Evolutionary Computation

Dirk Thierens

Universiteit Utrecht
The Netherlands

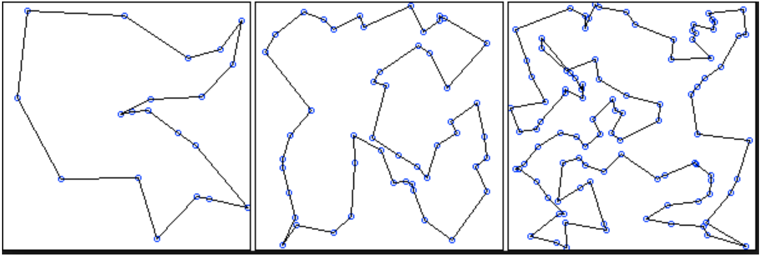
Genotype Representations

- Genotype representations need to be compatible with the recombination & mutation operators
- Specific problem-dependent examples:
  1. Permutation Representation
  2. Neural Network Representation
  3. Real-Valued Vector Representation

Permutation problems

- **Goal**
  Design suitable representations and genetic operators for permutation or sequencing problems
- **Examples**
  - scheduling
  - vehicle routing
  - queueing
  - ...

Traveling salesman problem



Find the shortest route while visiting all cities exactly once.

Permutation problems

- travelling salesman
- non-binary strings
  - p1 = 1 2 3 4 5 6 7 8
  - p2 = 4 6 2 1 7 8 5 3
  - standard crossover ⇒ illegal tours
  - c1 = $\underline{1}$ 2 $\underline{3}$ | $\underline{1}$ 7 8 5 $\underline{3}$
  - c2 = $\underline{4}$ $\underline{6}$ 2 | $\underline{4}$ 5 $\underline{6}$ 7 8
- alternative search space representation
- alternative genetic operators

Insert mutation

randomly select one element from the sequence and insert it at some other random position in the sequence

$$\begin{array}{c} A\ \ B\ \ \underline{C}\ \ D\ \ E\ \ F\ \ G\ \ H \\ \Downarrow \\ A\ \ B\ \ D\ \ E\ \ F\ \ \underline{C}\ \ G\ \ H \end{array}$$

Swap mutation

randomly select two elements from the sequence and swap their position

$$\begin{array}{c} A\ \ B\ \ \underline{C}\ \ D\ \ E\ \ F\ \ \underline{G}\ \ H \\ \Downarrow \\ A\ \ B\ \ \underline{G}\ \ D\ \ E\ \ F\ \ \underline{C}\ \ H \end{array}$$

Scramble mutation

randomly select a subsequence and scramble all elements in this subsequence

$$\begin{array}{c} A\ \ B \mid C\ \ D\ \ E\ \ F \mid G\ \ H \\ \Downarrow \\ A\ \ B \mid D\ \ F\ \ E\ \ C \mid G\ \ H \end{array}$$

very destructive $\Rightarrow$ limit length of the subsequence

Mutation operator: 2-opt

randomly select two points along the sequence and invert one of the subsequences

$$\begin{array}{c} A\;\;B\;|\;C\;\;D\;\;E\;\;F\;|\;G\;\;H \\ \Downarrow \\ A\;\;B\;|\;F\;\;E\;\;D\;\;C\;|\;G\;\;H \end{array}$$

Mutation operators

- TSP: *adjacency* of elements in permutation is important
  $\rightarrow$ 2-opt only minimal change
- scheduling: *relative ordering* of elements in permutation is important
  $\rightarrow$ 2-opt large change
  e.g.: priority queue: line of people waiting for supply of tickets for different seats on different trains

Recombination operators

- ’standard’ crossover operators generate infeasible sequences

$$\begin{array}{cccccccc} A & B & C & D & E\,| & F & G & H \\ b & f & d & h & g\,| & e & a & c \end{array}$$
$$\Downarrow$$
$$\begin{array}{cccccccc} A & B & C & D & E\,| & e & a & c \\ b & f & d & h & g\,| & F & G & H \end{array}$$

- different aspects
  - adjacency
  - relative order
  - absolute order

$\Rightarrow$ whole set of permutation crossover operators proposed !

Order crossover

p1: A B | C D E F | G H I
p2: h d | a e i c | f b g

$\Downarrow$

ch: a i C D E F b g h

1. randomly select two crosspoints
2. copy subsequence between crosspoints from p1
3. starting at 2nd crosspoint: fill in missing elements retaining relative order from p2

Partially mapped crossover

p1: A B | C D E F | G H I
p2: h d | a e i c | f b g
$\Downarrow$
ch: h i C D E F a b g

1. randomly select two crosspoints
2. copy p2 to child
3. copy elements between crosspoints from p1 to child while placing the replaced element from p2 at the location where the replacer is positioned

Position crossover

p1: <u>A</u> B <u>C</u> D <u>E</u> <u>F</u> G H <u>I</u>
p2: h d a e i c f b g

$\Downarrow$

ch: A h C d E F b g I

1. randomly mark k positions
2. copy marked elements from p1 to child
3. scan p2 from left to right and fill in missing elements

Maximal preservative crossover

$$\begin{array}{l}
\text{p1: A B } | \text{ C D E F } | \text{ G H I} \\
\text{p2: h d } | \text{ a e i c } | \text{ f b g} \\
\Downarrow \\
\text{ch: i a C D E F b g h}
\end{array}$$

1. randomly select two crosspoints
2. copy subsequence between crosspoints from p1
3. add successively an adjacent element from p2 starting at last element in child
4. if already placed: take adjacent element from p1

Cycle crossover

p1:	A	B	C	D	E	F	G	H	I
p2:	f	c	d	a	e	b	h	i	g
cy:	1	1	1	1	2	1	3	3	3

$$\Downarrow$$

ch: A B C D E F h i g

1. mark cycles
2. cross full cycles

⇒ emphasizes absolute position above adjacency or relative order

edge recombination

parent tours [ABCDEF] & [BDCAEF]

edge map:

city	*edges*
A	B F C E
B	A C D F
C	B D A
D	C E B
E	D F A
F	A E B

edge recombination algorithm:

1. choose initial city from one parent
2. remove current city from edge map
3. if current city has remaining edges
   goto step 4
   else
   goto step 5
4. choose current city edge with fewest remaining edges
5. if still remaining cities, choose one with fewest remaining cities

1. random choice $\Rightarrow$ B
2. next candidates: A C D F
   choose from C D F (same edge number) $\Rightarrow$ C
3. next candidates: A D
   (edgelist D $<$ edgelist A) $\Rightarrow$ D
4. next candidate: E $\Rightarrow$ E
5. next candidates: A F
   tie breaking $\Rightarrow$ A
6. next candidate: F $\Rightarrow$ F

resulting tour: [BCDEAF]

Fitness correlation coefficients

- genetic operators should preserve useful fitness characteristics between parents and offspring
- calculate the fitness correlation coefficient to quantify this
- k-ary operator: generate n sets of k parents
- apply operator to each set to create children
- compute fitness of all individuals
- $\{f(p_{g1}), f(p_{g2}), \ldots, f(p_{gn}\}$
- $\{f(c_{g1}), f(c_{g2}), \ldots, f(c_{gn}\}$

Fitness correlation coefficients

- F_p : mean fitness of the parents
  F_c : mean fitness of the children
  $\sigma(F_p)$ = standard deviation of fitness parents
  $\sigma(F_c)$ = standard deviation of fitness children
  $cov(F_p, F_c) = \sum_{i=1}^{n} \frac{(f(p_{gi})-F_p)(f(c_{gi})-F_c)}{n}$
  covariance between fitness parents and fitness children
- operator fitness correlation coefficient ρ_{op}:

$$\rho_{op} = \frac{cov(F_p, F_c)}{\sigma(F_p)\sigma(F_c)}$$

Traveling Salesman problem: mutation operators

- various mutation operators applicable
  - 2opt mutation ($2OPT$)
  - swap mutation ($SWAP$)
  - insert mutation (INS)

  performance: $2OPT > INS > SWAP$
- mutation fitness correlation coefficients ρ_{mutate} :

ρ_{2OPT}	0.86
ρ_{INS}	0.80
ρ_{SWAP}	0.77

Traveling Salesman problem: crossover operators

- various crossover operators in applicable
  - cycle crossover (CX)
  - partially matched crossover (PMX)
  - order crossover (OX)
  - edge crossover (EX)

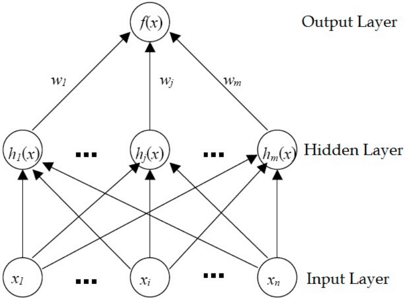
  performance: $EX > OX > PMX > CX$
- crossover correlation coefficients ρ_{cross} :

ρ_{EX}	0.90
ρ_{OX}	0.72
ρ_{PMX}	0.61
ρ_{CX}	0.57

A Non-Redundant Neural Network Representation for Genetic Recombination

- Multi-later perceptrons (MLPs) have a number of functional equivalent symmetries that make them difficult to optimize with genetic recombination operators.
- The functional mapping implemented MLPs is not unique to one specific set of weights.
- Can we represent MLPs such that the redundancy is eliminated ?

MLP genotype representation



- MLP genotype by concatenating all weights to a vector
- Mapping from input vector X to output vector Y (transfer function: hyperbolic tangent *tanh*)

$$Y = tanh(W \times tanh(V \times X))$$

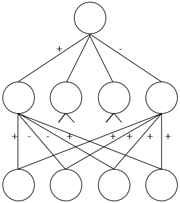
V: matrix of weights from input layer to hidden layer
W: matrix of weights from hidden layer to output layer.

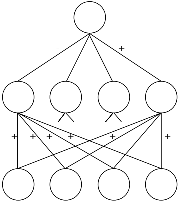
The structural-functional redundancy

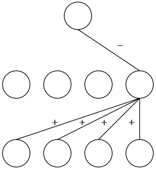
- A number of structurally different neural nets have the same input-output mapping
- These networks form a finite group of symmetries defined by two transformations.
- Any member of this group can be constructed from any other member by a sequence of these transformations.

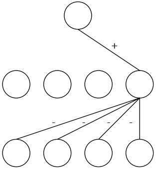
1. The first transformation is a permutation of hidden neurons. Interchanging the hidden neurons including their incoming and outgoing connection weights does not change the functional mapping of the network.
2. The second transformation is obtained by flipping the weight signs of the incoming and outgoing connection weights of a hidden neuron. Since the transfer function is an odd symmetric function this sign flipping leaves the overall network mapping unchanged.

MLP redundancies









- A network with a single hidden layer of n neurons has a total of $n!$ permutations.
- Any combination of the n hidden neurons can have their weight signs flipped, this results in 2^n networks.
- Since the two transformations are independent of each other, there are a total of $2^n n!$ structurally different but functionally identical networks.
- In (Chen, Lu, & Hecht-Nielsen, 1993) it is proven that all the functionally equivalent neural networks are compositions of hidden node permutations and sign flips.

- For the traditional local weight optimization algorithms this redundancy poses no problem since they only look in the immediate neighborhood of the current point of the search space.
- Global optimization algorithms however will try to explore the whole connection weight search space and this is a factor $2^n n!$ bigger than it really ought to be for the network to function as a universal function approximator.
- For the genetic algorithm the problem is not only one of scale but also of crossover efficiency: functional equivalent near optimal networks often give rise to totally inappropriate networks after straightforward recombination because their weight structure is only equivalent up to a certain amount of transformations.

Non-Redundant genotype coding

The functional redundancies can be eliminated if we transform each neural network to a canonical network with a unique representation in each functional equivalence class.

1. Transformation 1: Flip the weight signs of a hidden neuron whenever its bias weight is negative, so only hidden neurons with a positive bias are allowed in the non-redundant neural network representation.
2. Transformation 2: Rearrange all hidden neurons in each hidden layer such that the bias weights are sorted in ascending order.

Neural network transformation:

1. $\forall$ hidden neurons:
   if (bias < 0)
   flip signs of each node weight
2. $\forall$ hidden layer:
   sort neurons in increasing bias order

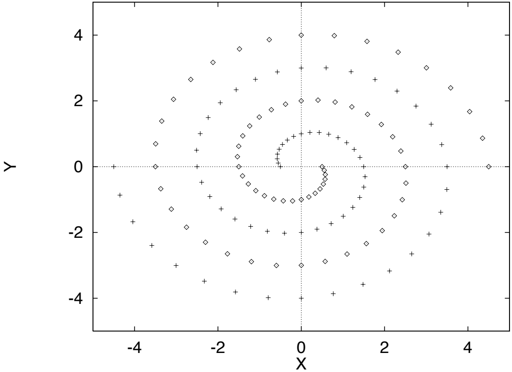
The two transformations do not interfere with each other so all the $2^n n!$ equivalent networks are transformed to a single canonical form

Crossover correlation coefficient ρ_X

- Elimination of the structural redundancies from the genotype representation ensures that the crossover operator transmits more information from the parent strings to the offspring.
- This information preservation can be quantified by comparing the crossover correlation coefficient for the redundant and non-redundant genotype coding.
- The crossover correlation coefficient is a statistical feature expressing how correlated the fitness landscape appears to the crossover operator.
- The fitness landscape is defined by the combination of the fitness function and the specific genotype coding.
- The more correlated a landscape appears to be for a specific operator the more efficient the GA search will be because the higher the correlation coefficient the more information is transmitted from the parents to the children.

Two spirals classification problem

Multi-layer perceptrons need several hidden neurons to be able to discriminate between the two spirals.



Crossover correlation coefficient ρ_X

- Two NN structures: one with 1 hidden layer of 15 neurons, another with 2 hidden layers with 15 and 5 hidden neurons.
- ρ_X computed by recombining 2500 randomly generated parent pairs for the redundant and non-redundant representation.

NNs	*redundant*	*non-redundant*
2-15-1	0.456	0.892
2-15-5-1	0.598	0.903

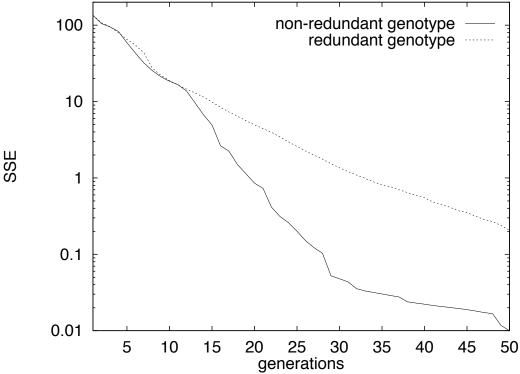
- ρ_X for the non-redundant representation is much higher $\Rightarrow$ crossover transmits more information from the parent NNs to the offspring NNs and thus will lead to more efficient GA search.

Experiment

- Hybrid genetic algorithm + backpropagation (BP) as local search
- Population of 30 neural networks
- One-point crossover
- Parents optimized by BP for 100 epochs, children optimized for 200 epochs
- Elitist family competition: best 2 of 2 parents and their 2 children survive
- Fitness is Sum-of-Squared classification error on test set

Experimental result

Non-redundant NN genotype representation leads to much more efficient search (note the log scale of the SSE of best NN in population)



Evolutionary Strategies

- Evolutionary Strategies (ES) are Evolutionary algorithms specifically developed for real-valued, semi-continuous, parameter optimization
- Key characteristic: ES use an advanced mutation operator which controls its own mutability $\rightarrow$ **self-adaptation**
- Genotype representation also includes a set of strategy parameters encoding the mutation probability distribution

ES representation

- Fitness function: $f(x_1, \ldots, x_n) : \Re^n \rightarrow \Re$
- Genotype representation of an individual solution:

$$\boxed{(x_1, ..., x_n, \sigma_1^2, ..., \sigma_n^2, c_{12}, ..., c_{n-1,n})}$$

Parameters $(x_1, \ldots, x_n)$ need to be optimized

- Individual solution consists of 3 parts:
  1. $\vec{x}$: problem variables $\Rightarrow$ Fitness $f(\vec{x})$
  2. $\vec{\sigma}$: standard deviations $\Rightarrow$ variances
  3. $\vec{\alpha}$: rotation angles $\Rightarrow$ covariances

ES representation

- The strategy parameter set $(\vec{\sigma}, \vec{\alpha})$ is part of the individual and represents the probability function for its mutation
- Strategy parameters $(\sigma_1^2, ..., \sigma_n^2, c_{12}, ..., c_{n-1,n})$ specify the n-dimensional normal distribution describing how X is mutated
- The n-dimensional normal probability density function:

$$p(X = x_1, \ldots, x_n) = \frac{\exp(-\frac{1}{2} X^T \mathbf{C}^{-1} X)}{\sqrt{(2\pi)^n |\mathbf{C}|}}$$

C: correlation matrix (c_{ij}); $|\mathbf{C}|$ determinant
$\Rightarrow$ rotation angles α_{ij} : $\tan 2\alpha_{ij} = 2c_{ij}/(\sigma_i^2 - \sigma_j^2)$

- cfr. 1-dimensional Gaussian function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{\frac{(x-\mu)^2}{2\sigma^2}}$$

ES representation

- Amount of strategy parameters decided by the user: global search reliability and robustness increases at the cost of computing time when number of strategy parameters increases
- Commonly used settings:
  1. only single standard deviation controlling the mutation of all problem parameters x_i (no correlated mutations):

$$\sigma_1 = \ldots = \sigma_n; \quad c_{ij} = 0 \; (i \neq j)$$

  2. individual standard deviations controlling the mutation of all problem parameters x_i (no correlated mutations)::

$$\sigma_1, \ldots, \sigma_n; \quad c_{ij} = 0 \; (i \neq j)$$

  3. complete covariance matrix: $\sigma_1, \ldots, \sigma_n; \quad c_{ij} \neq 0 \; (i \neq j)$

ES mutation I

1. Case 1: one single standard deviation controls the mutation of all problem parameters x_i (no correlated mutations):

$$\sigma = \sigma_1 = \ldots \sigma_n; \quad c_{ij} = 0 \ (i \neq j)$$

2. First, the strategy parameters are mutated. $\mathbb{N}(0,1)$ = a normally distributed random number (mean = 0, variance = 1):

$$\sigma' = \sigma e^{\frac{\mathbb{N}(0,1)}{\sqrt{n}}}$$

lower limit ϵ : if $\sigma' < \epsilon \Rightarrow \sigma' := \epsilon$

3. Second, problem parameters are mutated with the new strategy parameter:

$$x_i' = x_i + \sigma' \mathbb{N}_i(0,1)$$

ES mutation II

1. Case 2: individual standard deviations controlling the mutation of all problem parameters x_i (no correlated mutations)::

$$\sigma_1, \ldots, \sigma_n; \quad c_{ij} = 0 \; (i \neq j)$$

2. First, the strategy parameters are mutated:

$$\sigma_i' = \sigma_i e^{\frac{\mathbb{N}(0,1)}{\sqrt{2n}} + \frac{\mathbb{N}_i(0,1)}{\sqrt{2\sqrt{n}}}}$$

   lower limit ϵ : if $\sigma_i' < \epsilon \Rightarrow \sigma_i' := \epsilon$

3. Second, problem parameters are mutated with new strategy parameters:

$$x_i' = x_i + \sigma_i' \mathbb{N}_i(0,1)$$

ES mutation III

1. case 3: complete covariance matrix: $\sigma_1, \ldots, \sigma_n; \quad c_{ij} \neq 0 \; (i \neq j)$
2. First, the strategy parameters are mutated:

$$\sigma_i' = \sigma_i e^{\frac{\mathbb{N}(0,1)}{\sqrt{2n}} + \frac{\mathbb{N}_i(0,1)}{\sqrt{2\sqrt{n}}}}$$

$$\alpha_j' = \alpha_j + \beta \mathbb{N}_j(0,1)$$

   $\beta \approx 0.0873$ (5^o in radians), $\mathbb{N}(0,1)$: standard normal distribution
3. Second, problem parameters are mutated with new strategy parameters:

$$\vec{x}' = \vec{x} + \vec{N}(\vec{0}, \vec{\sigma}', \vec{\alpha}')$$

   $\vec{N}$: n-dimensional normal distribution

ES recombination

- Creates one offspring from several parents that are selected at random from the parent population
- Problem parameters and strategy parameters are differently recombined:
  1. *problem parameters:* select at random 2 parents of the μ parents for each parameter x_i and take their average

$$x_i^{offspring} = \frac{1}{2}\,(x_i^{parent_1^i} + x_i^{parent_2^i})$$

  2. *standard deviations:* select at random 2 parents of the μ parents and take at random one of the two parent values

$$\sigma_i^{offspring} = \sigma_i^{parent_1} \quad \text{or} \quad \sigma_i^{parent_2}$$

  3. *rotation angles:* not recombined

ES selection

- ES applies a high selection pressure: from μ parents λ offspring are generated with $\lambda >> \mu$ (typically, $\lambda \approx 5$ to 10 times μ)
- Common 'standard' values: $\mu = 15, \lambda = 100$
- The best μ solutions of the λ offspring are selected for the next generation - this is, (μ, λ)-selection - or, the best μ solutions of the μ parents and the λ offspring are selected for the next generation - this is, $(\mu + \lambda)$-selection
- Experimental results: self-adaptation works better with (μ, λ) selection

Self-adaptation: necessary conditions

Necessary conditions found by experiments to let self-adaptation work well:

- Generation of a surplus offspring: $\lambda > \mu$
- (μ, λ)-selection to guarantee extinction of misadapted individuals (as opposed to $(\mu + \lambda)$
- Intermediate selective pressure, eg. $(\mu, \lambda) = (15, 100)$
- Multiple parents necessary: $\mu > 1$
- Recombination also applied on strategy parameters (more specifically the use of intermediate recombination)