Evolutionary Computation: Evolutionary Strategies and Genetic Programming

GA's Quick Recap

- Typically applied to:
  - discrete optimization

- Characteristics:
  - not too fast
  - good heuristic for combinatorial optimization problems
  - many variants

Evolution Strategies (ES)

Slides adapted from slides Chapter 4 AE Eiben and JE Smith, Introduction to Evolutionary Computing, include information from literature and other online material, e.g. by Thomas Baeck

Evolution Strategies

- Typically applied to numerical optimisation problems
- Characteristics:
  - fast
  - good optimizer for real-valued optimisation problems
  - self-adaptation of (mutation) parameters standard

$(\mu/\rho, \lambda)$ and $(\mu/\rho + \lambda)$ notation

$$\rho \le \mu, \mu \le \lambda$$

An ES is an iterative (generational) procedure. At each generation new individuals (offspring) are created from existing individuals (parents)

- Parent population contains μ individuals
- ρ out of μ individuals are selected
- λ offspring are generated at each iteration
- In $(\mu/\rho + \lambda)$ the μ best of $\mu + \lambda$ are chosen
- In $(\mu/\rho, \lambda)$ the μ best of λ are chosen
- When $\rho = \mu$ then ρ can be omitted

An historical real-life example: the jet nozzle experiment

Task: optimize the shape of a jet nozzle (a pipe or tube of varying cross sectional area, can be used to direct or modify the flow of a fluid)
Approach: random mutations to shape + selection

Initial shape

Final shape

http://ls11-www.cs.tu-dortmund.de/people/schwefel/EADemos/

Introductory example: (1+1)-ES

Task: minimise a real-value function $f\colon\ \mathbb{R}^n \rightarrow \mathbb{R}^n$

- t=0
- Create initial point $x^t = (x_1^t, \dots, x_n^t)$
- REPEAT UNTIL (*TERMIN.COND* satisfied)
  - Draw z_i from a Normal distribution $N(0,1)$, for all $i = 1, \dots, n$
  - $y_i^t = x_i^t + \sigma \cdot z_i$
  - IF $f(x^t) < f(y^t)$ THEN $x^{t+1} = x^t$ ELSE $x^{t+1} = y^t$
  - t = t+1

Introductory example: mutation mechanism

- σ is called mutation step size
- This mimics the evolutionary process where small changes occur more often than larger ones

Introductory example: mutation mechanism

- Although σ may be kept constant, usually it is periodically modified – a procedure called self-adaptation
- A popular heuristic is the "1/5 success rule"
- Demonstrated to hold for two objective functions: the corridor and the sphere one

$$f(x_1, \dots, x_n) = c \cdot x_1 \qquad -b \leq x_2, \dots, x_n \leq b$$

$$f(x_1, \dots, x_n) = \sum_{i=1}^{n} x_i^2$$

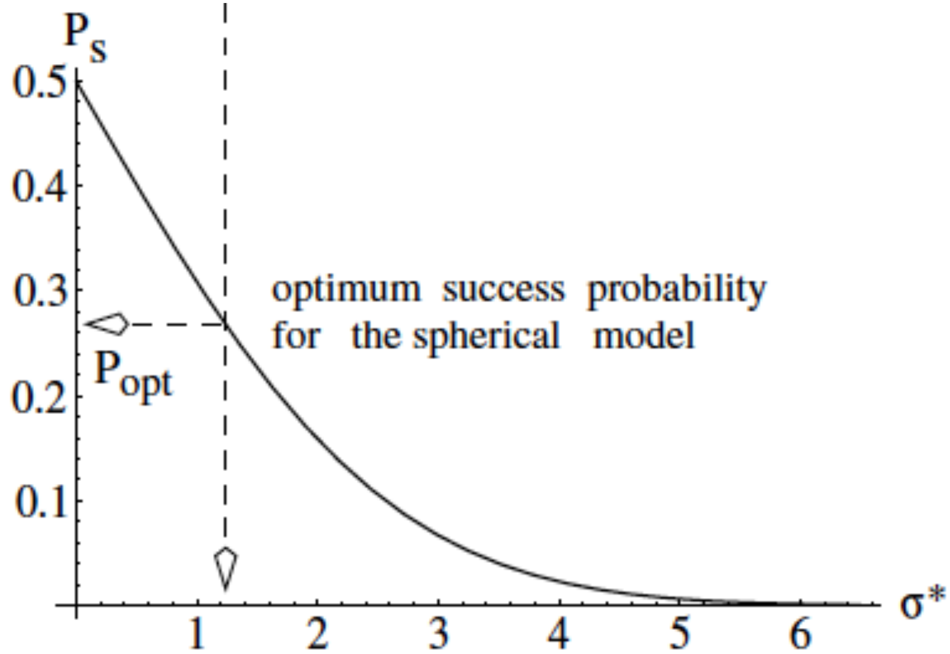
Self-adaptation of (1+1)-ES: the “1/5 success rule”

- For these models, the success probability p_s (the probability that an offspring is better than the parent, $P(f(mutate(x)) < f(x))$) should be about 1/5

- In order to obtain nearly optimal (local) performance of the (1+1)-ES in real-valued search spaces, tune the mutation strength in such a way that the (measured) success rate is about 1/5.

Self-adaptation of (1+1)-ES: the "1/5 success rule"

- Due to the monotonicity of Ps with respect to σ^*, the normalized σ, the tuning of σ can be done as follows:



1/5 success rule: If Ps < 1/5 the mutation strength must be reduced, whereas in the opposite case Ps > 1/5, σ must be increased.

Beyer, Hans-Georg, and Hans-Paul Schwefel. "Evolution strategies–a comprehensive introduction." *Natural computing* 1.1 (2002): 3-52.

The simple (1+1)-ES with self-adaptation

- t=0
- $\sigma(t) = \sigma_0$
- Create initial point $x^t = (x_1^t, \ldots, x_n^t)$
- REPEAT UNTIL (*TERMIN.COND* satisfied)
  - Draw z_i from a Normal distribution $N(0,1)$, for all $i = 1, \ldots, n$
  - $y_i^t = x_i^t + \sigma(t) \cdot z_i$
  - IF $f(x^t) < f(y^t)$ THEN $x^{t+1} = x^t$ ELSE $x^{t+1} = y^t$
  - t = t+1
  - IF (t mod n) = 0 THEN $\sigma(t) = \begin{cases} \frac{\sigma(t-n)}{c} \; if \; p_s > \frac{1}{5} \\ \sigma(t-n) \cdot c \; if \; p_s < \frac{1}{5} \\ \sigma(t-n) \; if \; p_s = \frac{1}{5} \end{cases}$

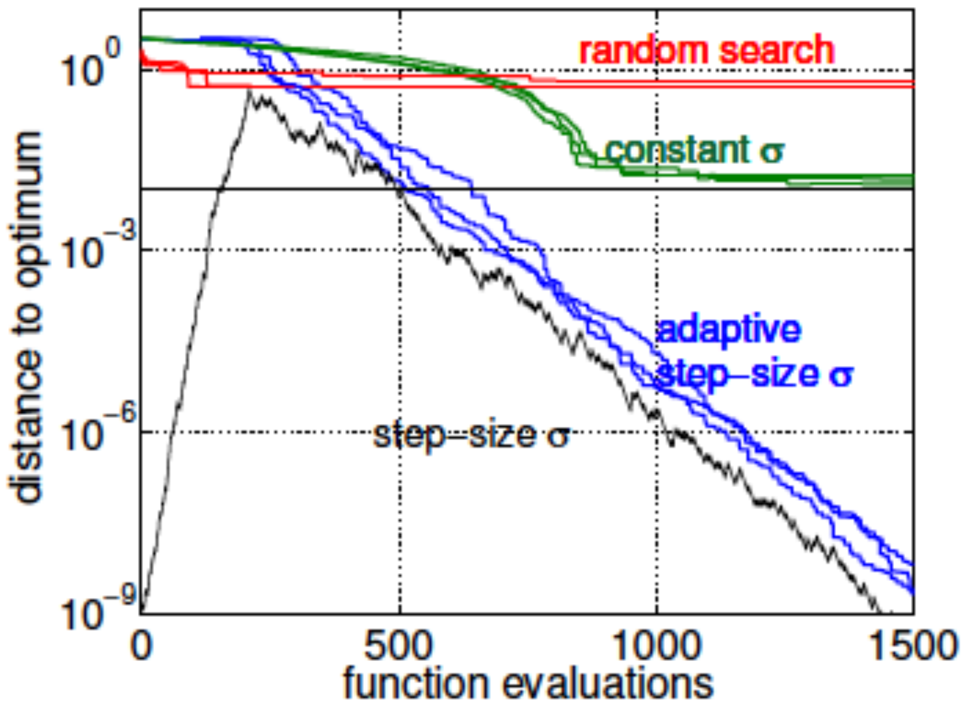
  ELSE $\sigma(t) = \sigma(t-1)$

- p_s estimated as relative frequency of successful mutations over intervals of G=$10 \cdot n$ trials. For large $n > 30,\ 0.817 \leq c < 1$ was recommended by Schwefel

Schwefel H-P (1975) Evolutionsstrategie und numerische Optimierung. Dissertation, TU, Berlin, Germany)

Example: (1+1)-ES with 1/5th success rule

- Runs of the (1+1)-ES with constant step-size (in green), of pure random search (in red), and (1+1)-ES with 1/5th success rule (in blue), on a spherical function $f(x) = ||x||^{\alpha}, \alpha > 0$. Step size evolution (in black).



http://www.cmap.polytechnique.fr/~nikolaus.hansen/es-overview-2015.pdf

Mutation parameter control

- The goal of parameter control is to drive the endogenous strategy parameters close to their optimal values
- These optimal values, can significantly change over time or depending on the position in search space

Self-adaptation principle

- Basic idea: strategy parameters (σ in the previous (1+1)-ES algorithm) undergo an evolutionary process themselves
- Nature inspired idea: in some organisms self-repair mechanisms exist, such as *repair enzymes* and *mutator genes*
- Claim of ES: self-adaptation of strategy parameters works

Mutation

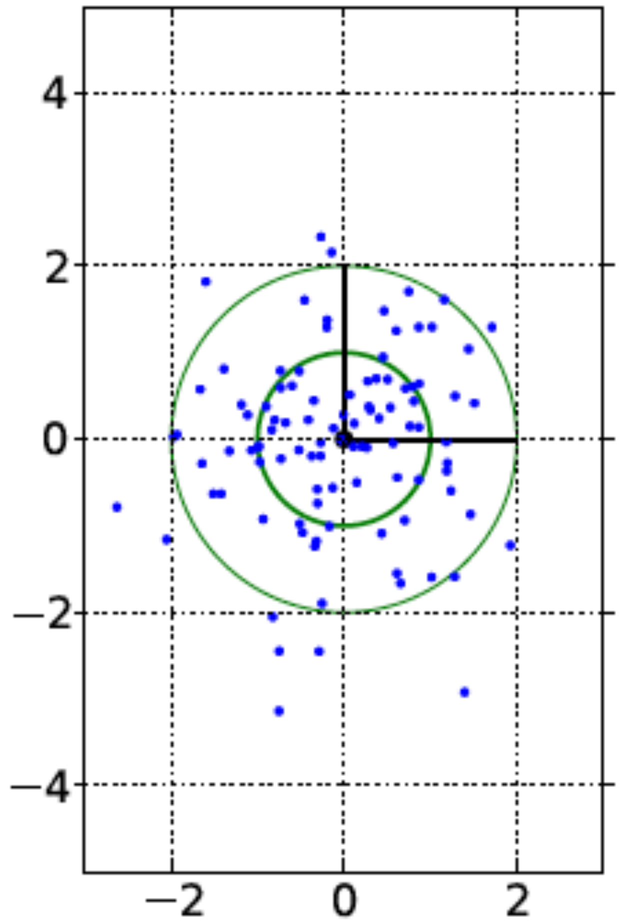
- The mutation operator introduces *small* variations by adding a point symmetric perturbation to the result of recombination. This perturbation is drawn from a multivariate normal distribution with 0 mean and covariance C
- Mutation: $\boldsymbol{x}^{new} = \boldsymbol{x}^{old} + \boldsymbol{z}$. $\quad \boldsymbol{z} \sim N(0, C)$
- ES evolves $(\boldsymbol{x}, C)$, that is, C evolves along with $\boldsymbol{x}$
- Based on multivariate normal distributions, three different mutation operators can be distinguished

Spherical/isotropic mutation

- $C = \sigma^2 I$
- C proportional to the identity

$$\boldsymbol{x}^{new} = \boldsymbol{x}^{old} + \boldsymbol{z},$$
$$\boldsymbol{z} \sim \sigma N(0, I)$$

- uncorrelated mutations, global step size, only one parameter σ

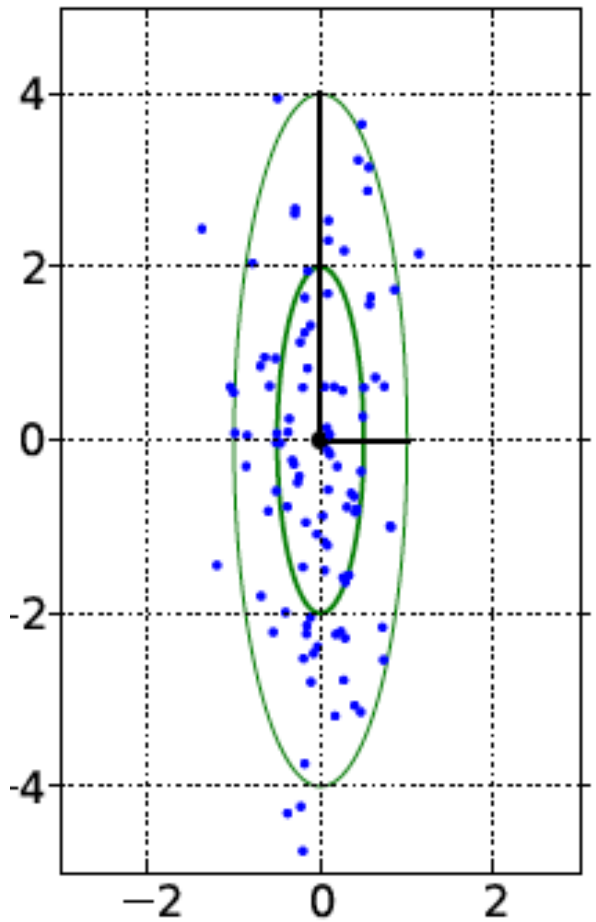


Diagonal matrix: $C = \sigma^2 I$

http://www.cmap.polytechnique.fr/~nikolaus.hansen/es-overview-2015.pdf

Axis-parallel mutation

- $C = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_n^2 \end{pmatrix}$
- C is a diagonal matrix,
  $\boldsymbol{x}^{new} = \boldsymbol{x}^{old} + \boldsymbol{z},$
  $\boldsymbol{z} \sim N(0, diag(\boldsymbol{\sigma})^2)$
- uncorrelated mutations, local step size, n parameters $\sigma_1, \ldots, \sigma_n$



diagonal matrix $C = \begin{pmatrix} 1/4 & 0 \\ 0 & 4 \end{pmatrix}$
(axis-parallel distribution)

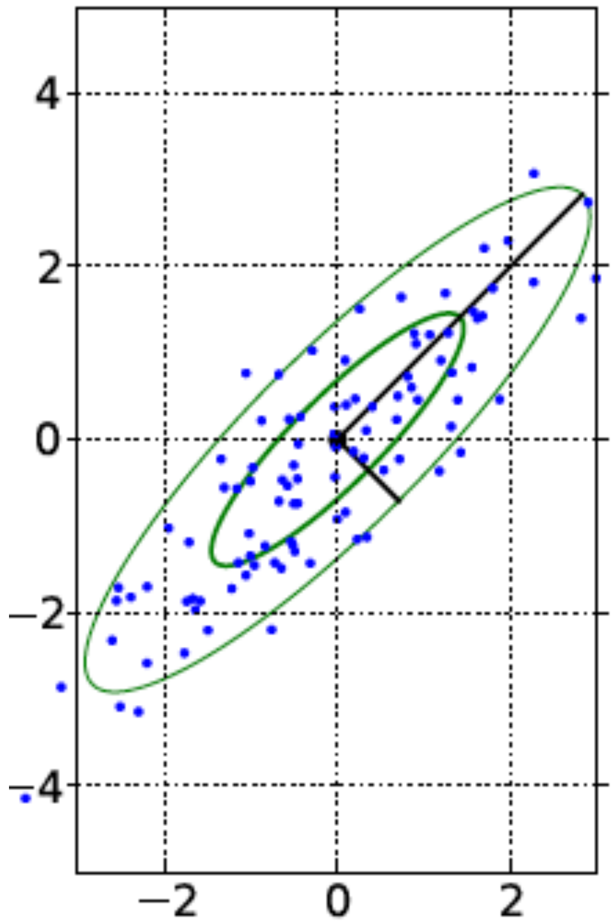
http://www.cmap.polytechnique.fr/~nikolaus.hansen/es-overview-2015.pdf

Mutation: General

- General:
- C includes non-diagonal elements,

$$\boldsymbol{x}^{new} = \boldsymbol{x}^{old} + \boldsymbol{z},$$
$$\boldsymbol{z} \sim N(0, C)$$

- correlated mutations, $(n^2 + n)/2$ parameters c_{ij}
- The general case includes the previous axis-parallel and spherical cases.



matrix C = $\begin{pmatrix} 2.125 & 1.875 \\ 1.875 & 2.125 \end{pmatrix}$

http://www.cmap.polytechnique.fr/~nikolaus.hansen/es-overview-2015.pdf

Mutation parameter control

- Controlling the parameters of the mutation operator is key to the design of evolution strategies
- For isotropic mutation, where the stepsize is a scaling factor for the random vector perturbation, the step-size controls to a large extent the convergence speed
- In situations where larger stepsizes lead to larger expected improvements, a stepsize control technique should aim at increasing the step-size (and decreasing it in the opposite scenario).

A self-adaptation ES

Algorithm 2 The $(\mu/\mu, \lambda)$-σSA-ES

given $n \in \mathbb{N}_+$, $\lambda \geq 5n$, $\mu \approx \lambda/4 \in \mathbb{N}$, $\tau \approx 1/\sqrt{n}$, $\tau_{\mathrm{i}} \approx 1/n^{1/4}$

initialize $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\sigma} \in \mathbb{R}^n_+$

while not happy

  for $k \in \{1, \ldots, \lambda\}$

    // random numbers i.i.d. for all k

    $\xi_k = \tau \mathcal{N}(0, 1)$ // global step-size

    $\boldsymbol{\xi}_k = \tau_{\mathrm{i}} \mathcal{N}(\mathbf{0}, \mathbf{I})$ // coordinate-wise $\boldsymbol{\sigma}$

    $\boldsymbol{z}_k = \mathcal{N}(\mathbf{0}, \mathbf{I})$ //$\boldsymbol{x}$-vector change

    // mutation

    $\boldsymbol{\sigma}_k = \boldsymbol{\sigma} \circ \exp(\boldsymbol{\xi}_k) \times \exp(\xi_k)$

    $\boldsymbol{x}_k = \boldsymbol{x} + \boldsymbol{\sigma}_k \circ \boldsymbol{z}_k$

  $\mathcal{P} = \mathsf{sel\_}\mu\mathsf{\_best}(\{(\boldsymbol{x}_k, \boldsymbol{\sigma}_k, f(\boldsymbol{x}_k)) \,|\, 1 \leq k \leq \lambda\})$

  // recombination

  $\boldsymbol{\sigma} = \frac{1}{\mu} \sum_{\boldsymbol{\sigma}_k \in \mathcal{P}} \boldsymbol{\sigma}_k$

  $\boldsymbol{x} = \frac{1}{\mu} \sum_{\boldsymbol{x}_k \in \mathcal{P}} \boldsymbol{x}_k$

Recombination

- Creates one child
- Acts per variable / position by either
  - Averaging parental values, or
  - Selecting one of the parental values
- From two or more parents by either:
  - Using two selected parents to make a child
  - Selecting two parents for each position anew

Parent selection

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Thus: ES parent selection is unbiased - every individual has the same probability to be selected
- Note that in ES “parent” means a population member (in GA’s: a population member selected to undergo variation)

Survivor selection

- Applied after creating λ children from the μ parents by mutation and recombination
- Deterministically chops off the "bad stuff"
- Basis of selection is either:
  - The set of children only: (μ,λ)-selection
  - The set of parents and children: (μ+λ)-selection

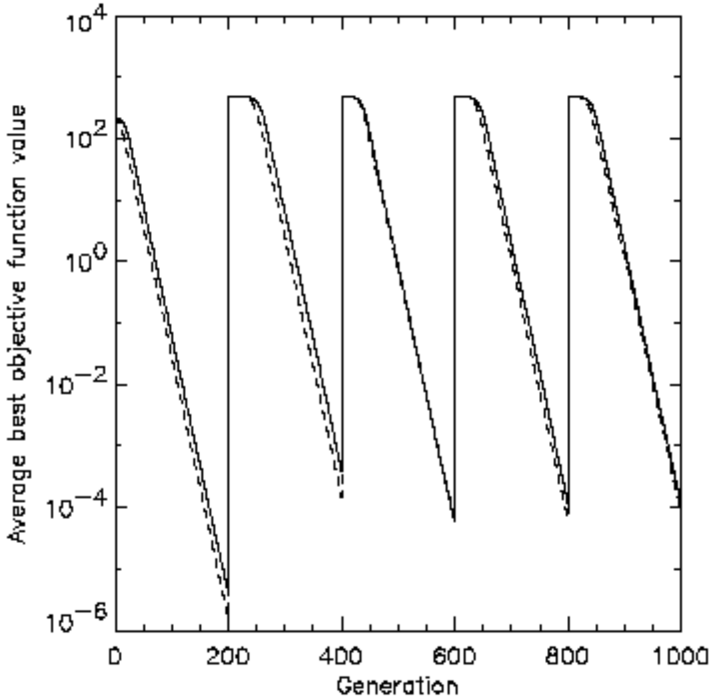
Survivor selection cont'd

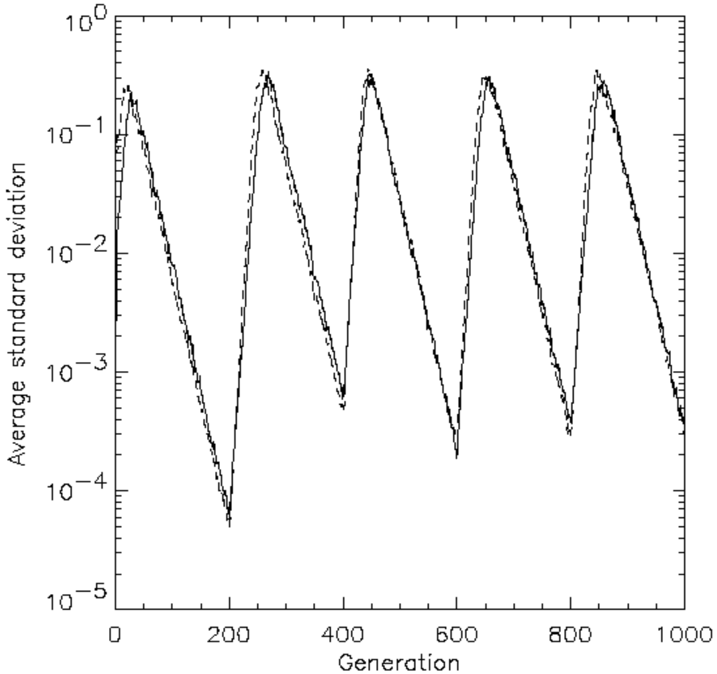
- $(\mu+\lambda)$-selection is an elitist strategy
- (μ,λ)-selection can "forget"
- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
  - Better in following moving optima
  - Using the + strategy bad σ values can survive in $\langle x,\sigma\rangle$ too long if their host x is very fit
- Selective pressure in ES is very high ($\lambda \approx 7 \cdot \mu$ is the common setting)

Self-adaptation illustrated

- Given a dynamically changing fitness landscape (optimum location shifted every 200 generations)
- Self-adaptive ES is able to
  - follow the optimum and
  - adjust the mutation step size after every shift !

Self-adaptation illustrated cont’d





Changes in the fitness values (left) and the mutation step sizes (right)

Prerequisites for self-adaptation

- $\mu > 1$ to carry different strategies
- $\lambda > \mu$ to generate offspring surplus
- Not “too” strong selection, e.g., $\lambda \approx 7 \cdot \mu$
- (μ,λ)-selection to get rid of mis-adapted σ‘s
- Mixing strategy parameters by (intermediary) recombination on them

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

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

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

- Evolution strategy:
  - Representation:
    - $-30 < x_i < 30$ (coincidence of 30’s!)
    - 30 step sizes
  - (30,200) selection
  - Termination : after 200000 fitness evaluations
  - Results: average best solution is $7.48 \cdot 10^{-8}$ (very good)

ES technical summary tableau

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$
Specialty	Self-adaptation of mutation step sizes

Genetic Programming

Slides adapted from Chapter 6

Book by Eiben, Smith

GP quick overview

- Typically applied to:
  - machine learning tasks (symbolic regression, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - Structured representation: trees, graphs
  - mutation possible but not necessary (disputed!)

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed, generated from historical data

ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

Introductory example: credit scoring

- A possible model:

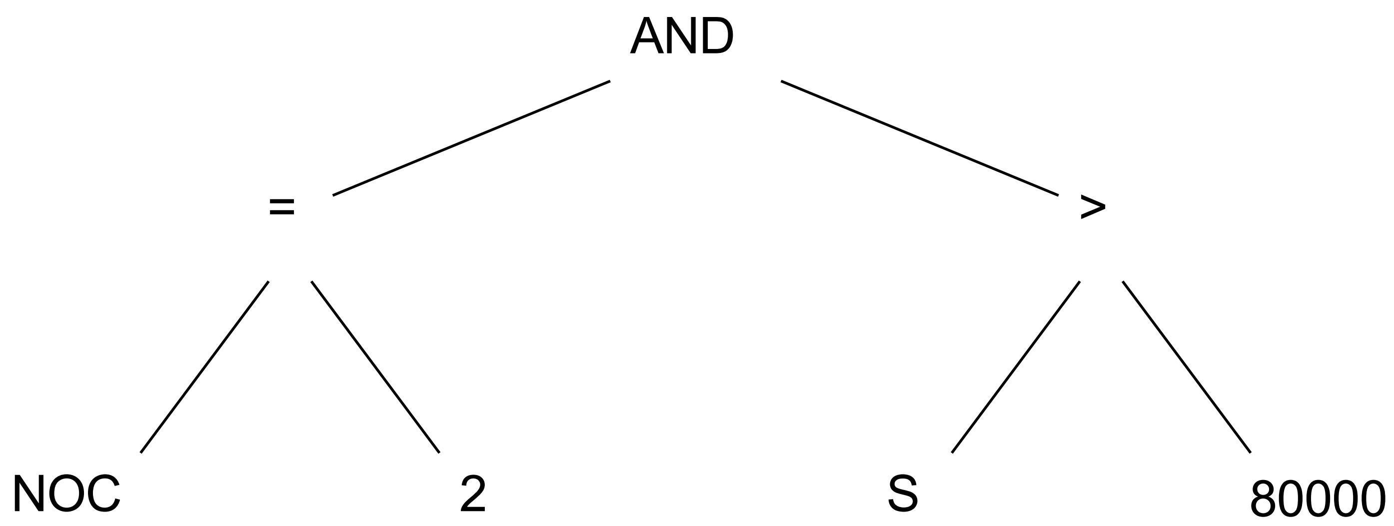
  IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
- In general:

  IF formula THEN good ELSE bad
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

can be represented by the following tree



Tree based representation

- Trees are a general representation form for, e.g.

- arithmetic formulas: $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$

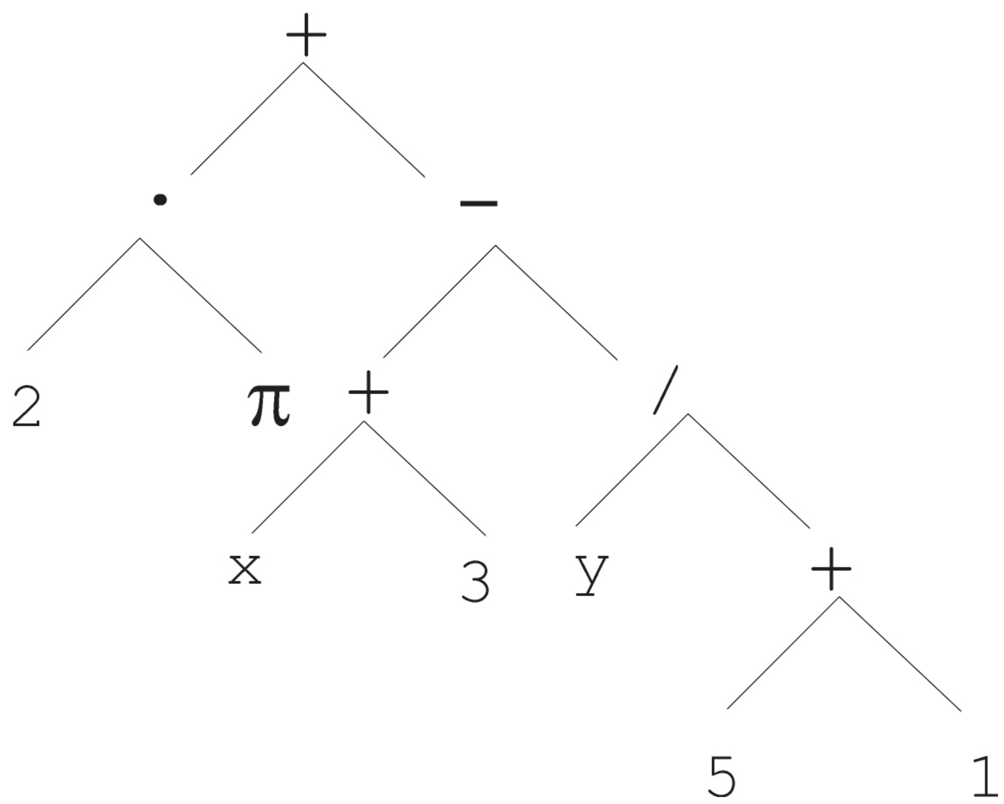
- logical formulas: $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

- programs:

```
i =1;
while (i < 20)
{
        i = i +1
}
```

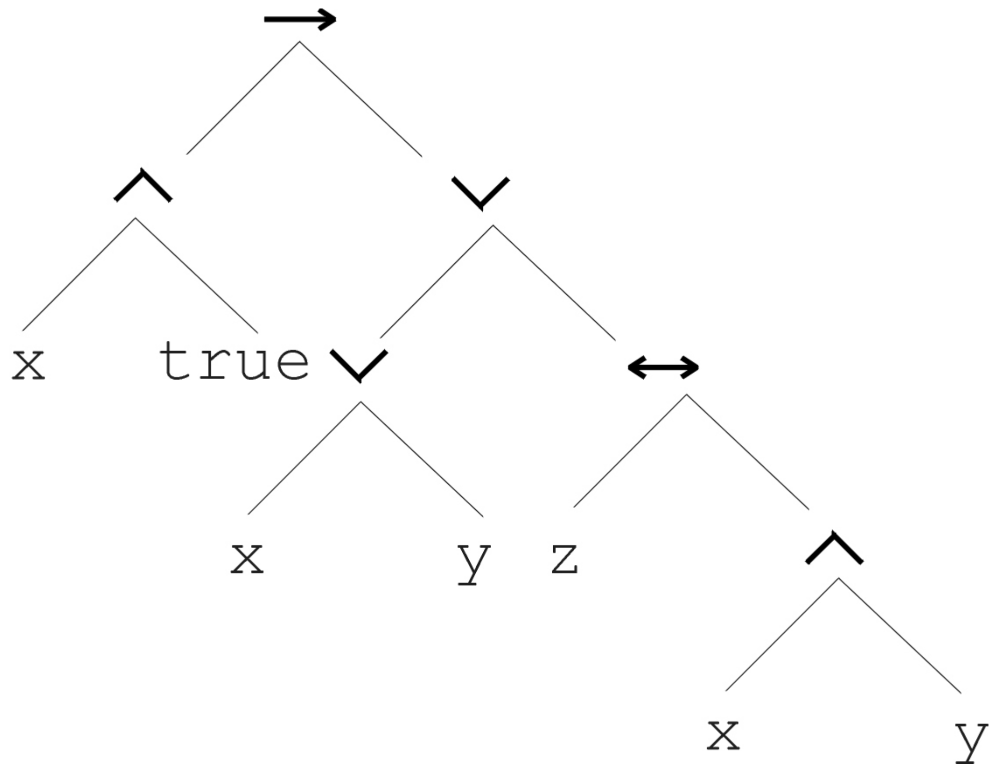
Tree based representation

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$



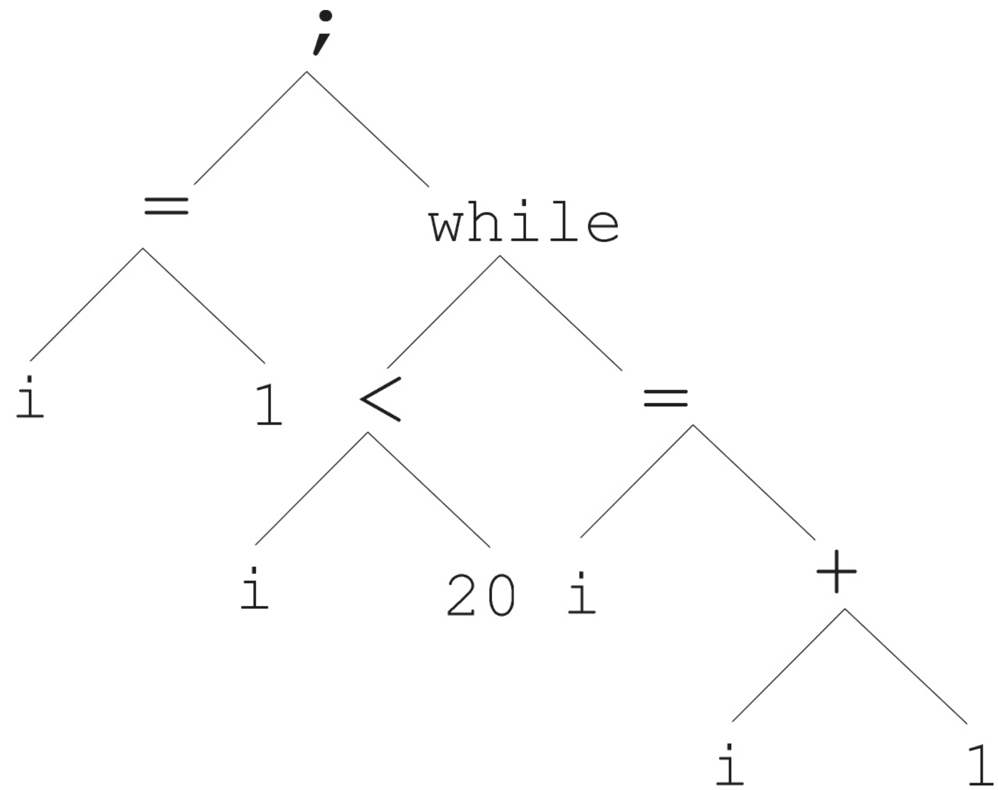
Tree based representation

$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$



Tree based representation

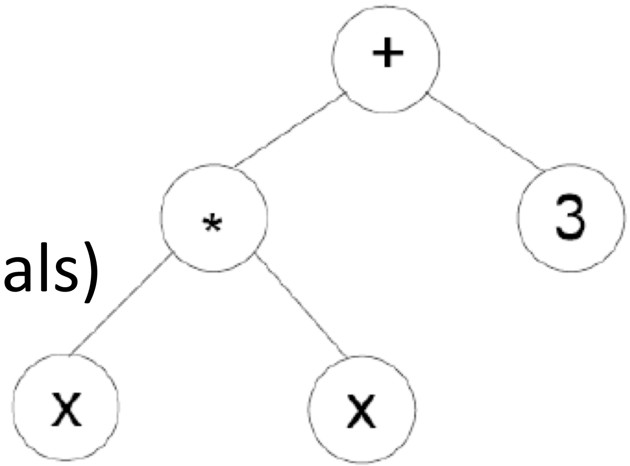
```
i =1;
while (i < 20)
{
        i = i +1
}
```



Koza’s evolution of LISP programs

- Lisp is a functional language: f (x; y) is written as (f x y)
- 10 - (3+4) is written as (-10 (+ 3 4))
- Lisp programs can be represented as trees

f (x) = x^2 + 3 ➡ (+ (* x x) 3)



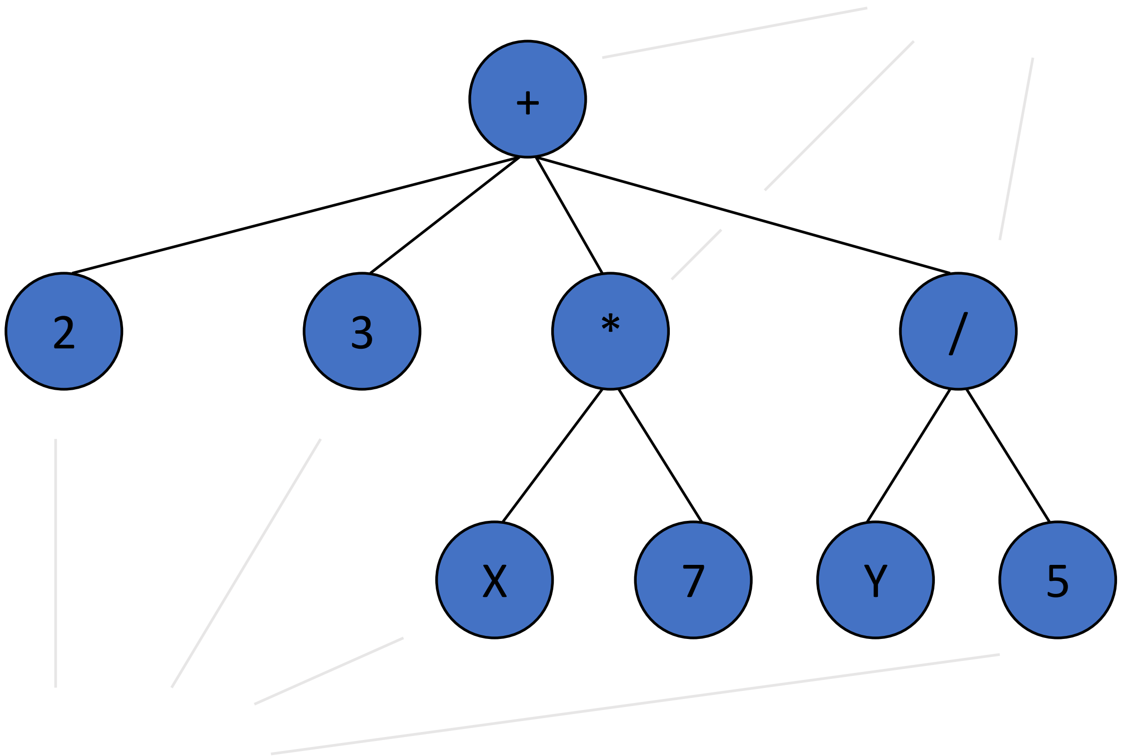
- + and * are function symbols (non-terminals)
- x and 3 are terminals

- Goal of GP’s: evolve programs given a bag of terminals and non-terminals

(Peter Seibel: Practical Common Lisp, 2004)

Tree Representation

LISP S-expression (+ 2 3 (* X 7) (/ Y 5))



Tree based representation

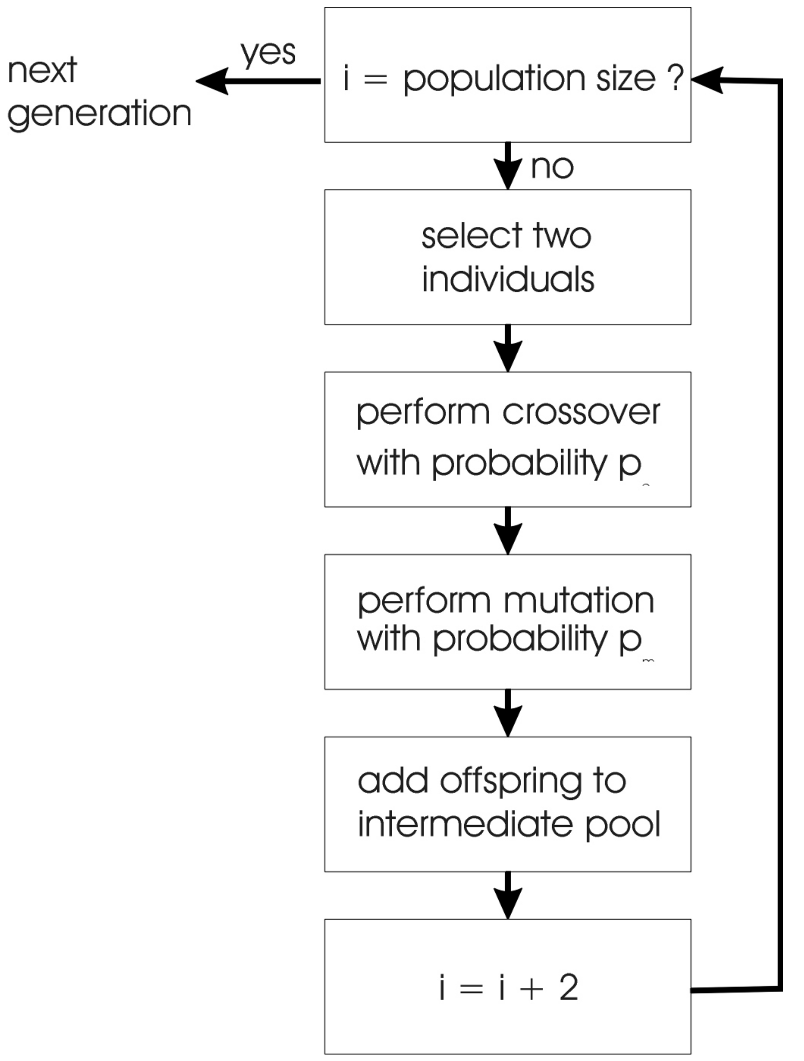
- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  1. Every $t \in T$ is a correct expression
  2. $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  3. There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure assumption: any $f \in F$ can take any $g \in F$ as argument)

Representation: GA, ES vs GP

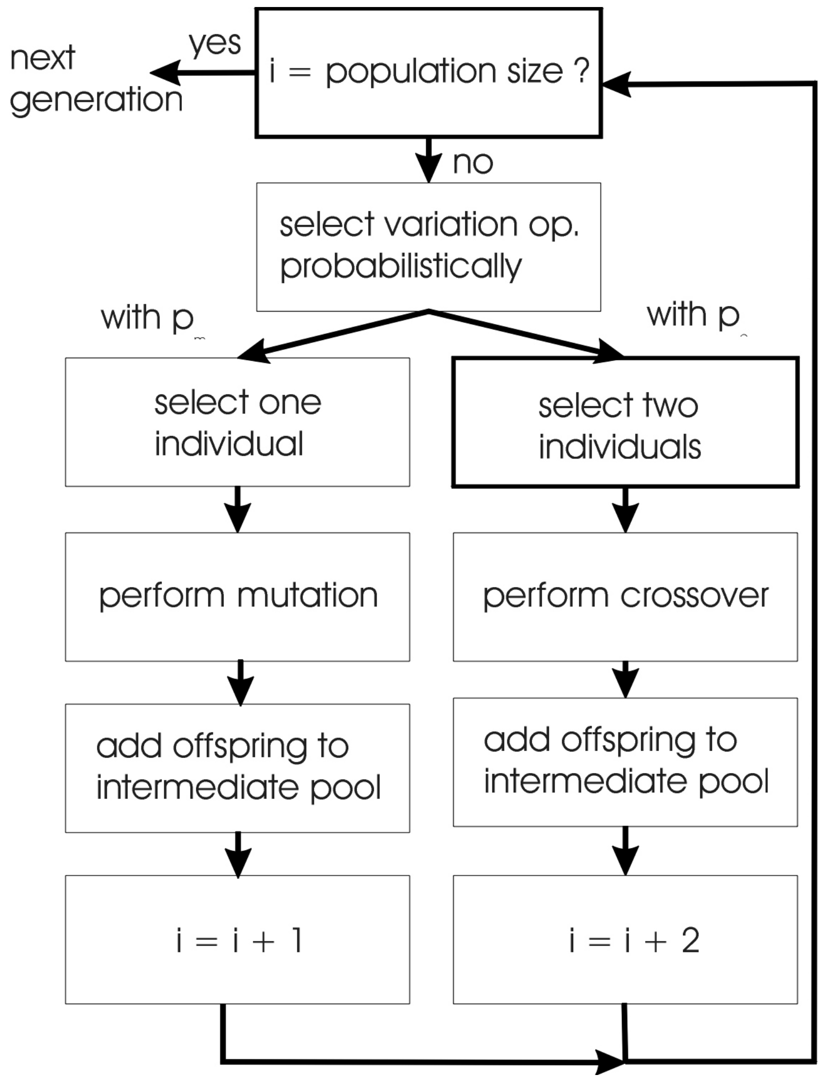
- In GA and ES chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- In GP chromosomes are tree shaped (non-linear structure)
- In GA and ES the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

Offspring creation: GA vs GP

- GA uses crossover AND mutation sequentially (be it probabilistically)
- GP uses crossover OR mutation (chosen probabilistically)



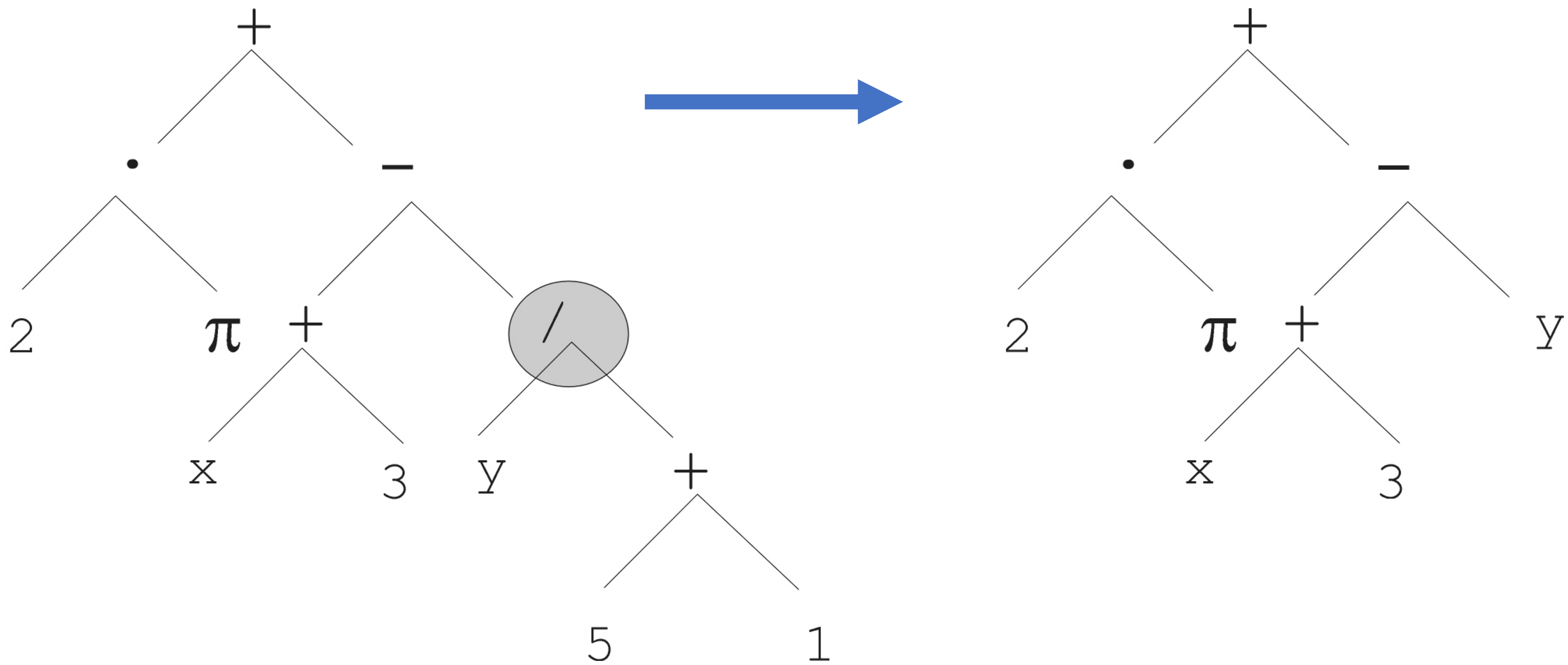
GA offspring pool creation flowchart



GP offspring pool creation flowchart

Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree

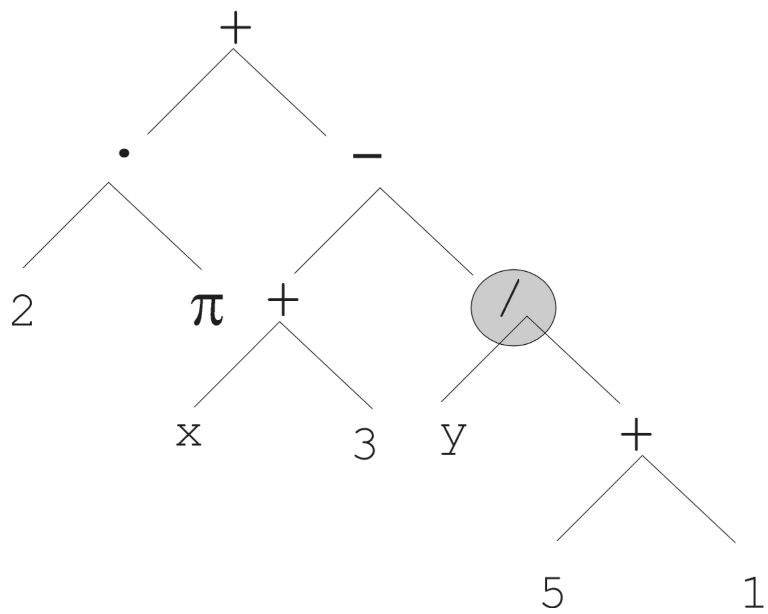


Mutation cont'd

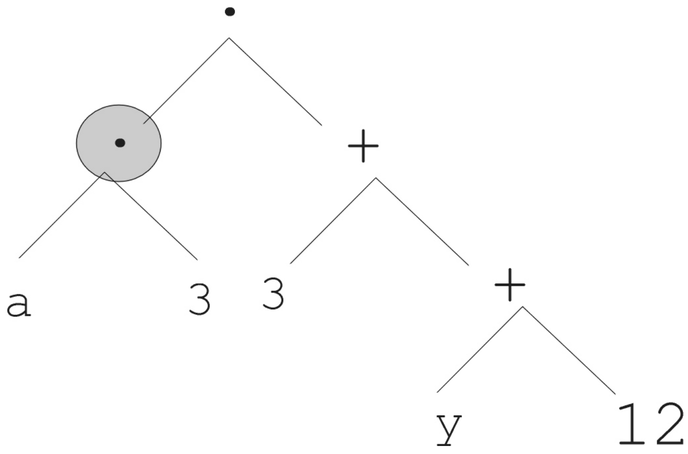
- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Recombination

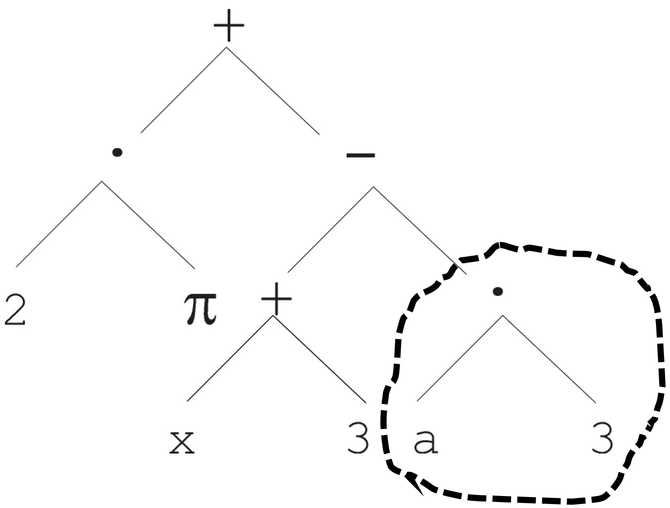
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents



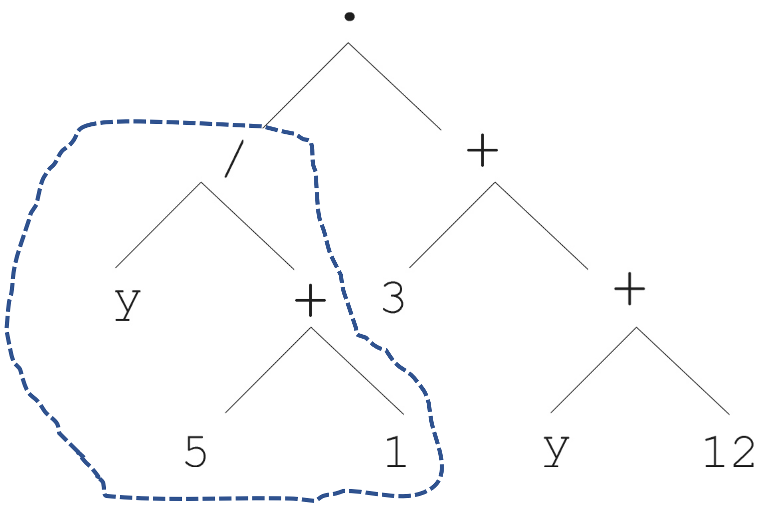
Parent 1



Parent 2



Child 1



Child 2

Selection

- Parent selection is typically fitness proportionate or tournament
- Survivor selection:
  - Typical: generational scheme (thus none)
  - Recently steady-state also popular for its elitism

Initialization

- Maximum initial depth of trees D_{max} is set
- Full method (each branch has depth = D_{max}):
  - nodes at depth $d < D_{max}$ randomly chosen from function set F
  - nodes at depth $d = D_{max}$ randomly chosen from terminal set T
- Grow method (each branch has depth $\leq D_{max}$):
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - nodes at depth $d = D_{max}$ randomly chosen from T
- Common GP initialisation: ramped half-and-half, where grow & full method each deliver half of initial population

Issue: Bloat

- Bloat = “survival of the fattest”, i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver “too big” children
  - Parsimony pressure: penalty for being oversized

R Poli, W B Langdon, N F McPhee (2008) A Field Guide to Genetic Programming.

Example application: symbolic regression

- Given some points in $\mathbf{R}^2$, $(x_1, y_1), \ldots, (x_n, y_n)$
- Find function f(x) s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- Possible GP algorithm:
  - Representation by F = {+, -, /, sin, cos}, $T = \mathbf{R} \cup \{x\}$
  - Fitness is the error, e.g. $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - All operators standard
  - pop. size = 1000, ramped half-half initialisation
  - Termination: n "hits" or 50000 fitness evaluations reached (where "hit" is if $| f(x_i) - y_i | < 0.0001$)

Practical guidelines

- Apply a two-stage process:
  - Use GP to discover a selected subset of terminals and functions from a more general set
  - Use GP to determine an appropriate data model
- Study your populations: analyze mean and variance of fitness, trees depth, size, code used, run time, ... and correlations among these
- Runs can be very long: consider checkpoint results
- Control bloat
- Encourage diversity and save good candidates

When to use GP

- For machine learning tasks like
  - protein structure prediction
  - symbolic regression
  - feature selection/importance especially in bioinformatics applications
  - classification
- For “black art" problems involving synthesis of topology and sizing of systems, like analog circuits

Example: TPOT

- Automate pipeline creation and algorithm selection for machine learning.

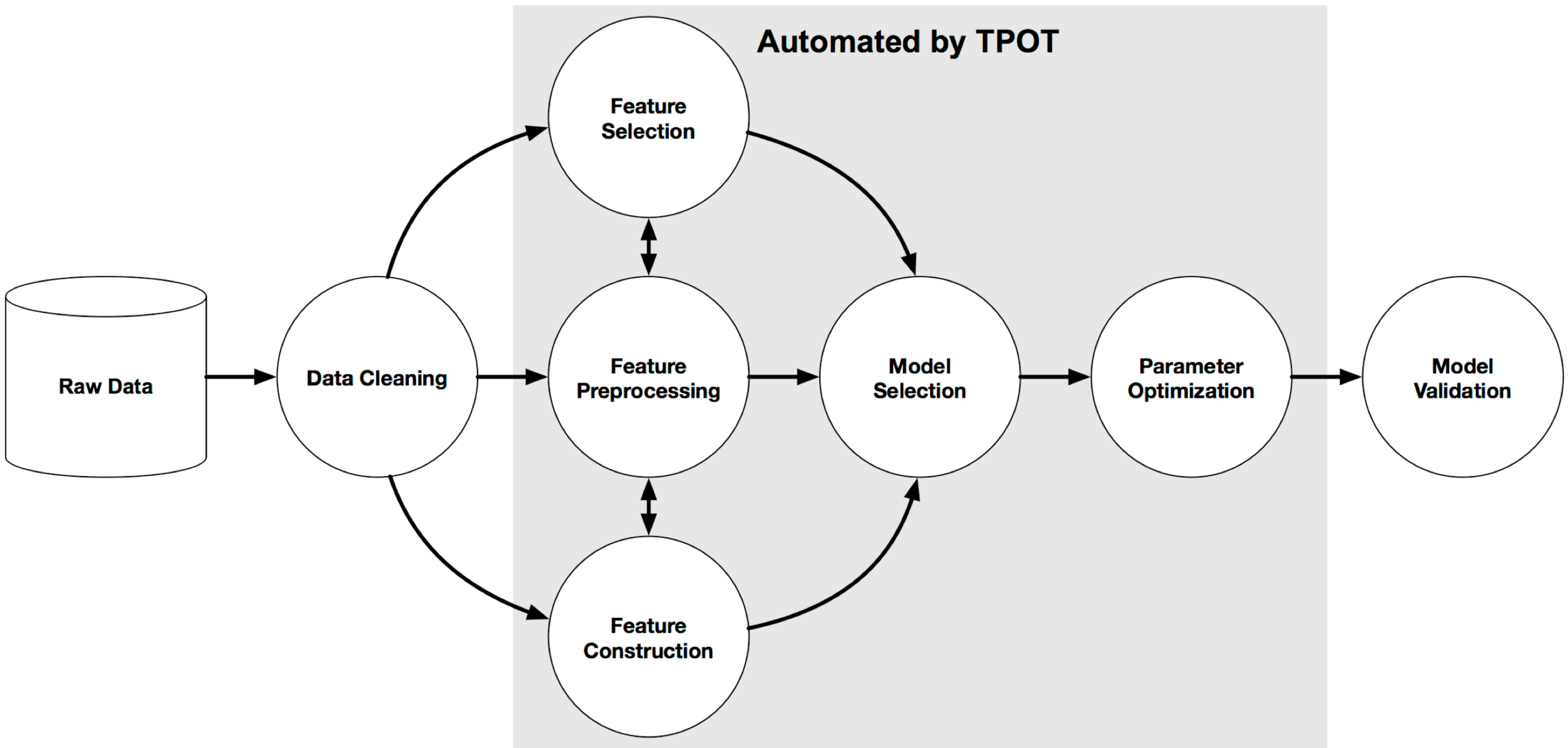
- TPOT is a Python Automated Machine Learning tool built on top of scikit-learn, a general-purpose Python machine learning library.

  https://github.com/EpistasisLab/tpot

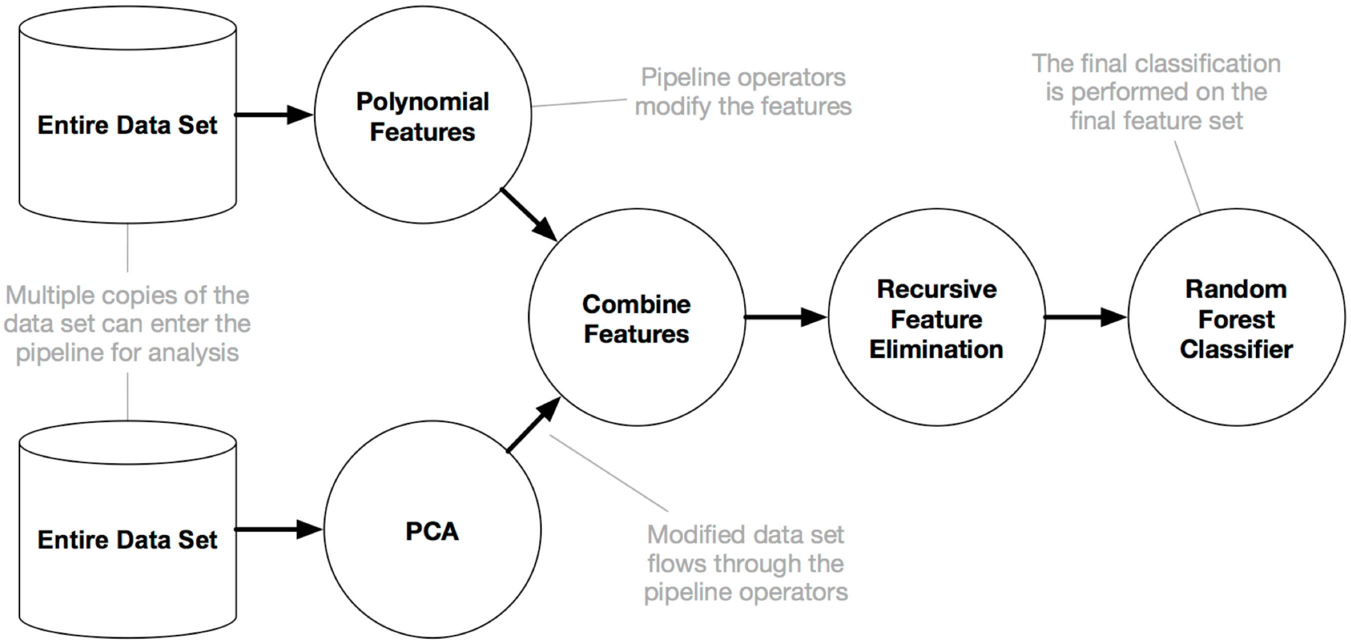
Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore (2016). Automating biomedical data science through tree-based pipeline optimization. *Applications of Evolutionary Computation*, pages 123-137.

TPOT

An example Machine Learning pipeline:



TPOT



- A GP builds trees of pipeline operators to maximize the final classification accuracy of the pipeline
- It evolves 1) the sequence of pipeline operators that act on the data set as well as 2) the parameters of these operators, e.g., the number of trees in a random forest or the number of features to select during feature selection.
- It uses the Python GP package DEAP.

Other related systems

- Autostacker: evolves multiple ensemble models within a stacked architecture
- DarwinML: graph-based

GP technical summary tableau

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

Project on GP or ES

- Choose your own topic!
- Look for inspiration at
- GP for automatic machine learning: see, e.g. http://automl.chalearn.org/
- Papers on GP and ES at GECCO 2020 https://gecco-2020.sigevo.org/index.html/Accepted+Papers
- Papers on GP and ES at PPSN 2020 https://ppsn2020.liacs.leidenuniv.nl/

Deadline for the first assignment is 9 February

Upload your report with answers and the source code in Brightspace.

Next week: Swarm Intelligence