of the evolutionary search process. Mutation step size is not set by user but coevolves with the solution. Step-sizes are included in the genome and undergo variation and selection themselves. The rationale behind this is that a new individual $\langle \bar{x}', \sigma' \rangle$ is effectively evaluated twice. Primarily, it is evaluated directly for its viability during survivor selection based on $f(\bar{x}')$. Second, it is evaluated for its ability to create good offspring. This happens indirectly: a given step size evaluates favourably if the offspring generated by using it prove viable (in the first sense). Thus, an individual $\langle \bar{x}', \sigma' \rangle$ represents both a good $\bar{x}'$ that survived selection and a good σ' that proved successful in generating this good $\bar{x}'$ from $\bar{x}$.

There is an important underlying assumption behind the idea of using varying mutation step sizes. Namely, we assume that under different circumstances different step sizes will behave differently: some will be better than others.

Self-adaptive mutation for Real-valued Representation

- **Uncorrelated mutation** with one σ. Chromosomes $\langle x_1, x_2, \ldots, x_n, \sigma \rangle$
  $\sigma' = \sigma \cdot e^{\tau \cdot N(0,1)}$ $x_i' = x_i + \sigma' N_i(0,1)$
  τ is the learning rate,
  N_i is the ith dimension of a draw from a Normal distribution of n dimensions.

- Uncorrelated mutation with n sorts of σ's. Chromosomes $\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$ The motivation behind using n step sizes is the wish to treat dimensions differently.

  $\sigma_i' = \sigma_i \cdot e^{[¿¿' \cdot N(0,1) + \tau \cdot N_i(0,1)]}$ $x_i' = x_i + \sigma_i' N_i(0,1)$
  τ' overall learning rate,

τ is the coordinate wise learning rate.

- **Correlated mutation**. Chromosomes $\langle x_1,\ldots,x_n,\sigma_1,\ldots,\sigma_n,\alpha_1,\ldots,\alpha_k\rangle$, where k=n(n-1)/2
  The rationale behind correlated mutations is to allow the ellipses to have any orientation by rotating them with a rotation (covariance) matrix C.
  Covariance matrix C is defined as:
  * $c_{ii} = \sigma_i^2$
  * $c_{ij} = 0$ if i and j not correlated
  * $c_{ij} = ½ (\sigma_i^2 - \sigma_j^2 ¿ \tan(2\alpha_{ij})$ if i and j are correlated

  Mutation Mechanism is then

$$\tau$$
$$[¿¿' \cdot N(0,1) + \tau \cdot N_i(0,1)]$$
$$\sigma_i' = \sigma_i \cdot e^{¿}$$
$$\alpha_j' = \alpha_j + \beta \cdot N(0,1)$$
$$x' = x + N(0, C')$$

C' is the covariance matrix C after the mutation of α values. **x** is the vector of all x.

4. Termination

Termination condition is checked every generation
- Reaching some fitness
- Reaching some maximum allowed number of generations
- Reaching some maximum allowed number of evaluations
- Reaching some minimum level of diversity
- Reaching some specified number of generations without fitness improvement

5. Constraint handling

FOP: Free optimisation problem
- Black box problems

CSP: Constraint satisfaction problem $\langle S, \Phi\rangle$
- 8 Queens problem
- Graph three colouring problem

COP: Constraint optimisation problem $\langle S, f, \Phi\rangle$
- Traveling Salesman problem (TSP)

CSP → FOP
All constraints are handled indirectly by optimization function: Φ is transformed in f.

CSP → COP
Some constraints handled indirectly (those transformed into f)
Some constraints handled directly (those remaining constraints in ψ)

Indirect approaches: Penalty functions

Direct approaches:

- Special representation and operators
- Repair mechanism: Knapsack (change some ones into zeros)
- Decoder functions

6. Premature Convergence

Premature convergence is getting stuck in a local optimum due to a lack of diversity.

Approaches for **preserving diversity**

1. Explicit
    a. Make similar individuals compete for resources (with their fitness)
        i. **Fitness Sharing**: Restricts the number of individuals within a given niche by "sharing" their fitness, so as to allocate individuals to niches in proportion to the niche fitness.
    b. Make similar individuals compete with each other for survival
        i. **Crowding**: Attempts to distribute individuals evenly amongst niches relies on the assumption that offspring will tend to be close to parents.
2. Implicit
    a. Impose an equivalent of **geographical separation**
        i. **Island model Parallel**: Run multiple populations in parallel. After a (usually fixed) number of generations exchange individuals with neighbours
    b. Impose an equivalent of **speciation**
        i. **Cellular EA's**: Consider each individual to exist on a point on a grid. Selection (hence recombination) and replacement happen using concept of a neighbourhood a.k.a. deme. Leads to different parts of grid searching different parts of space, good solutions diffuse across grid over a number of generations.

Parameter **Tuning**: Done before the run
Tuning effort:

- A= number of vectors tested
- B= number of tests per vector

Tuning methods can be positioned by their rationale:

- To optimize A (iterative search)
  Meta-EA, SPOT
- To optimize B (multi-stage search)
  ANOVA, Racing
- To optimize A and B (combination)
  REVAC with racing and sharpening,
  SPOT with less evals per p.v.

Parameter Control: Done during the run

1. **Determinist** parameter control: without feedback from the search
2. **Adaptive** parameter control: based on some measure monitoring search progress
3. **Self-adaptive** parameter control: parameter values evolve along with solutions; encoded onto chromosomes they undergo variation and selection. Here the evolution of evolution is used to implement the self-adaptation of parameters.

Examples:

I. Changing mutation step size
II. Changing the penalty Coefficients

What evidence Informs the change?

- **Absolute evidence**: predefined event triggers change, e.g. increase p_m by 10% if population diversity falls under threshold x. Direction and magnitude of change is fixed.
- **Relative evidence**: compare parameter values through solutions created with them, e.g. increase p_m if top quality offspring came by high mutation rates. Direction and magnitude of change is not fixed.

Summary parameter control (+ is possible, - is impossible)

	Deterministic	Adaptive	Self-adaptive
Absolute	+	+	-
Relative	-	+	+

Performance measures:
- Algorithm speed
- Solution quality
- Robustness

MBF (mean best fitness),
AES (average number of evaluations to a solution),
SR (success rate).

7. Popular algorithms

Genetic Algorithms (GA)

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent selection	Fitness proportional - implemented by Roulette Wheel
Survival selection	Generational

Evolution Strategies (ES)
The (μ+λ) and (μ, λ) ES's gave rise to the possibility of more sophisticated forms of step-size control, and led to the development of a very useful feature in evolutionary computing: self-adaptation of strategy parameters. The selective pressure in evolution strategies is very high because λ is typically much higher than μ.

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or (μ + λ)
Speciality	Self-adaptation of mutation step sizes

Evolutionary Programming (EP)

Representation	Real-valued vectors
Recombination	None
Mutation	Gaussian perturbation
Parent selection	Deterministic (each parent creates one offspring via mutation)
Survivor selection	Probabilistic (μ + μ)
Speciality	Self-adaptation of mutation step sizes (in meta-EP)

Genetic Programming (GP)
Over-selection is often used to deal with the typically large population sizes (population sizes of several thousands are not unusual in GP).

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

PSO: Particle Swarm Optimization

Representation	Real-valued vectors
Recombination	None
Mutation	Adding velocity vector
Parent selection	Deterministic (each parent creates one offspring via mutation)
Survival selection	Generational (offspring replace parents)

During an evolutionary cycle each triple $\langle \acute{x}_i , \acute{v}_i , \acute{b}_i \rangle$ is replaced by the mutant triple $\langle \acute{x}_i^{'} , \acute{v}_i^{'} , \acute{b}_i^{'} \rangle$ using the following formulas

$$\acute{x}_i^{'} = \acute{x}_i + \acute{v}_i^{'}$$

$$\acute{v}_i^{'} = w \cdot \acute{v}_i + \phi_1 U_1 \left(\acute{b}_i - \acute{x}_i \right) + \phi_2 U_2 \left(\acute{c} - \acute{x}_i \right)$$

$$\acute{b}_i^{'} = \begin{cases} \acute{x}_i^{'} \text{ if } f\left(\acute{x}_i^{'}\right) < f\left(\acute{b}_i\right) \\ \acute{b}_i \text{ otherwise} \end{cases}$$

i: Indicates which individual
$\acute{b}_i$: the personal best of the i-th population member
$\acute{c}$: the population's global best (champion)

8. Evolutionary Robotics

Evolutionary robotics is an especially challenging application area because of two additional issues that other branches of evolutionary computing do not face:
- the very weak and noisy link between controllable design details and the target feature or features;
- the great variety of conditions under which a solution should perform well. It could be argued that the behaviour should be considered as phenotype, because it is the entity that is being evaluated.

Off-line evolution: This means that an evolutionary algorithm to find a good controller is applied before the operational period of the robot.
- Evolutionary algorithm is run before robot deployment
- Best controller is chosen from the evolved population
- Controller is downloaded to the robot and remains fixed during robot operation
  - 95% of existing work

Online evolution. The alternative is to apply Online Evolution to evolve controllers during the operational period. This implies that evolutionary operators can change the robot's control software even after its deployment.
- Evolution takes place during actual robot operation
- Controllers are evolved and tested concurrently with task execution
  - New, < 40 papers

- Two objectives: fit for environment, fit for purpose

Challenges in Evolutionary Robotics:
Online and offline challenges:

1. Fitness function is very **noisy**: Noise is inherent in the physical world. For instance, two LEDs on a robot are not fully identical and the light they emit can be different even though they receive the same controller instruction.
2. Fitness function is very **costly**: The controller must be tested under different initial conditions, for instance starting at different locations of the arena and/or under various lights. Ultimately this means that many time- consuming measurements are needed to assess the fitness of a given controller.
3. Fitness function is very **complex**: A controller can only determine the actuator response on a low level (e.g., the torque on the left wheel or the colour of the LED light), whereas the fitness depends on exhibited robot behaviour on a high level (e.g., the robot is driving in circles). Hence, the phenotypes, i.e., the controllers, cannot be directly evaluated and one has to observe and assess the robot behaviour induced by the given controller.
4. There may **not be an explicit fitness** function at all: In the true spirit of biological evolution, EC can be used in an objective-free fashion to invent robots that are well suited to some (previously unknown and/or changing) environment. In such cases robots do not have a quantifiable level of fitness; they are fit if they survive long enough to be able to mate and propagate their genetic makeup.

Online challenge only:

5. The fitness landscape has "**no-go areas**": Evaluating a poor candidate solution of a traveling salesman problem can waste time. Evaluating a poor candidate solution (bad controller) can destroy the robot. When working in real hardware, such candidate solutions must be avoided during the evolutionary search.
6. Optimising quantifiable skills of the robots ↔ enabling open-ended adaptation to the given environment
7. Robots can be passive ↔ active components of the evolutionary system.
8. Robots can also influence the evolutionary dynamics implicitly by structuring the population. This influence is grounded in the physical embedding which makes a group of robots spatially structured.

Online **Encapsulated evolution**: Every robot implements a full EA internally
Each robot stores a separate population
Each robot implements evolutionary operators
Time sharing of controllers for evaluation

- Adv: Different populations can facilitate
- Dis: Inefficient use of robots

Online **Distributed evolution**: A single EA is implemented collectively by all robots
Each robot stores/uses one individual
Evolutionary operators are implemented by robot interaction

- Adv: Efficient use of robot
- Dis: Interrobot population

Online **Hybrid Evolution**: Similar to island-model EAs
Each robot stores a separate sub-population
Each robot implements evolutionary operators
Individuals migrate between subpopulations

- Adv: Best of both worlds
- Dis: No known disadvantages

Objective driven (Optimisation)
Explicit fitness: Travelled distance, Number of collected pucks
Implicit fitness: Energy level, Transferability, Novelty Search

Environment driven (Adaption)
No fitness: Goal is to survive in the environment and spread your genome while alive
Fitness is a posteriori measurement

Neuro Evolution of Augmenting Topologies (NEAT)
A typological innovation might not be good in the beginning. You can protect them through speciation:
- Similar networks are clustered (based on similarity value)
- Mating is done only in the specie
- Fitness sharing is applied to prevent that one species takes over the population

9. Multi-objective Problems (MOP's)

Wide range of problems can be categorised by the presence of a number of n possibly conflicting objectives.
Buying a car: speed vs. price vs. reliability
Robotics: Different morphology for different environments

Dominance: Solution x dominates solution y, ($x \preccurlyeq y$), if:
– x is better than y in at least one objective,
– x is not worse than y in all other objectives

Pareto Optimality
Solution x is non-dominated in a set of solutions Q if no solution from Q dominates x
- Pareto-optimal set: a set of non-dominated solutions in the solution space. Its members are called the Pareto-optimal solutions
- Pareto-optimal front: an image of the Pareto-optimal set in the objective space P

Characteristics	Single-objective optimisation	Multi-objective optimisation
Number of objectives	One single	Multiple
Spaces	Two: decision variable space and objective space	Two: decision variable space and objective space
Comparison of candidate solutions	X is better than Y	X **dominates** Y
Result	One (or several equally good ones)	Pareto optimal set
Algorithm goals	Convergence on global peak	Approximate whole pareto front

Two multi-objective optimisation approaches
Preference-based, a.k.a. aggregation-based approach: Traditional, using single objective optimisation methods. Example: weighted-sum with costs and speed of buying a car.

Real multi-objective approach: possible with novel multi-objective optimisation techniques, enabling better insight into the problem. Can return a set of trade-off solutions (approximation set) in a single run. Remembering all the non-dominated points you have seen – usually using elitism or an archive. Usually based on dominance:
Dominance rank: by how many individuals is an individual dominated
Dominance count: how many individuals does an individual dominate?

MAP-elite
1. Define behaviour characterisation with N dimensions
2. MAP-elites transforms the behaviour space into discrete bins according to a user defined granularity level

Memetic Algorithm: The combination of an EA with Local Search Operators within the EA loop. Local search operators can use instance-specific knowledge.
Why **Hybrid Evolution/hybridise**:

- No Free Lunch Theory: averaged over all problems, all algorithms perform the same.
- Thus, an algorithm is as good for a problem as the problem-dependent knowledge it incorporates.
- Local search operators be viewed as a sort of "lifetime learning"
- Want to apply the concept of memes (cultural transmission iso biological transmission)

Two Models of Lifetime Adaptation

Lamarckian

- Traits (eigenschappen) acquired by an individual during its lifetime can be transmitted to its offspring, e.g., replace individual with fitter neighbour

Baldwinian

- Traits acquired by individual cannot be transmitted to its offspring, e.g., individual receives fitness (but not genotype) of fitter neighbour

Lifetime learning seems like a good fit for robots. Evolution on the body and lifetime learning on the controller. Offspring must learn a controller for and arbitrary morphology. Probably parents have a similar morphology to offspring. Therefore, Lamarckian evolution might be more logical

Co-evolution

Fitness of one individual depends upon others. The fitness landscape is not fixed, but coupled. Adaption by one individual changes the fitness landscape for others. For example: predator-prey interactions.

→ Search gradient can be obtained only by letting individuals interact. Exact fitness may be not computable.

Competitive co-evolution: Prey's and predators compete with each other.
Cooperative co-evolution: Predators cooperate with each other to catch a prey.