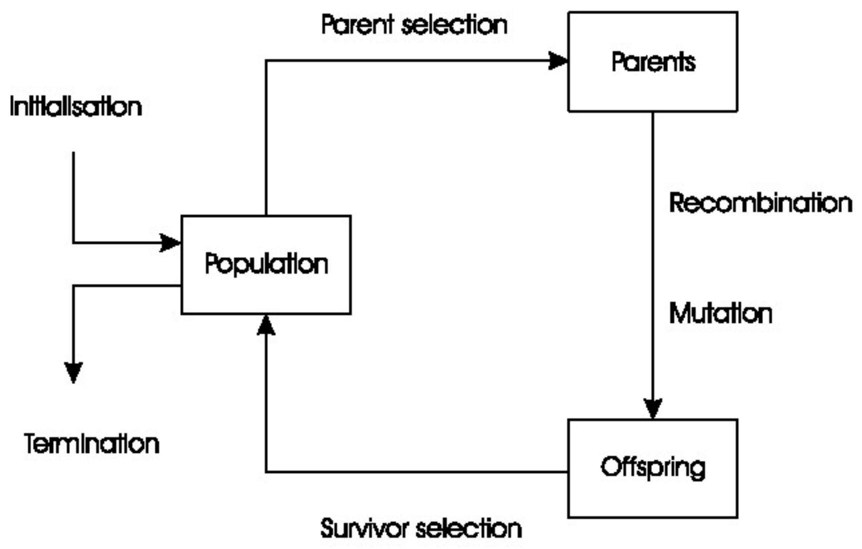
Evolution Strategies, Evolutionary Programming, and Genetic Programming



Evolution Strategies

Eiben/Smith Chapter 4

ES quick overview

- Developed: Germany in the 1970's
- Early names: I. Rechenberg, H.-P. Schwefel
- Typically applied to:
  - numerical optimisation
- Attributed features:
  - fast
  - good optimizer for real-valued optimisation
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard

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

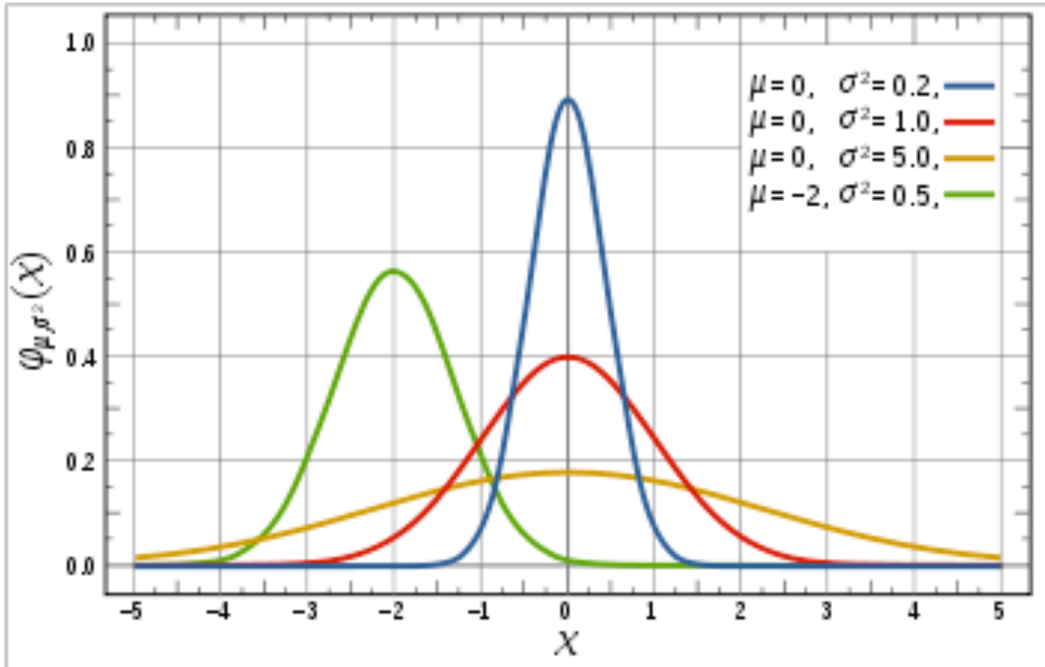
The μ and λ are Greek letters, written as mu and lambda.

Introductory example

- Task: minimimise f : $R^n \rightarrow R$
- Original algorithm: "two-membered ES" using
  - Vectors from R^n directly as chromosomes
  - Population size 1
  - Only mutation creating one child
  - Greedy selection

Introductory example: pseudocde

```
Set $t = 0$
Create initial point $x^t = \langle x_1^t,\ldots,x_n^t \rangle$
REPEAT UNTIL (*TERMIN.COND* satisfied) DO
    Draw $z_i$ from a normal distr. for all $i = 1,\ldots,n$
    $y_i^t = x_i^t + z_i$
    IF $f(x^t) < f(y^t)$ THEN $x^{t+1} = x^t$ ELSE $x^{t+1} = y^t$
    Set $t = t+1$
OD
```



Introductory example: mutation mechanism

- z values drawn from normal distribution $N(\mu,\sigma)$
  - mean μ is set to 0
  - variation σ is called mutation step size
- σ is varied on the fly by the "1/5 success rule":
  This rule resets σ after every k iterations by
  - $\sigma = \sigma / c$ if $p_s > 1/5$
  - $\sigma = \sigma \cdot c$ if $p_s < 1/5$
  - $\sigma = \sigma$ if $p_s = 1/5$

  where p_s is the % of successful mutations, $0.8 \leq c \leq 1$

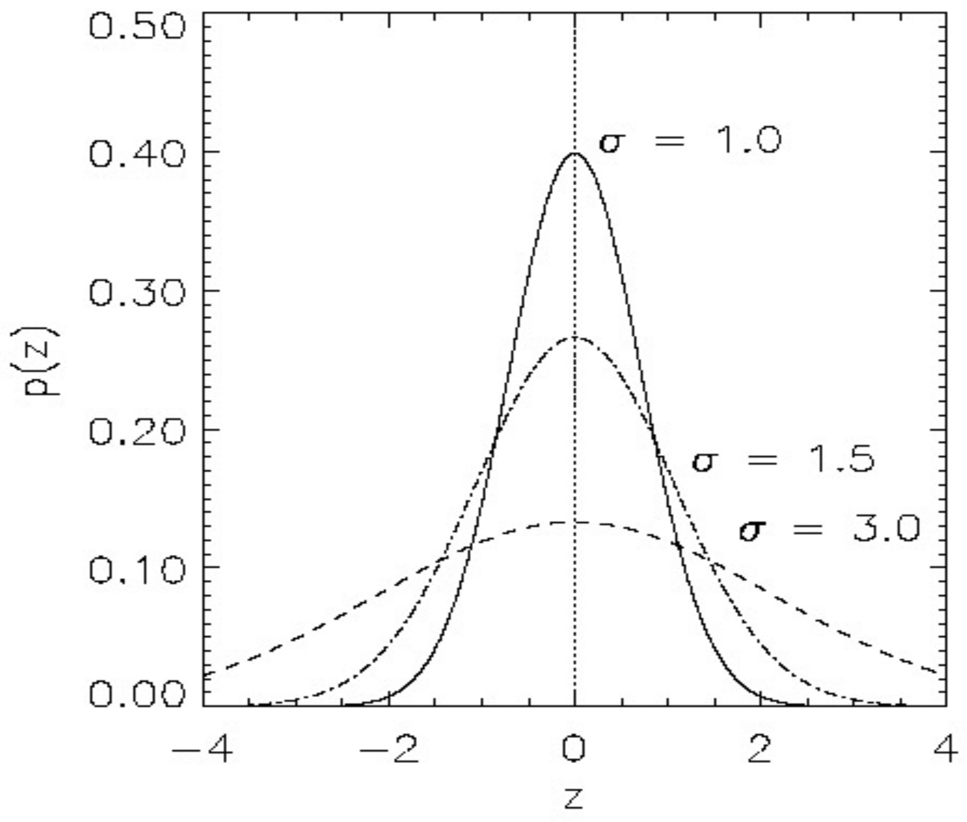
Introductory example: mutation mechanism

- z values drawn from normal distribution $N(\mu,\sigma)$
  - mean μ is set to 0
  - variation σ is called mutation step size
- σ is varied on the fly by the "1/5 success rule":
  This rule resets σ after every k iterations by
  - $\sigma = \sigma / c$ if $p_s > 1/5$ (frequent success $\rightarrow$ increase σ)
  - $\sigma = \sigma \cdot c$ if $p_s < 1/5$ (rare success $\rightarrow$ reduce σ)
  - $\sigma = \sigma$ if $p_s = 1/5$

  where p_s is the % of successful mutations, $0.8 \leq c \leq 1$

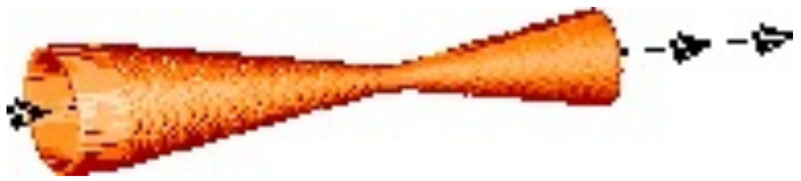
Illustration of normal distribution

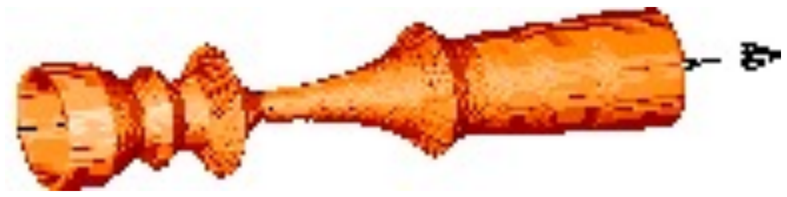
$\mathcal{N}(\mu, \sigma^2)$ $\qquad \dfrac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$



Another historical example: the jet nozzle experiment

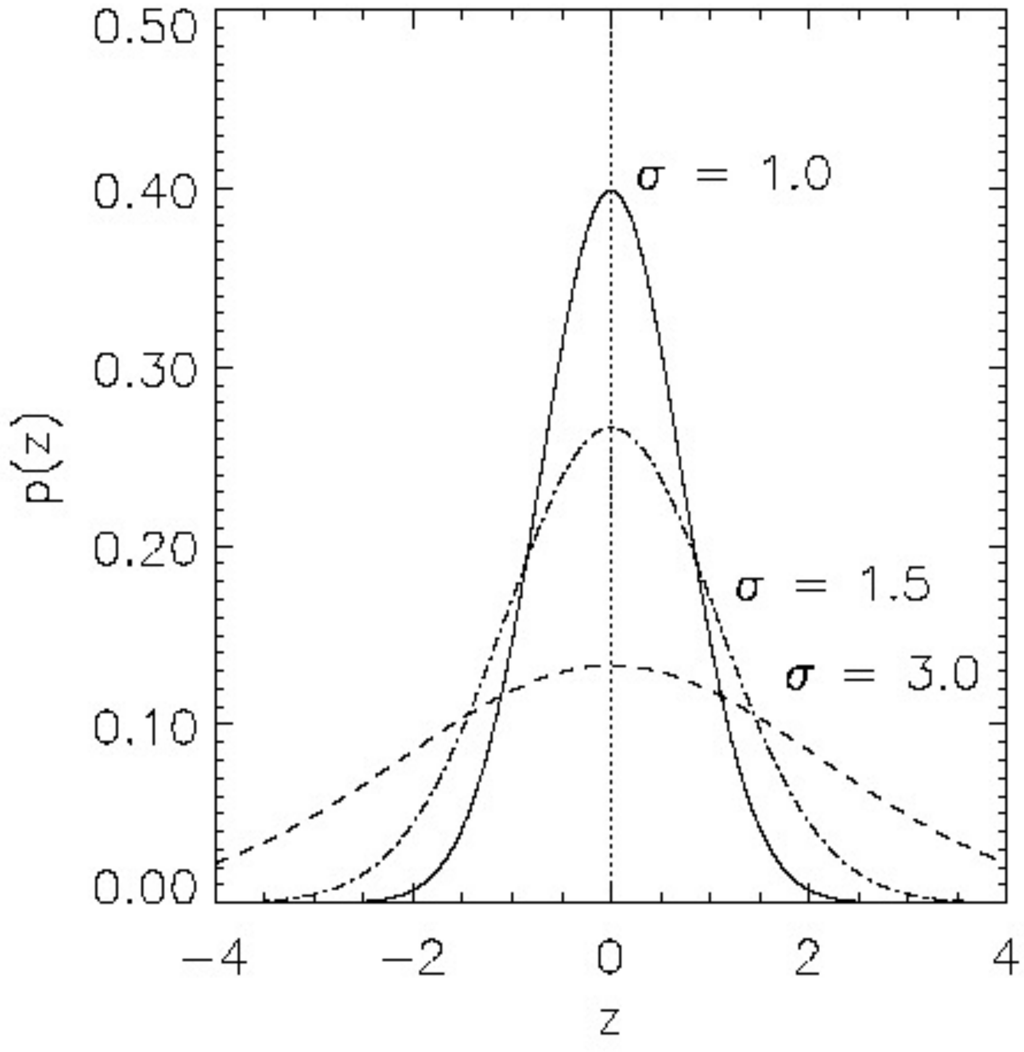
Task: to optimize the shape of a jet nozzle
Approach: random mutations to shape + selection


Initial shape


Final shape
(32% more efficient)

Genetic operators: mutations

The one dimensional case

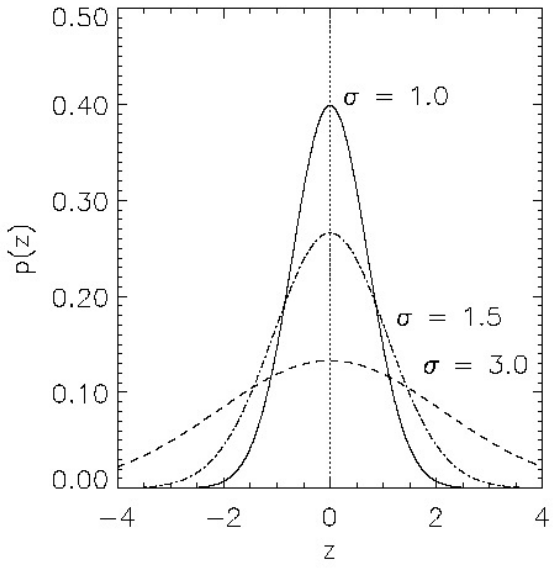


General Representation

- Chromosomes consist of three parts:
  - Object variables: $x_1,\ldots,x_n$
  - Strategy parameters:
    - Mutation step sizes: $\sigma_1,\ldots,\sigma_{n_\sigma}$
    - Rotation angles: $\alpha_1,\ldots,\alpha_{n_\alpha}$
- Not every component is always present
- Full size: $\langle x_1,\ldots,x_n, \sigma_1,\ldots,\sigma_n, \alpha_1,\ldots,\alpha_k \rangle$
- where $k = n(n-1)/2$ (number of i,j pairs)

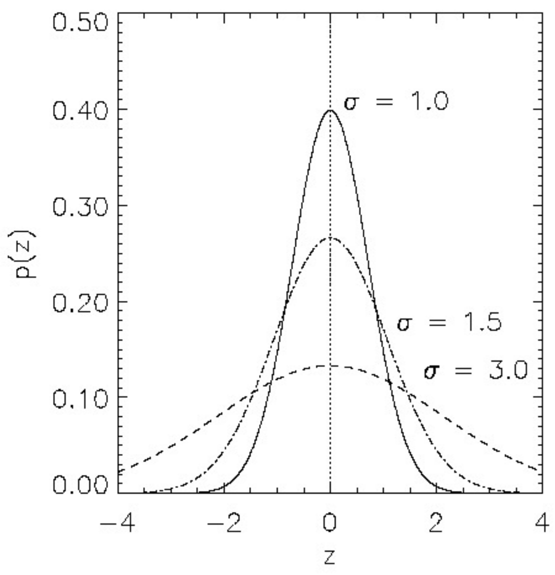
Mutation

- Main mechanism: changing value by adding random noise drawn from normal distribution
- $x'_i = x_i + N(0,\sigma)$
- Key idea:
  - σ is part of the chromosome $\langle x_1,\ldots,x_n, \sigma \rangle$
  - σ is also mutated into σ' (see later how)
- Thus: mutation step size σ is coevolving with the solution x



Mutate σ first

- Net mutation effect: $\langle x, \sigma \rangle \rightarrow \langle x', \sigma' \rangle$
- Order is important:
  - first $\sigma \rightarrow \sigma'$ (see later how)
  - then $x \rightarrow x' = x + N(0,\sigma')$
- Rationale: new $\langle x', \sigma' \rangle$ is evaluated twice
  - Primary: x' is good if $f(x')$ is good
  - Secondary: σ' is good if the x' it created is good
- Reversing mutation order would not work

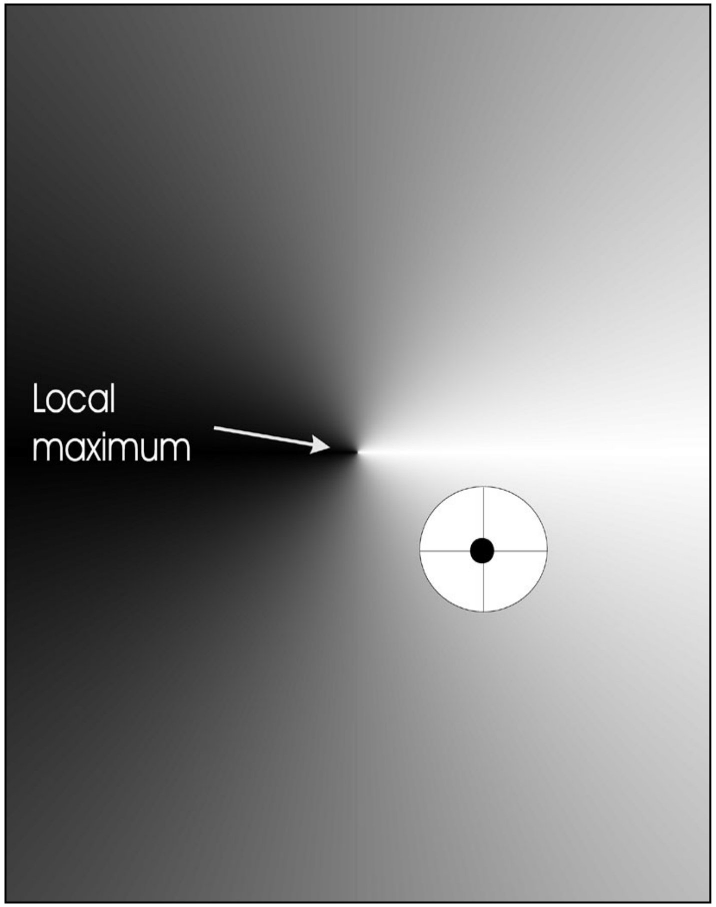


Mutation case 1: Uncorrelated mutation with one σ

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma \rangle$
- $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
- $x'_i = x_i + \sigma' \cdot N(0,1)$
- Typically the "learning rate" $\tau \propto 1/ n^{½}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

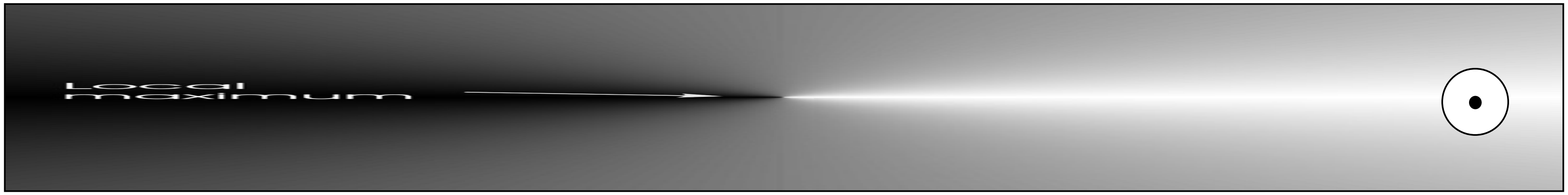
Note: the exp(…) expression is used to get a factor whose values are on average 1.0, and concentrated around 1.0, and the deviations from 1.0 are neutral on average.

Mutants with equal likelihood



Circle: mutants having the same chance to be created

Mutants with equal likelihood

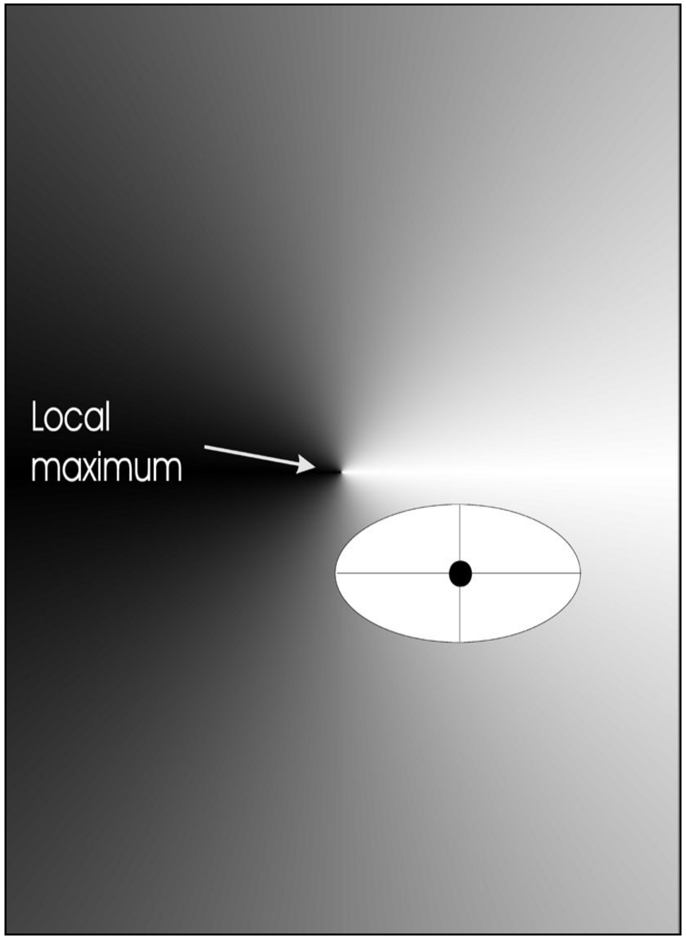


Circle: mutants having the same chance to be created

Mutation case 2: Uncorrelated mutation with n σ' s

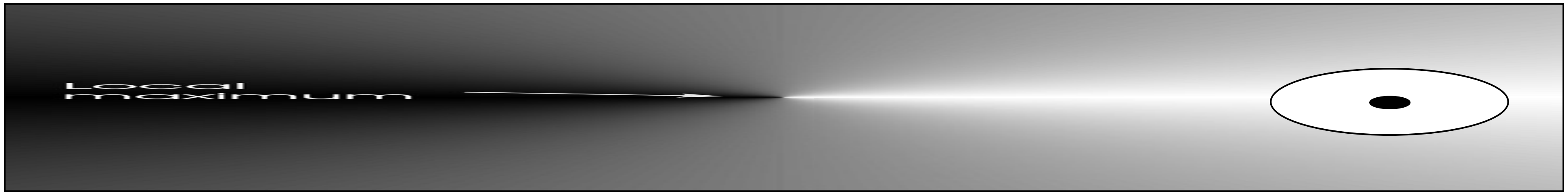
- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n \rangle$
- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $x'_i = x_i + \sigma'_i \cdot N_i(0,1)$
- Two learning rate parmeters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau \propto 1/(2n)^{½}$ and $\tau \propto 1/(2n^{½})^{½}$
- And $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

Mutants with equal likelihood



Circle: mutants having the same chance to be created

Mutation case 3: Correlated mutations

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n ,\alpha_1,\ldots, \alpha_k \rangle$
- where $k = n \cdot (n-1)/2$
- and the covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = ½ \cdot (\sigma_i^2 - \sigma_j^2) \cdot \tan(2\, \alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α ‘s

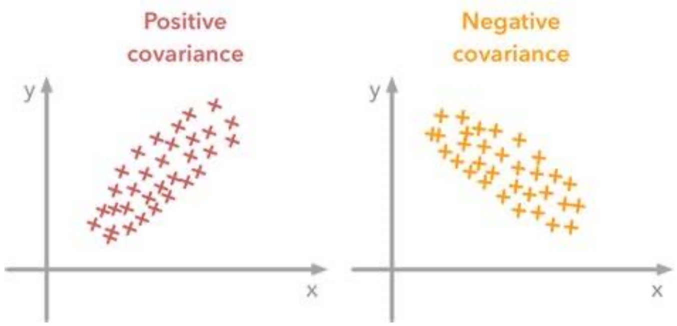
If you’d like to learn more, then look up “Covariance Matrix Adaptation Evolution Strategy (CMA-ES)”

Correlated mutations cont'd

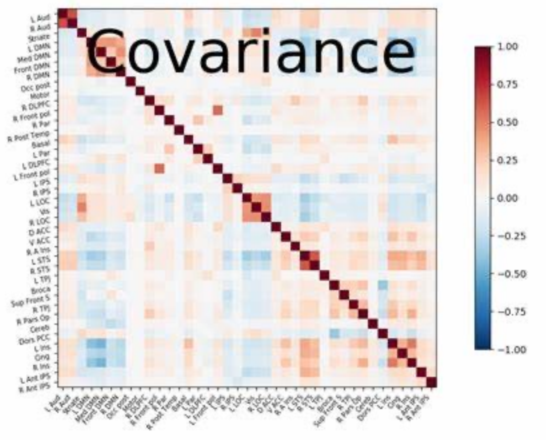
The mutation mechanism is then:

- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $\alpha'_j = \alpha_j + \beta \cdot N(0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + N(\boldsymbol{0}, \boldsymbol{C}')$
  - $\boldsymbol{x}$ stands for the vector $\langle x_1,\dots,x_n \rangle$
  - $\boldsymbol{C}'$ is the covariance matrix $\boldsymbol{C}$ after mutation of the α values
- $\tau' \propto 1/(2n)^{½}$ and $\tau \propto 1/(2n^{½})^{½}$ and $\beta \approx 5°$
- $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$ and
- $|\alpha'_j| > \pi \Rightarrow \alpha'_j = \alpha'_j - 2\pi \operatorname{sign}(\alpha'_j)$

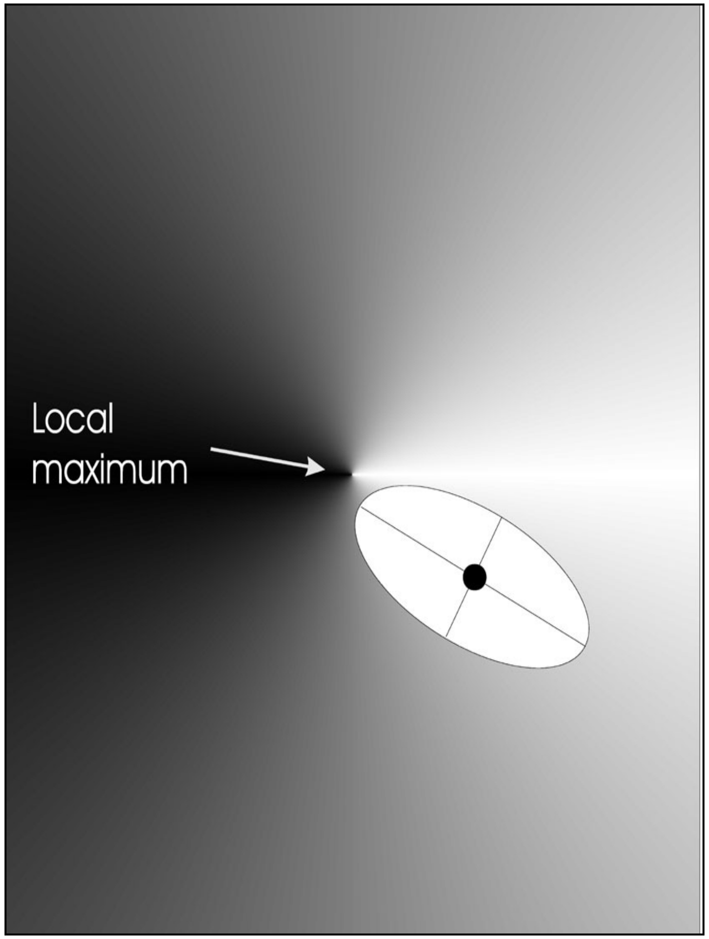
Example



Matrix for pairs of variables



Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

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

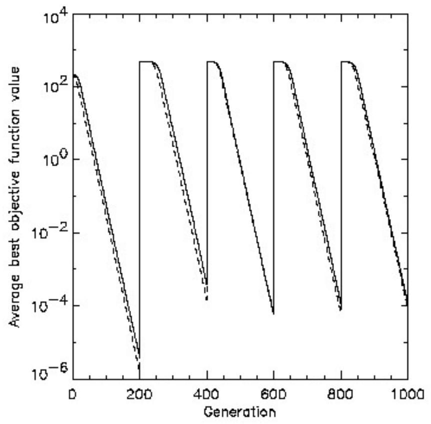
- Applied after creating λ children from the μ parents by mutation and recombination
- Deterministically chops off the "bad stuff"
- Basis of selection is either:
  - The set of children only: (μ,λ)-selection
  - The set of parents and children: $(\mu+\lambda)$-selection

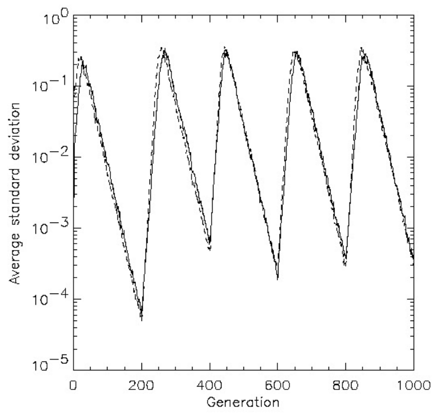
Survivor selection

- $(\mu+\lambda)$-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!





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 misadapted σ 's
- Mixing strategy parameters by (intermediary) recombination on them

Example application: the cherry brandy experiment

- Task to create a colour mix yielding a target colour (that of a well known cherry brandy)
- Ingredients: water + red, yellow, blue dye
- Representation: ⟨ w, r, y ,b ⟩ no self-adaptation!
- Values scaled to give a predefined total volume (30 ml)
- Mutation: lo / med / hi σ values used with equal chance
- Selection: (1,8) strategy

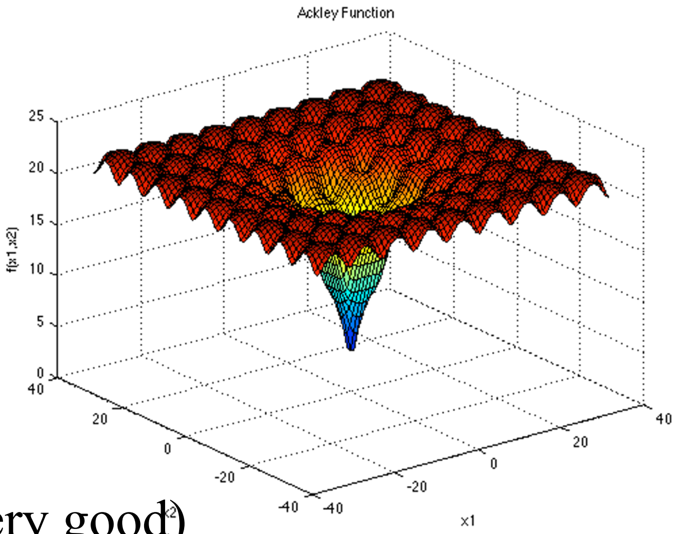
Example application: the cherry brandy experiment

- Fitness: students effectively making the mix and comparing it with target colour
- Termination criterion: student satisfied with mixed colour
- Solution is found mostly within 20 generations
- Accuracy is very good

Example application: 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 200 000 fitness evaluations
- Results: average best solution is $7.48 \cdot 10^{-8}$ (very good)

Evolutionary Programming

Eiben/Smith Chapter 5

EP quick overview

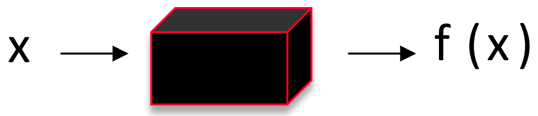
- Developed: USA in the 1960's
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: machine learning tasks by finite state machines
  - contemporary EP: (numerical) optimization
- Attributed features:
  - very open framework: any representation and mutation operators are OK
  - crossbred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

Historical EP perspective

- 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

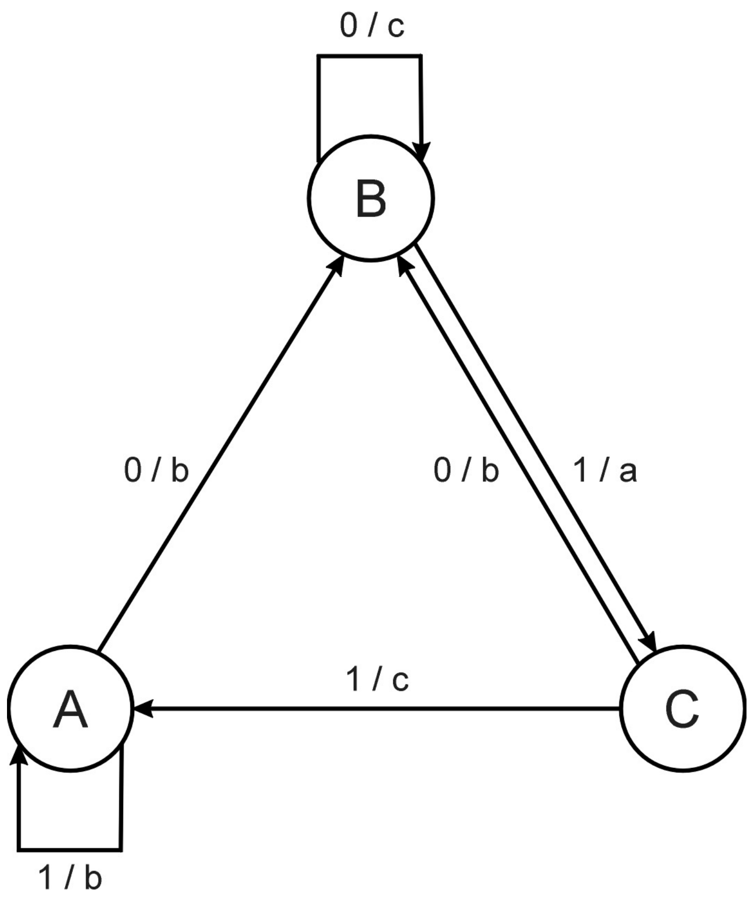
Prediction by finite state machines

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



FSM example

- Consider the FSM with:
  - States S = {A, B, C}
  - Inputs I = {0, 1}
  - Outputs O = {a, b, c}
  - δ given by a diagram



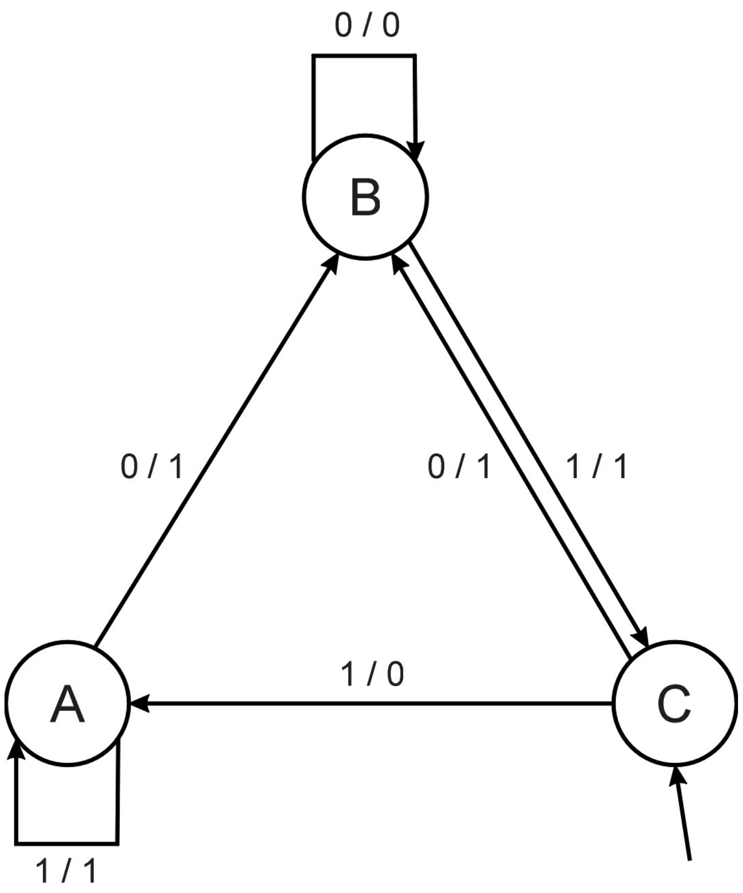
Question
What does this FSM do? (in English)

FSM as predictor

- Consider the following FSM
- Task: guess next input
- Quality: % of in$_{(i+1)}$ = out$_i$
- Given initial state C
- Input sequence 011101
- Leads to output 110111

  states BCAABC

- Quality: 3 out of 5 (2nd, 3rd, 6th)



Introductory example (1/2): 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 many” 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)
  - Add a state
  - Delete a state
  - Change the initial state
- Survivor selection: (μ+μ)
- Results: overfitting, after 202 inputs best FSM had one state and both outputs were 0, i.e., it always predicted "not prime"
- Main point: not perfect accuracy but proof that simulated evolutionary process can create good solutions for intelligent task

V.P. Holmes (1973) "Recognizing prime numbers with an evolutionary program," Master's thesis, New Mexico State University, Las Cruces, NM.

Introductory example (2/2): evolving FSMs to play checkers

2001: Evolving an Expert Checkers Playing Program without Using Human Expertise

Results:

- Expert class (better than 99.61% of all rated players)
- Interesting in the context of artificial intelligence
  - No input of human expertise about good short-term strategies or endgames
  - The selection function averages over five games (blurred effects of strategies)
  - The strategies evolved against each other, without human interaction!

More information:
https://pdfs.semanticscholar.org/27df/56624d9e0f8570e2624d6226201338b7df7f.pdf

http://fineamericanmadetoys.com/image/data/MLW/50322.jpg

Genetic Programming

Eiben/Smith Chapter 6

GP quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)

GP technical summary tableau

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

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches 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 $(NOC \leq 1)$ AND $(S \geq 30000)$ THEN good ELSE bad

In general:

IF formula THEN good ELSE bad

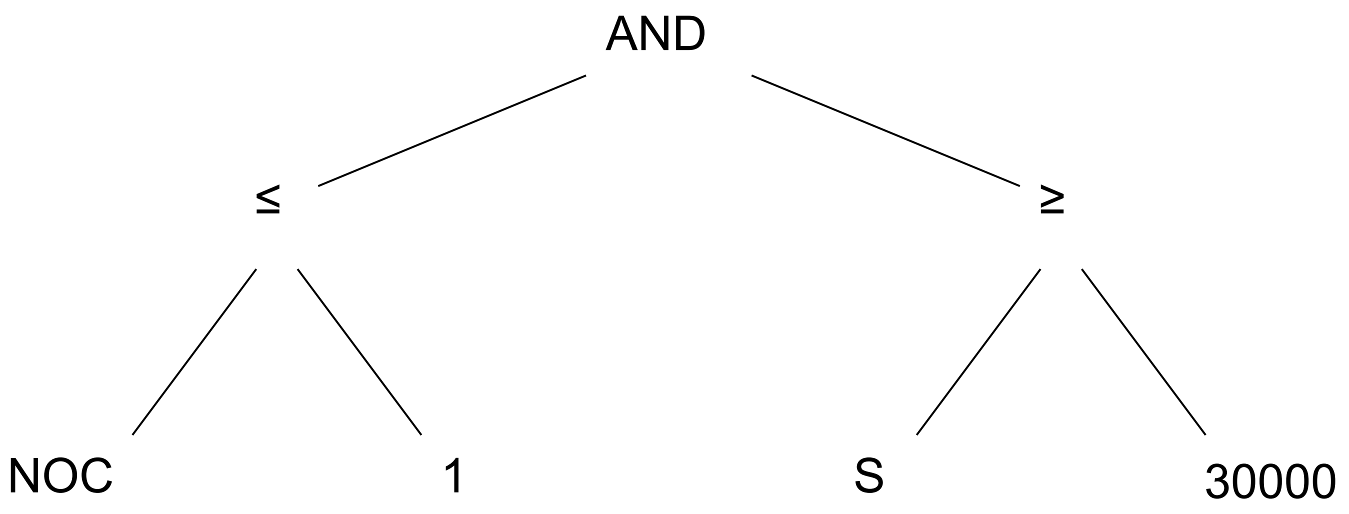
Only unknown is the right formula, hence

- 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 $(NOC \le 1)$ AND $(S \ge 30000)$ THEN good ELSE bad

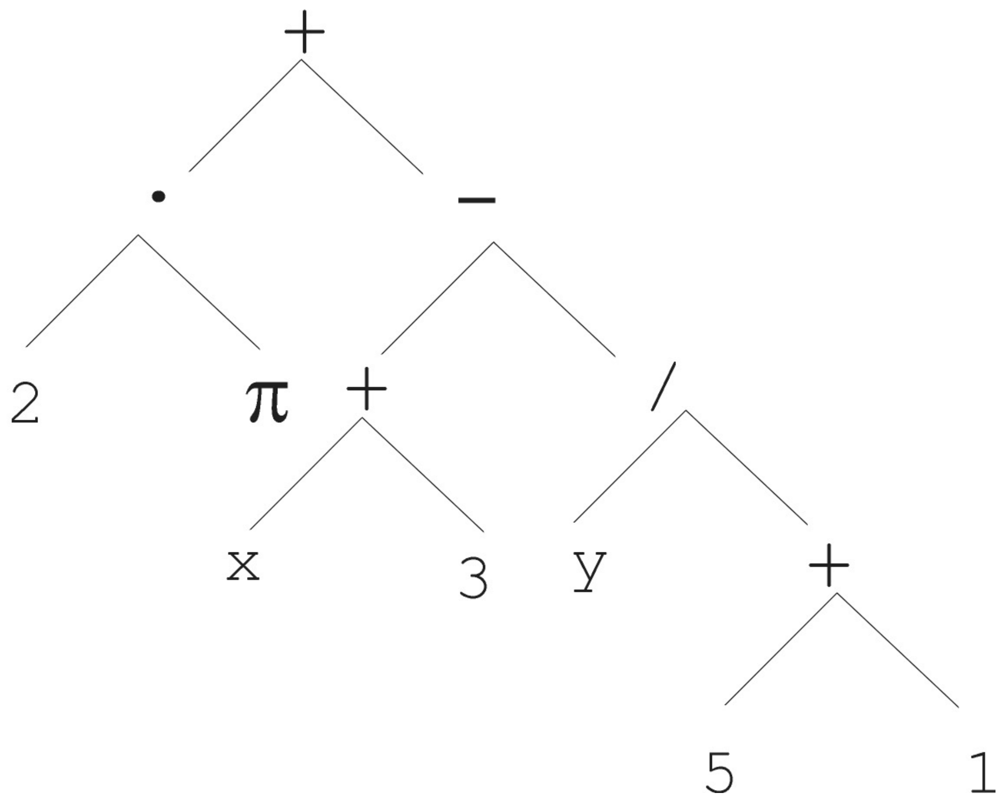
can be represented by the following tree



Tree-based representation

Trees are a universal form, e.g. consider

- Arithmetic formula

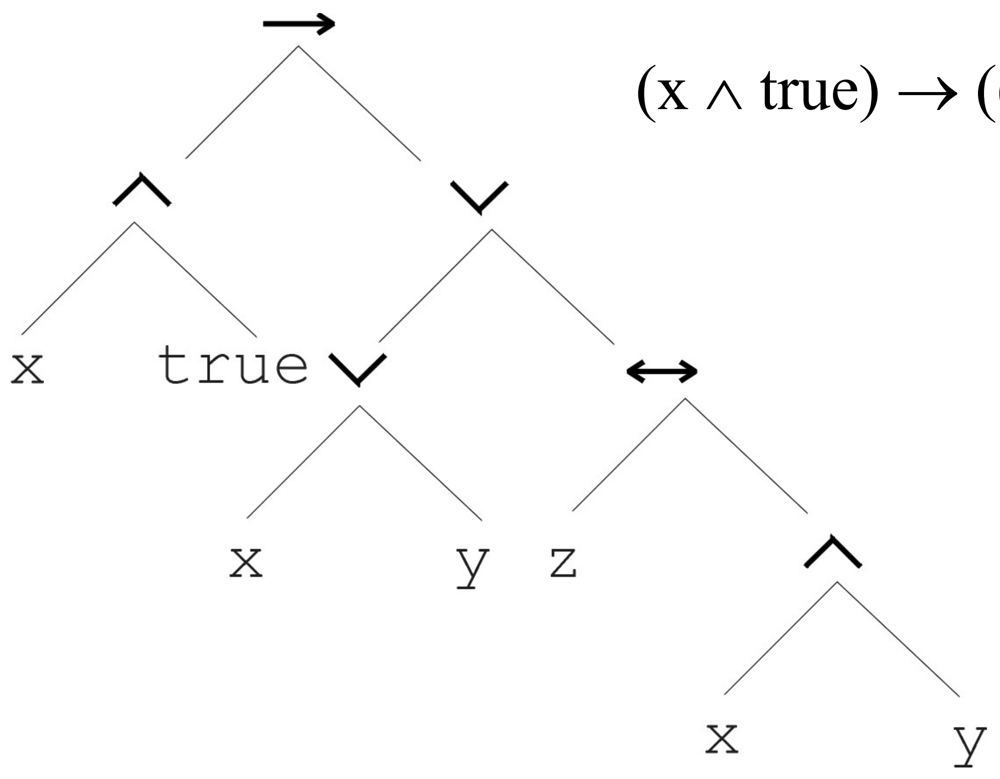


$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree-based representation

Trees are a universal form, e.g. consider

- Logical formula

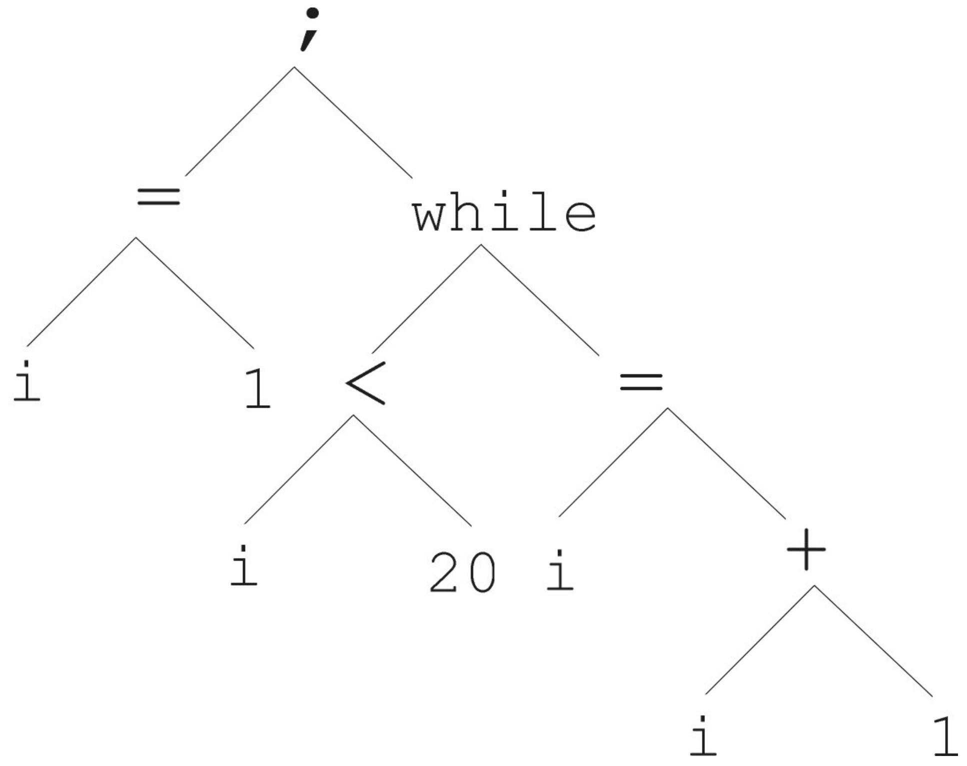


$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

Tree-based representation

Trees are a universal form, e.g. consider

- Program



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

Tree-based representation

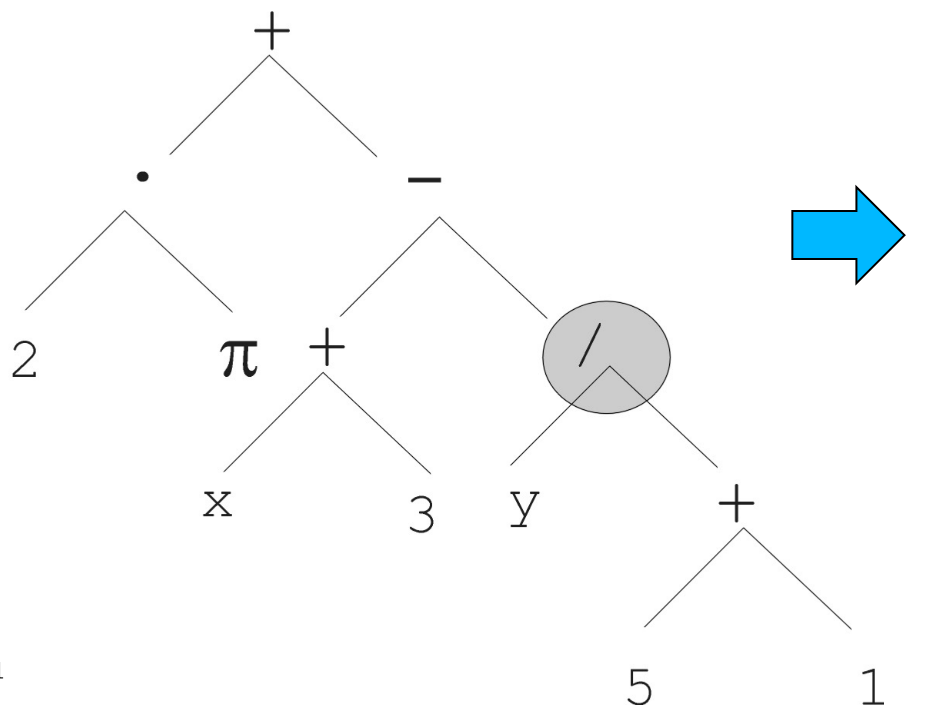
- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree-shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width

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 property: any $f \in F$ can take any $g \in F$ as argument)

Mutation

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

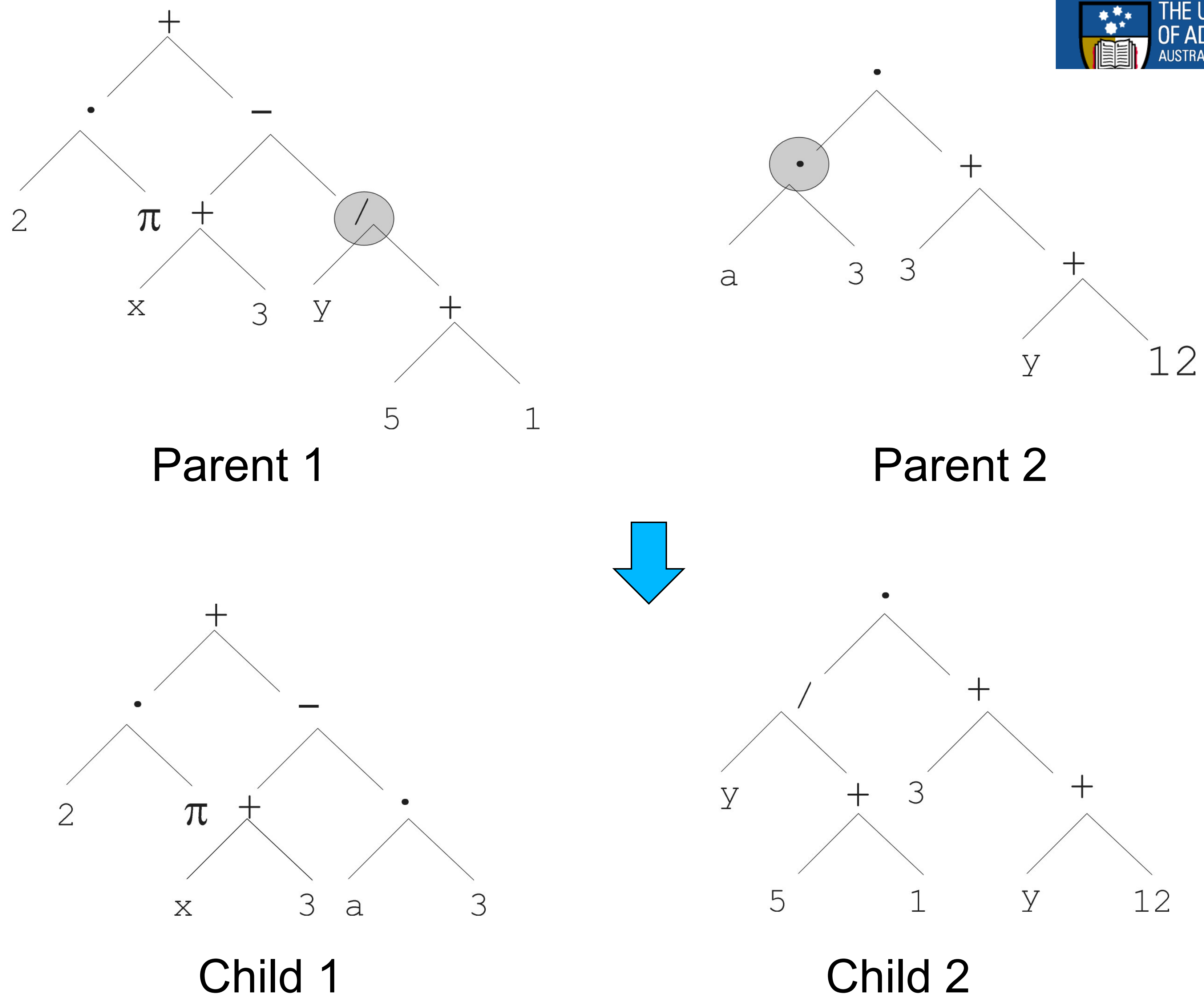


Mutation

- Mutation has two parameters:
  - Probability p_m to choose mutation
  - 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



+
·
−
2
π
+
/
x
3
y
+
5
1
Parent 1
·
·
+
a
3
3
+
y
12
Parent 2
+
·
−
2
π
+
·
x
3
a
3
Child 1
·
/
+
y
+
3
+
5
1
y
12
Child 2

Selection

- Parent selection typically fitness proportionate
- Over-selection in very large populations (1000s of solutions)

  rank population by fitness and divide it into two groups:

  – group 1: best x% of population, group 2 other (100-x)%

Group 1 (best x%)	Group 2

  – 80% of selection operations chooses from group 1, 20% from group 2

  – for pop. size = 1000, 2000, 4000, 8000 $\rightarrow$ x = 32%, 16%, 8%, 4%

  – motivation: to increase efficiency, %' s come from rule of thumb
- Survivor selection:
  – Typical: generational scheme (thus none)

  – Recently steady-state is becoming popular for its elitism

Initialisation

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

Bloat

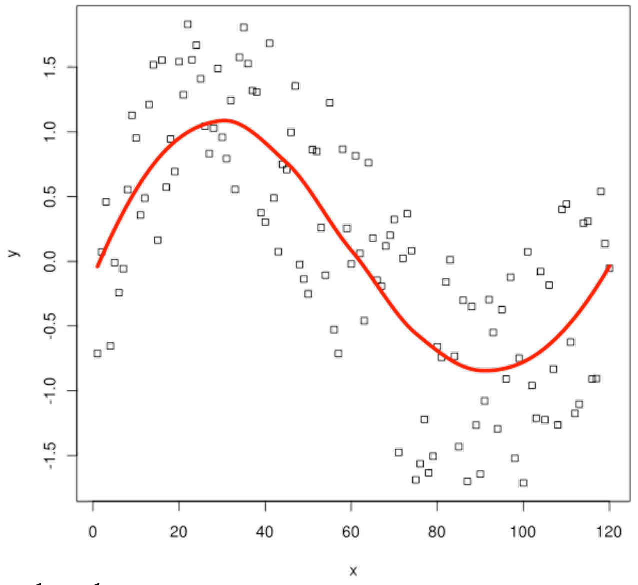
- Bloat = "survival of the fattest", i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver "too big" children
  - Parsimony pressure: penalty for being oversized

Problems involving "physical" environments

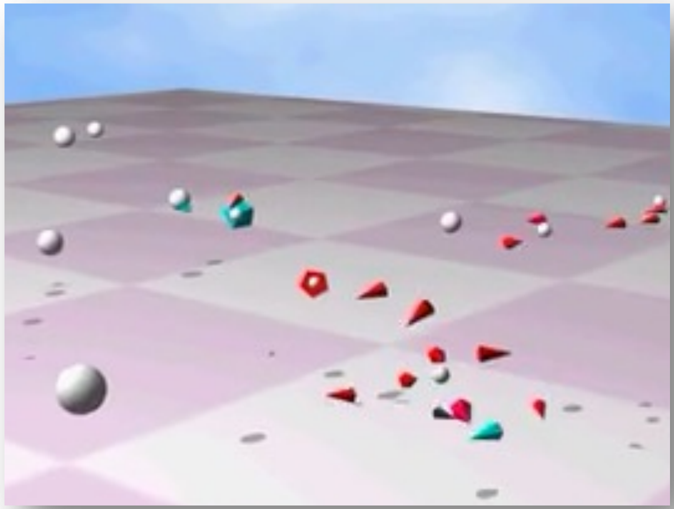
- Trees for data fitting vs. trees (programs) that are "really" executable
- Execution can change the environment → the calculation of fitness
- Example: robot controller
- Fitness calculations mostly by simulation, ranging from expensive to extremely expensive (in time)
- But evolved controllers are often to very good

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 solution:
  - Representation by $F = \{+, -, /, \sin, \cos\}$, $T = \mathbf{R} \cup \{x\}$
  - Fitness is the error $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - All operators standard
  - pop.size = 1000, ramped half-half initialisation
  - Termination: n “hits” or 50000 fitness evaluations reached (where “hit” is if $| f(x_i) - y_i | < 0.0001$)



Evolving Computer Programs



A short video of the SwarmEvolve 2.0 system described in Emergence of Collective Behavior in Evolving Populations of Flying Agents, by Lee Spector, Jon Klein, Chris Perry, and Mark Feinstein.

In this system agent behavior is driven by programs that are initially random but subsequently evolve "autoconstructively" (computing the code for their own offspring) over the course of the simulation.

In the initial portion of this video all of the agents have random programs, none of which give their agents sufficient intelligence to seek food (the white spheres) or produce offspring. New random agents are added periodically until, around the 40 second mark in this video, agents that happen to be "reproductively competent" (able to get food and produce offspring that can themselves get food and produce offspring, etc.) are created. This produces a population explosion, after which competition for the limited food resources will begin to drive evolution.

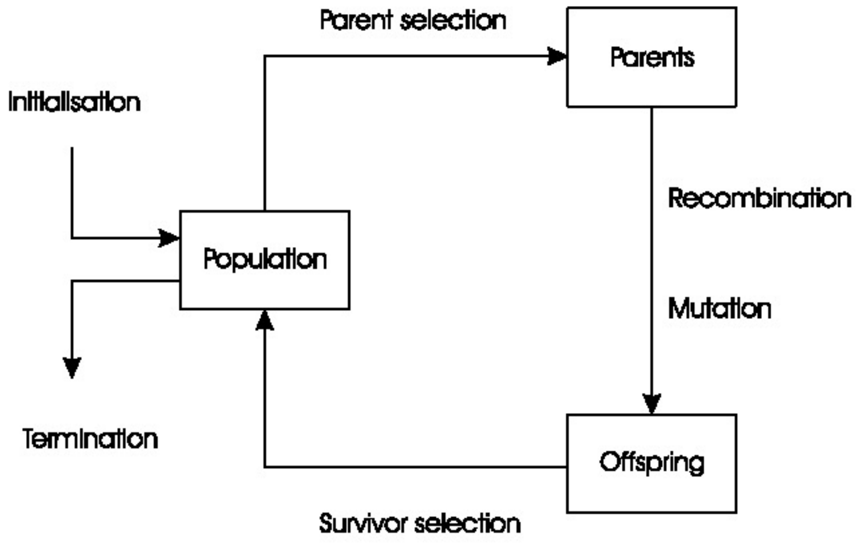
The programs driving the behavior of these agents are expressed in the Push programming language for evolutionary computation (http://pushlanguage.org).

https://www.youtube.com/watch?v=iADiA3HrxA0, more videos at http://faculty.hampshire.edu/lspector/gecco2003-collective.html

*** SUMMARY ***

What we have seen today:

- Evolution Strategies: for real-valued vectors, uses self-adaptation
- Evolution Programming: for finite-state machines
- Genetic Programming: for graphs


Parent selection
Parents
Initialisation
Recombination
Population
Mutation
Termination
Offspring
Survivor selection