Natural Computing

Lecture 8

Michael Herrmann
mherrman@inf.ed.ac.uk
phone: 0131 6 517177
Informatics Forum 1.42

14/10/2011

Differential Evolution (DE) and Genetic Programming (GP)

Differential Evolution
for Continuous Function Optimization

- Algorithm by Kenneth Price and Rainer Storn
- Individuals are continuous vectors (direct encoding)
- Using vector differences for mutating the vector population
- "This way no separate probability distribution has to be used which makes the scheme completely self-organizing."
- Small population size (usually up to 40)
- Mutation and crossover (only loosely related to the biological picture)

- "Contour matching": vector population adapts such that promising regions of the objective function surface are investigated automatically once they are detected
- "Swarm intelligence": Interaction among individuals is important, see PSO (later)
- Two parameters: weighting constant F, crossover constant C
- Typical values: $F = 0.5 \ldots 0.8$ ($F \geq \sqrt{(1 - C/2)/N}$), $C = 0.1 \ldots 0.9$ (if the problem is separable, C can be smaller), $4 < N \approx 5D$
- Check the applet at www.icsi.berkeley.edu/~storn/code.html

Differential Evolution: Algorithm

Population of N vectors of D-dimensions: $x_i,\ i = 1, 2, ..., N$

Step 1: $v_i(t+1) = x_q(t) + F \cdot (x_r(t) - x_s(t))$;

q, r, s are random indexes, all different and different from i.
Note that v_i has nothing to do with x_i! (t: generation counter)
(In a sense: three parents, but this is considered as mutation in DE)

$F \in [0, 2] \subset \mathbb{R}$ (possible amplification of the differential variation)

Step 2: Choose random numbers $\rho_j \in [0, 1],\ \ j \in \{1, \ldots, D\}$

$$u_{ji}(t+1) = \begin{cases} v_{ji}(t+1) & \text{if } \rho_j \leq C \\ x_{ji}(t) & \text{if } \rho_j > C \end{cases}$$

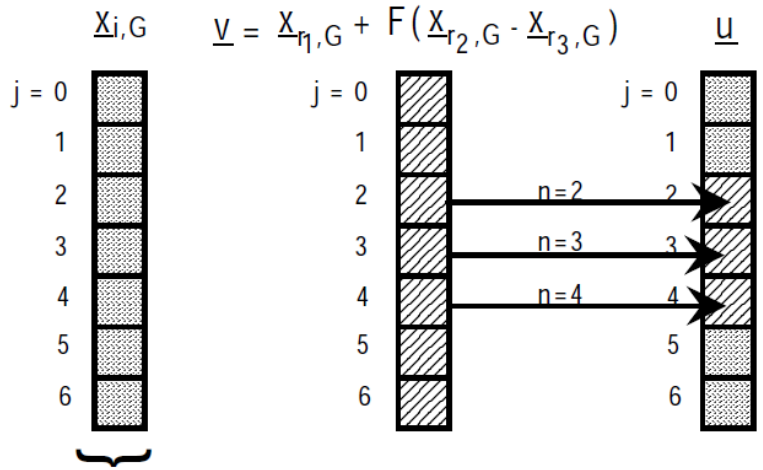
or by choosing a block $j \in [n, (n+L) \bmod D]$, $L \leq D$, $1 \leq n \leq D$ where L is randomly changed.

$$u_{i,t+1} = (u_{1i}(t+1), u_{2i}(t+1), \ldots, u_{Di}(t+1))$$

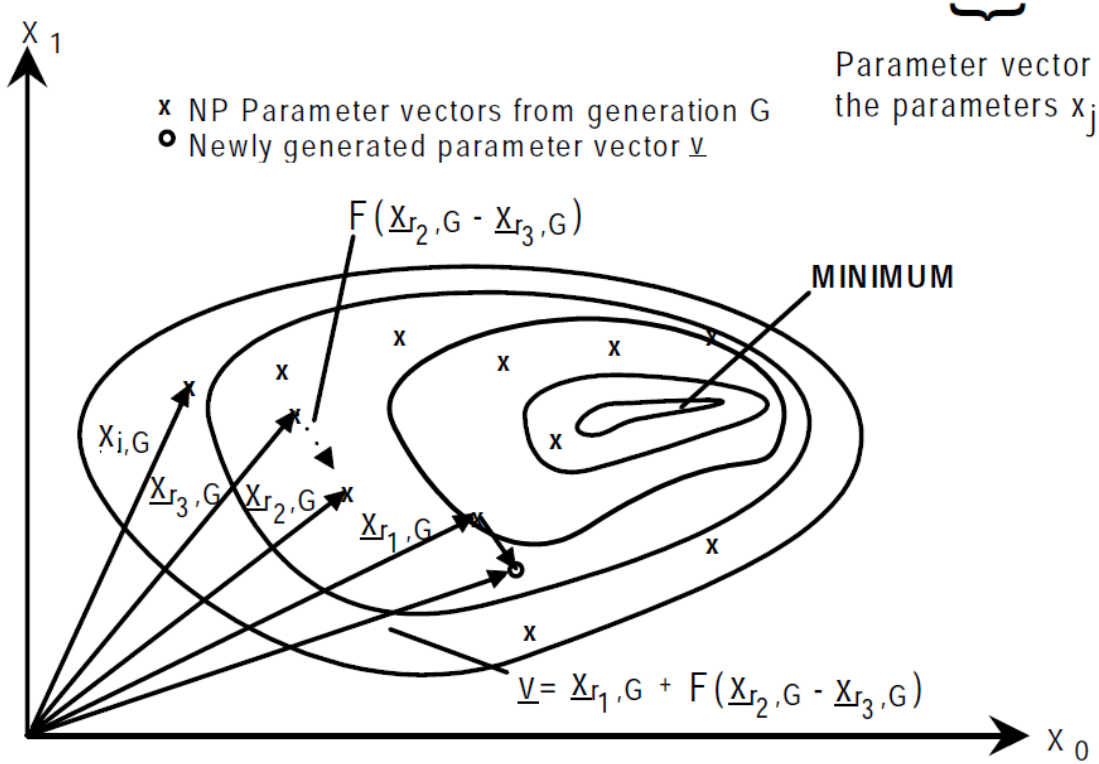
Selection: $x_i(t+1) = u_i(t+1)$ if $u_i(t+1)$ is better than $x_i(t)$,
otherwise $\quad x_i(t+1) = x_i(t)$

Rainer Storn & Kenneth Price (1997) Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11: 341–359.

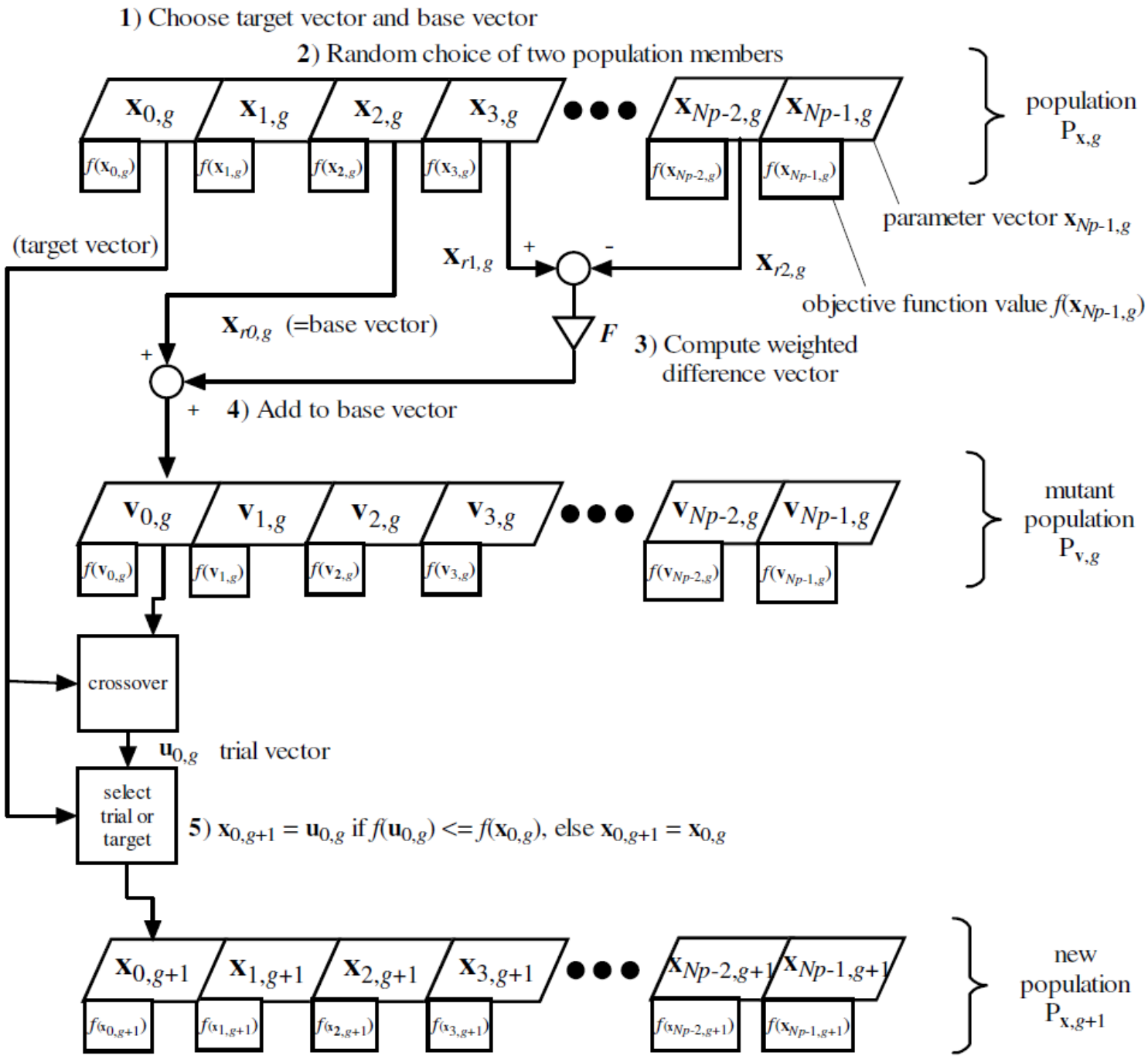
Differential Evolution



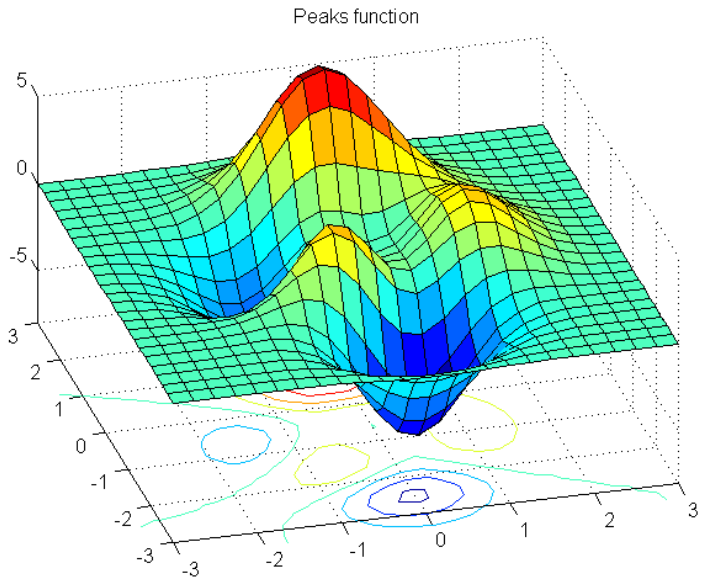
Parameter vector containing the parameters x_j, j=0,1, ... , D-1

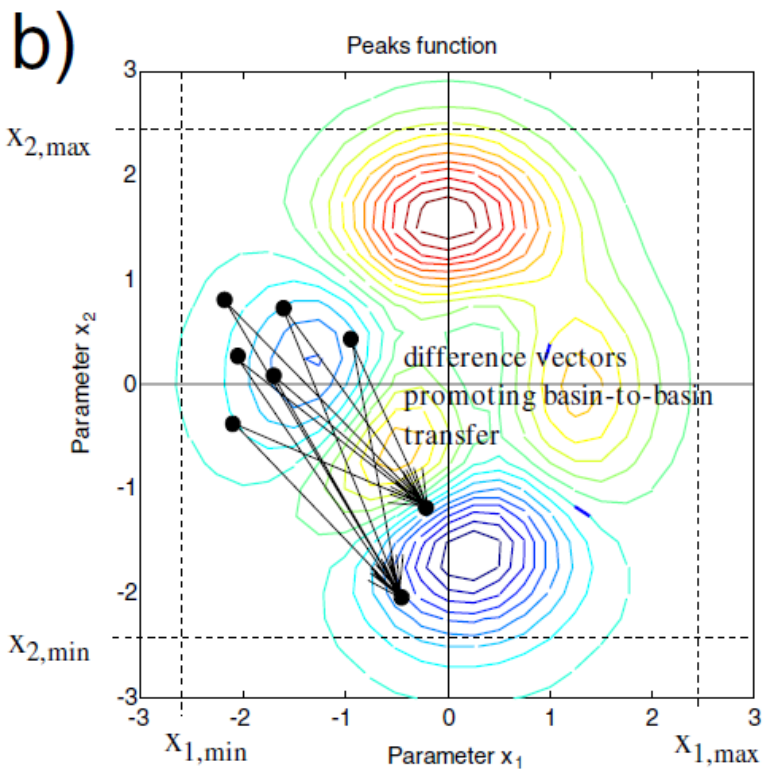


Rainer Storn and Kenneth Price: Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces, TR.


1) Choose target vector and base vector
2) Random choice of two population members
$\mathbf{x}_{0,g}$ $\mathbf{x}_{1,g}$ $\mathbf{x}_{2,g}$ $\mathbf{x}_{3,g}$ ●●● $\mathbf{x}_{Np-2,g}$ $\mathbf{x}_{Np-1,g}$
$f(\mathbf{x}_{0,g})$ $f(\mathbf{x}_{1,g})$ $f(\mathbf{x}_{2,g})$ $f(\mathbf{x}_{3,g})$ $f(\mathbf{x}_{Np-2,g})$ $f(\mathbf{x}_{Np-1,g})$
population $\mathbf{P}_{x,g}$
parameter vector $\mathbf{x}_{Np-1,g}$
objective function value $f(\mathbf{x}_{Np-1,g})$
(target vector)
$\mathbf{x}_{r1,g}$
$\mathbf{x}_{r2,g}$
+
-
$\mathbf{x}_{r0,g}$ (=base vector)
F
3) Compute weighted difference vector
+
+
4) Add to base vector
$\mathbf{v}_{0,g}$ $\mathbf{v}_{1,g}$ $\mathbf{v}_{2,g}$ $\mathbf{v}_{3,g}$ ●●● $\mathbf{v}_{Np-2,g}$ $\mathbf{v}_{Np-1,g}$
$f(\mathbf{v}_{0,g})$ $f(\mathbf{v}_{1,g})$ $f(\mathbf{v}_{2,g})$ $f(\mathbf{v}_{3,g})$ $f(\mathbf{v}_{Np-2,g})$ $f(\mathbf{v}_{Np-1,g})$
mutant population $\mathbf{P}_{v,g}$
crossover
$\mathbf{u}_{0,g}$ trial vector
select trial or target
5) $\mathbf{x}_{0,g+1} = \mathbf{u}_{0,g}$ if $f(\mathbf{u}_{0,g}) <= f(\mathbf{x}_{0,g})$, else $\mathbf{x}_{0,g+1} = \mathbf{x}_{0,g}$
$\mathbf{x}_{0,g+1}$ $\mathbf{x}_{1,g+1}$ $\mathbf{x}_{2,g+1}$ $\mathbf{x}_{3,g+1}$ ●●● $\mathbf{x}_{Np-2,g+1}$ $\mathbf{x}_{Np-1,g+1}$
$f(\mathbf{x}_{0,g+1})$ $f(\mathbf{x}_{1,g+1})$ $f(\mathbf{x}_{2,g+1})$ $f(\mathbf{x}_{3,g+1})$ $f(\mathbf{x}_{Np-2,g+1})$ $f(\mathbf{x}_{Np-1,g+1})$
new population $\mathbf{P}_{x,g+1}$

Differential Evolution: Basin-to-Basin Transfer

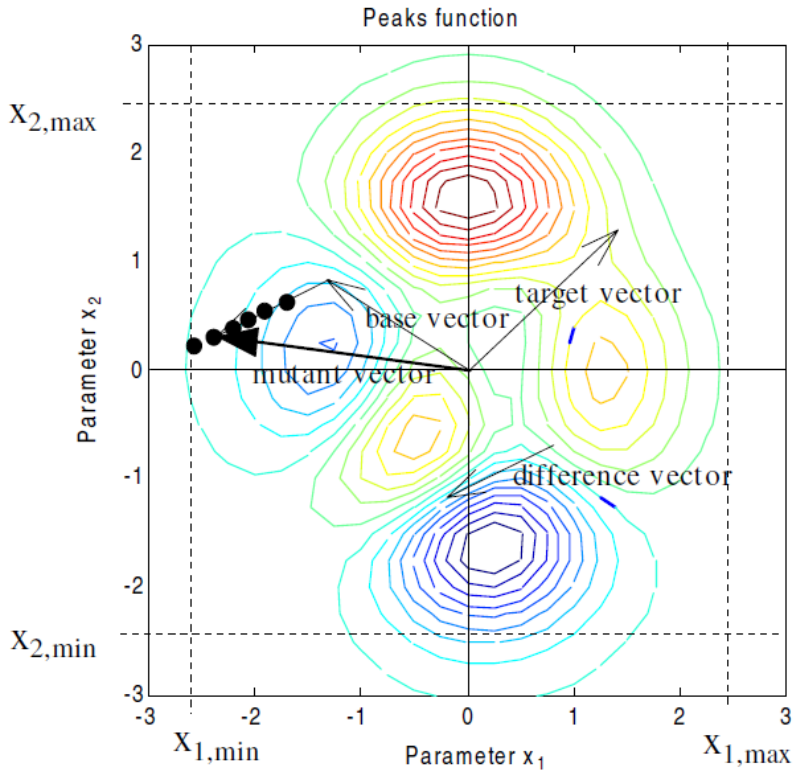




Peaks function a) and illustration of difference vectors b) that promote transfer of points between two basins of attraction of the objective function surface

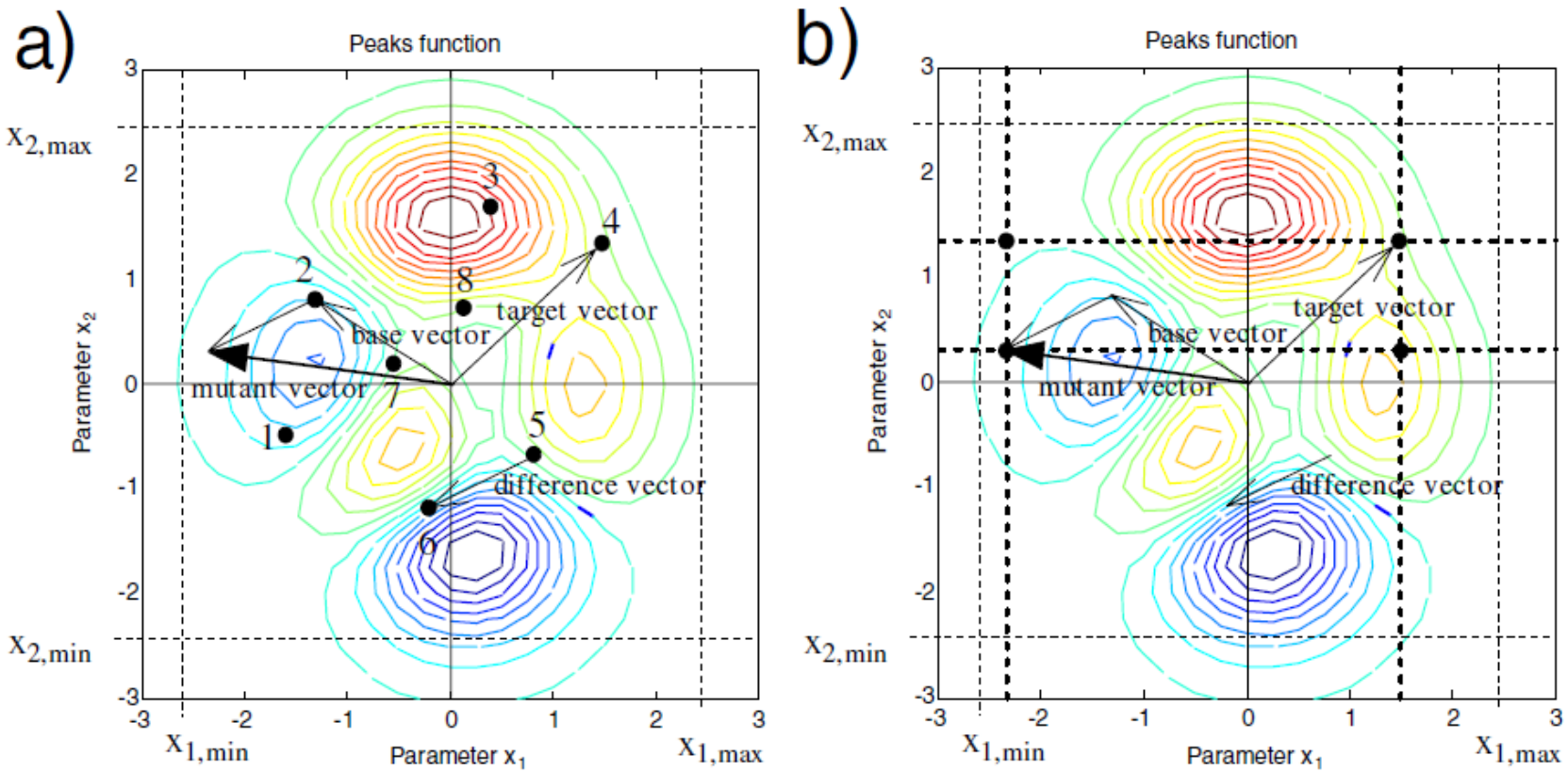
Rainer Storn (2008) Differential Evolution Research – Trends and Open Questions. Chapter 1 of Uday K. Chakraborty: Advances in Differential Evolution

Differential Evolution



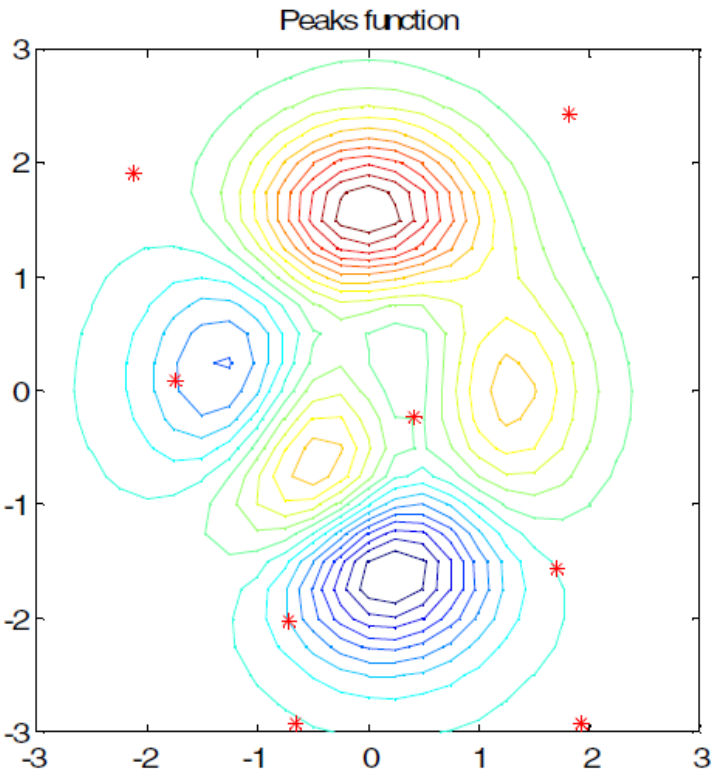
Effect of the factor F. Varying F is also used to dither (randomly mutate) the solution in the direction of the difference vector

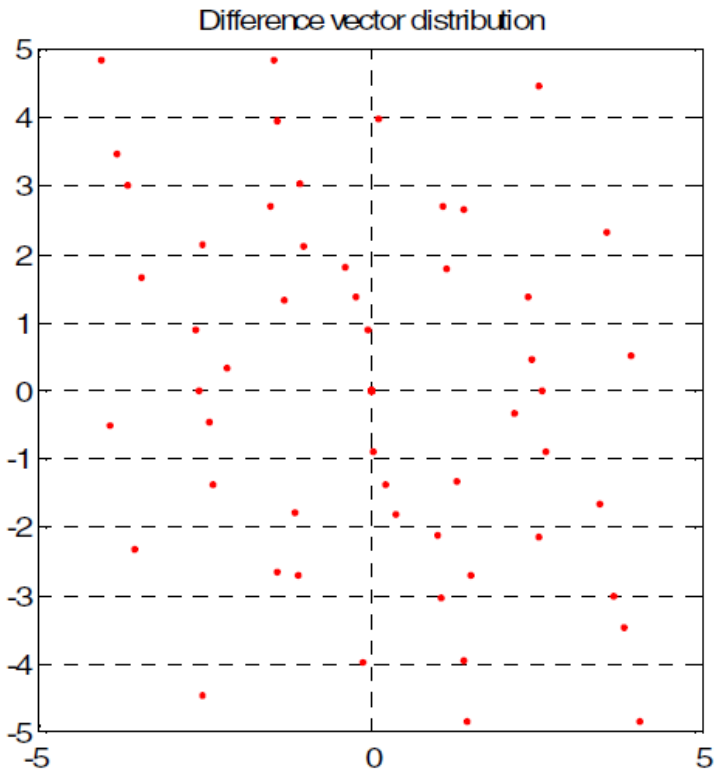
DE: Mutation and Crossover



Example for a population of $N = 8$ points and a mutation step a). The figure on the right b) shows the potential points when using crossover.

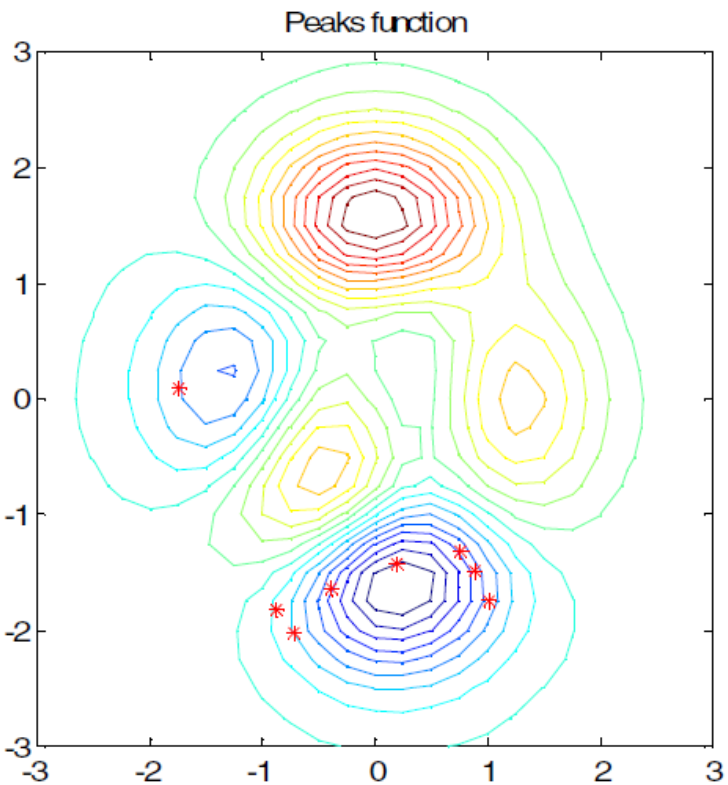
DE: Example

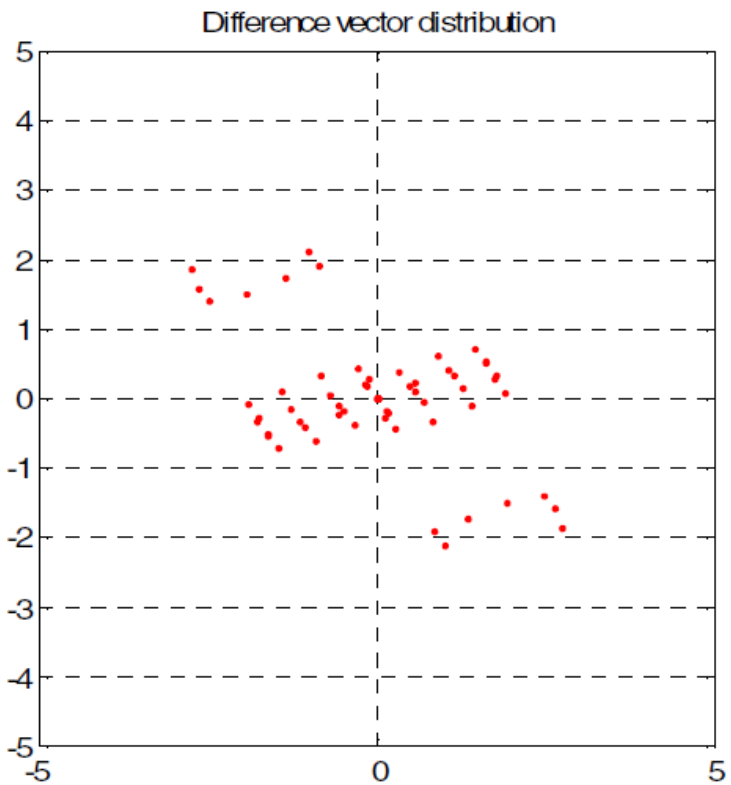




Generation $t = 1$ using $N = 8$

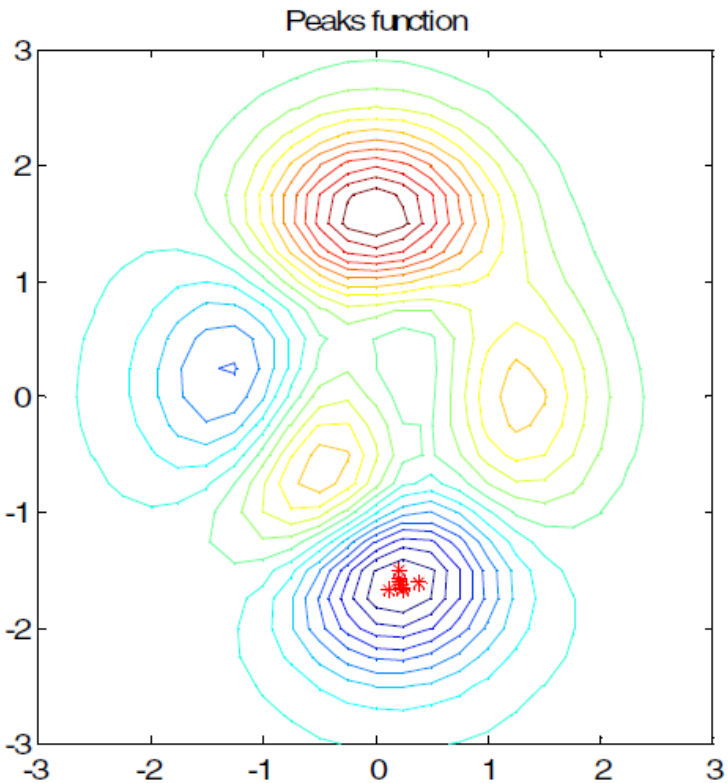
DE: Example

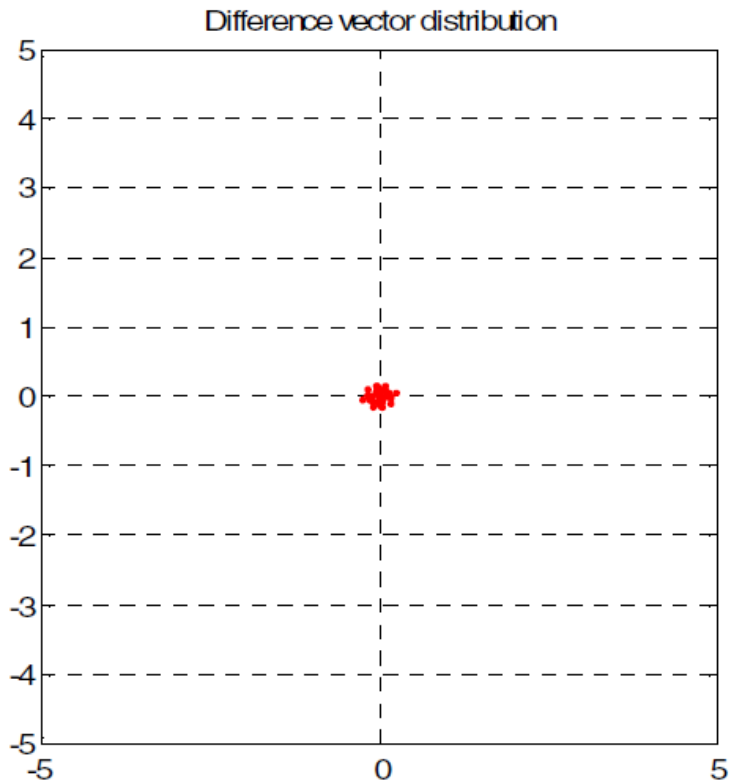




Generation $t = 10$ using $N = 8$. The difference vector distribution (only endpoints shown) exhibits three main clouds where the outer ones promote the transfer between two basins of attraction.

DE: Example





Generation $t = 20$ using $N = 8$. Now the difference vector distribution fosters the local search of the minimum the vector population is enclosing.

Differential Evolution: Variants

- Mutability and threshold parameters can also be evolved for each individual (as the step sizes in ES), i.e. dimension becomes $D + 2$.
- Various methods for diversification of the population: Jitter, dither, mixing perturbations, mirroring (opposition), ...
- Scheme for denoting DE variants:
  - x specifies the vector to be mutated which currently can be "*rand*" (a randomly chosen population vector) or "*best*" (the vector of highest fitness from the current population)
  - y is the number of difference vectors used
  - z denotes the crossover scheme. The current variant is "bin" (crossover due to independent binomial experiments)
- e.g. DE/best/2/bin (previously we had DE/rand/1/bin)
  $v_i(t+1) = x_{best}(t) + f \cdot (x_p(t) + x_q(t) - x_r(t) - x_s(t))$

Also a number of self-adapting variants exist cf. [Storn, 2008]

ES and DE: Conclusion

- Metaheuristic search algorithms
- Adaptive parameters important
- Relations to Gaussian adaptation
- Both the advanced versions of ES and DE compare favourably to other metaheuristic algorithms (see e.g. www.lri.fr/~Hansen)
- Diversity of the population of solutions needs often to be maintained by problem-dependent strategies (Mutations in DE, complex populations in ES)
- Relation to particle swarm optimisation (PSO) [later]

See also www.scholarpedia.org/article/Evolution_strategies

Evolutionary algorithms

	genotype (encoding)	mutation/ crossover	phenotype (applied to)
Genetic algorithm	strings of binary or integer numbers	e.g. 1-point for either one with p_m, p_c	optimization or search of optimal solutions
Genetic programming	trees (can be represented as strings)	like GA plus additional operators	computer programs for a computational problem
Evolutionary programming	real-valued parameter vector	mutation with self-adaptive rates	parameters of a computer program with fixed structure
Evolution strategy	real-valued encoding	mutation with self-adaptive rates	optimization or search of optimal solutions

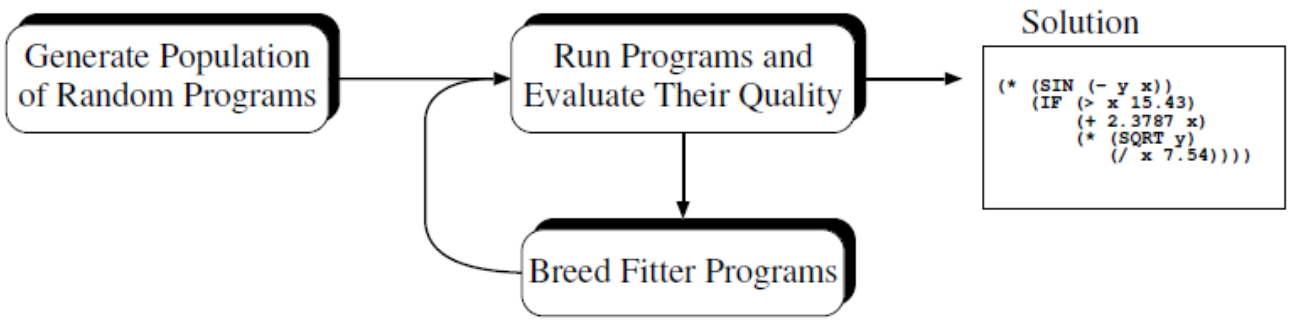
Genetic Programming

- Genetic programming now routinely delivers high-return human-competitive machine intelligence.
- Genetic programming is an automated invention machine.
- Genetic programming can automatically create a general solution to a problem in the form of a parametrized topology.
- Computer programs are the *lingua franca* for expressing the solutions to a wide variety of problems

Statements by John R. Koza et al. (2003)

Evolving Programs

- Is it possible to create computer programs by evolutionary means?
- Let $P(0)$ be a population of randomly generated programs p_i
- For each p_i, run it on some input and see what it does. Rate it for fitness based on how well it does.
- Breed the fitter members of $P(0)$ to produce $P(1)$
- If happy with the behaviour of the best program produced then stop.
- . . . but how?



Literature: Riccardo Poli, William B. Langdon, Nicholas F. McPhee (2008) A Field Guide to Genetic Programming (gp-field-guide.org.uk)

How?

- What language should the candidate programs be expressed in?
- C, Python, Java, Pascal, Perl, Lisp, Machine code?
- How can you generate an initial population?
- How can you run programs safely? Consider errors, infinite loops, etc.?
- How can you rate a program for fitness?
- Given two selected programs, how can they be bred to create offspring?
- What about subroutines, procedures, data types, encapsulation, etc.
- What about small, efficient programs?

Koza: Evolving LISP programs

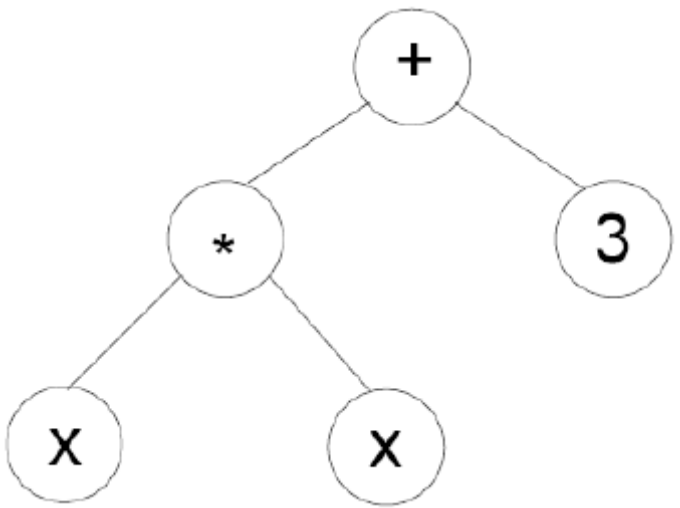
Lisp as a functional language

$f(x, y)$ is written as $(f\ x\ y)$

$10 - (3 + 4)$ is written as $(-\ 10\ (+\ 3\ 4))$

Lisp programs can be represented as trees

$f(x) = x^2 + 3$



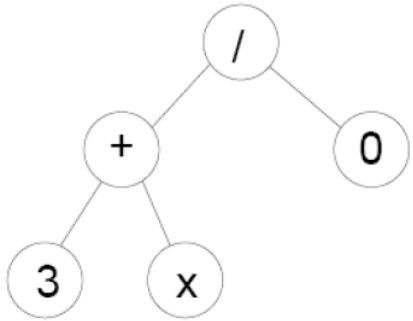
$f(x) = (+\ (*\ x\ x)\ 3)$

Here, $+$ and $*$ are function symbols (non-terminals) of arity, x and 3 are terminals. Given a random bag of terminals and non-terminals, we can make programs.

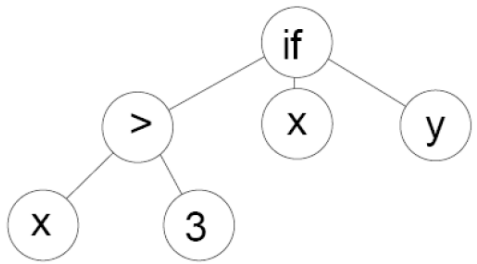
(Peter Seibel: Practical Common Lisp, 2004)

Random Programs and Closure

If we generate a random program:
How can we avoid an error?



Another random program
How can we evaluate this?



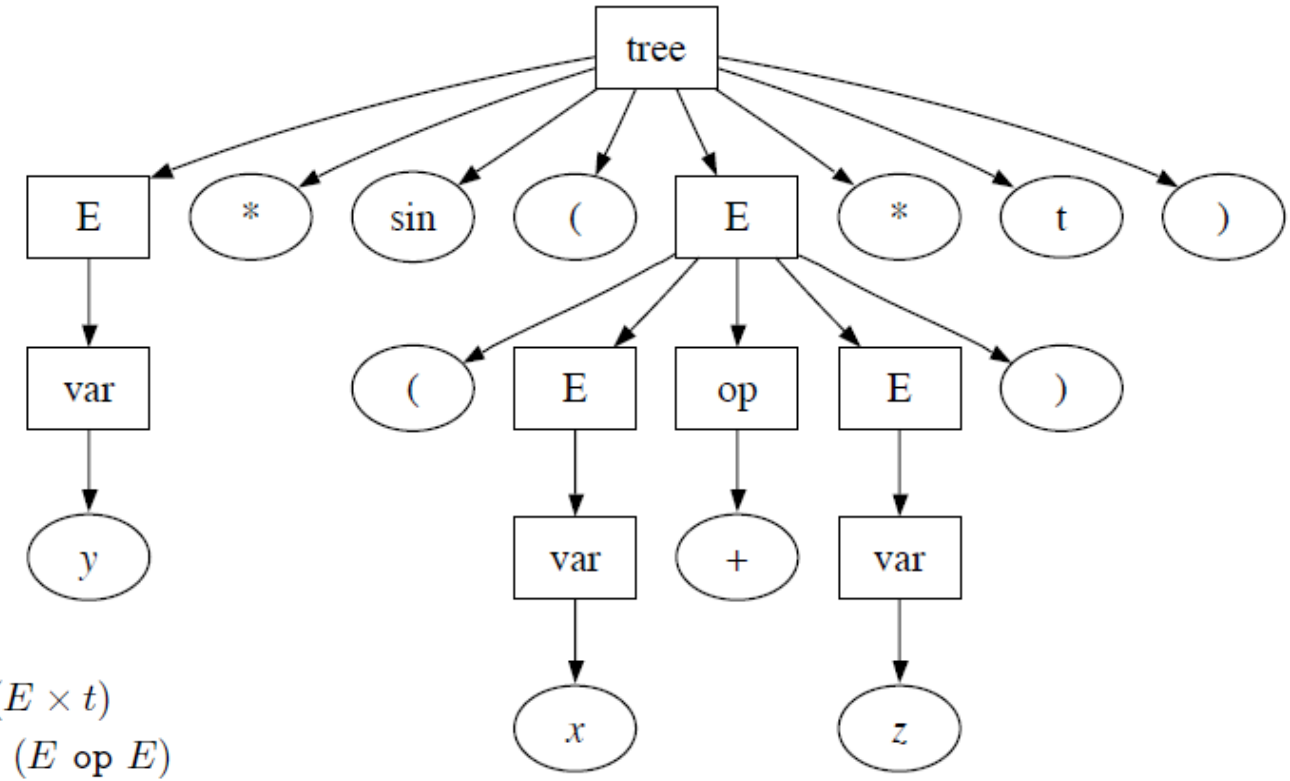
All function calls need to return a result

Closure: E.g. redefine division by zero to return FLT_MAX or zero; Overload operators to deal with variable numbers of arguments

Sufficiency: Set of nonterminals need to be sufficiently large, terminals need to be defined (if not given by data)

Grammar-Based Constraints (see GAs)

Typing or grammar-based approaches help to achieve closure



$$
\begin{array}{rcl}
\textit{tree} & ::= & E \times \sin(E \times t) \\
E & ::= & \texttt{var} \;\mid\; (E\ \texttt{op}\ E) \\
\texttt{op} & ::= & + \;\mid\; - \;\mid\; \times \;\mid\; \div \\
\texttt{var} & ::= & x \;\mid\; y \;\mid\; z
\end{array}
$$

Riccardo Poli, William B. Langdon, Nicholas F. McPhee (2008) A Field Guide to Genetic Programming

Fitness cases

How do we rate a program for fitness?

Answer: Run it on some "typical" input data for which we know what the output should be. The hope is the evolved program will work for all other cases (use **crossvalidation**!!).

Example: Symbolic regression on planetary orbits (Kepler's law). Given a set of values of independent and dependent variables, come up with a function that gives the values of the dependent variables in terms of the values of the independent variables.

Planet	A	P
Venus	0.72	0.61
Earth	1.00	1.00
Mars	1.52	1.84
Jupiter	5.20	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

Kepler's third law: Square of the period P of the planet proportional to cube of semimajor axis A, i.e. $P = A^{3/2}$.

Fitness Function

Given a number of example pairs (x_j, y_j) of data vectors x_j and desired outputs y_j, we could use the deviations of the generated program for an evaluation:

$$E_{\text{raw}} = \sum_j \| GP(x_j) - y_j \|^2$$

For a fitness function we could adjust

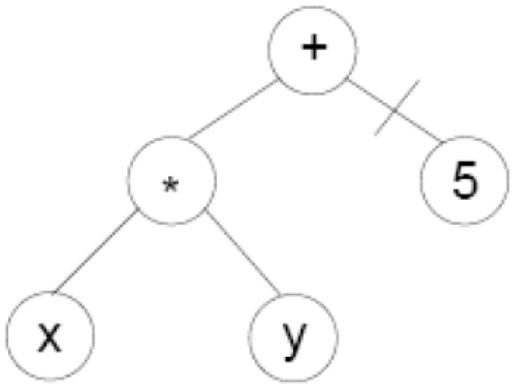
$$F_{\text{adj}} = \frac{1}{1 + E_{\text{raw}}}$$

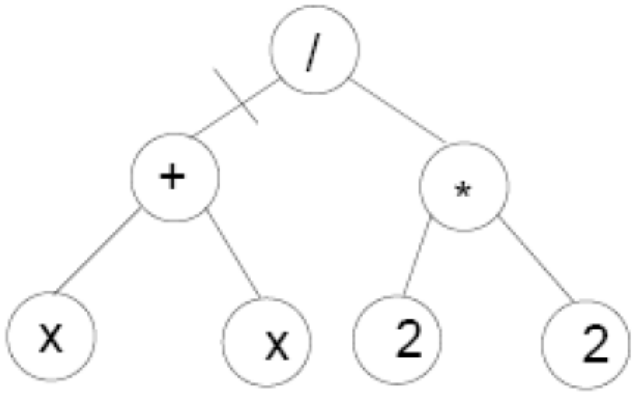
and normalise (or rank)

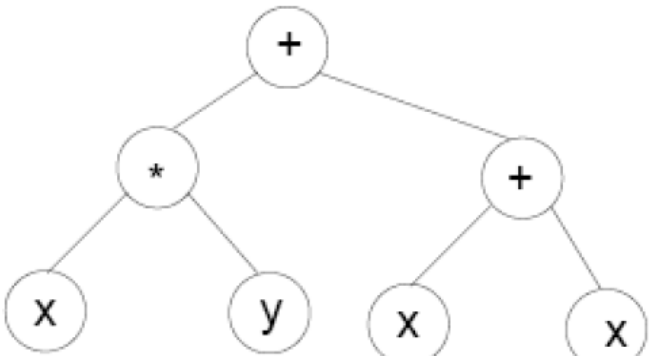
$$F_{\text{norm}}(i) = \frac{F_{\text{adj}}(i)}{\sum_k F_{\text{adj}}(k)}$$

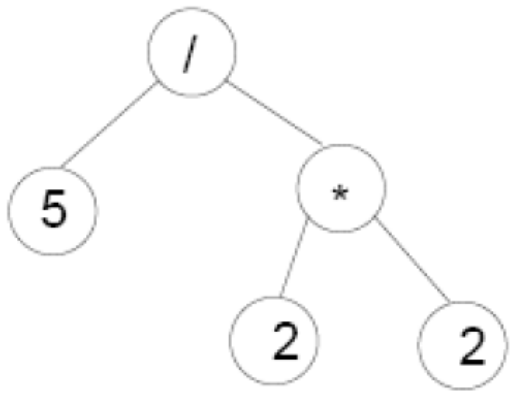
... so most fit is 1, least fit is 0 (i, j, k are the fitness cases)

Crossover









How can we cross programs?

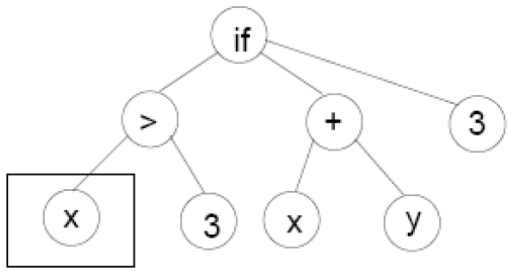
Subtree crossover

Koza's orginal (1988-92) GP system used only crossover, to try to demonstrate that GP is "more than a mutation".

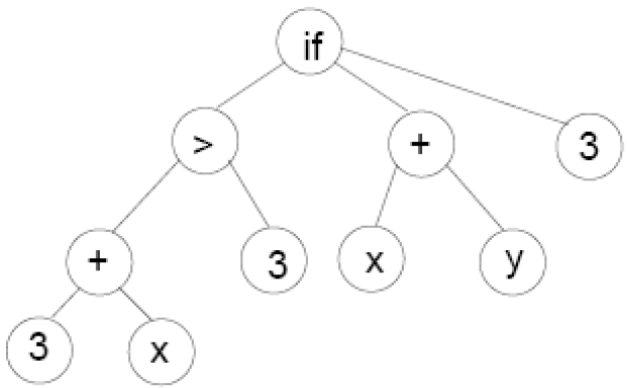
Mutation

How can we mutate a program?
Lots of other forms of mutation are possible, e.g. hoist, shrink

- shrink: replace a subtree by one of its terminals
- hoist: use only a subtree as a mutant



subtree or "grow"-mutation



or: vary numbers, exchange symbols, exchange subtrees, . . .

GP Algorithm

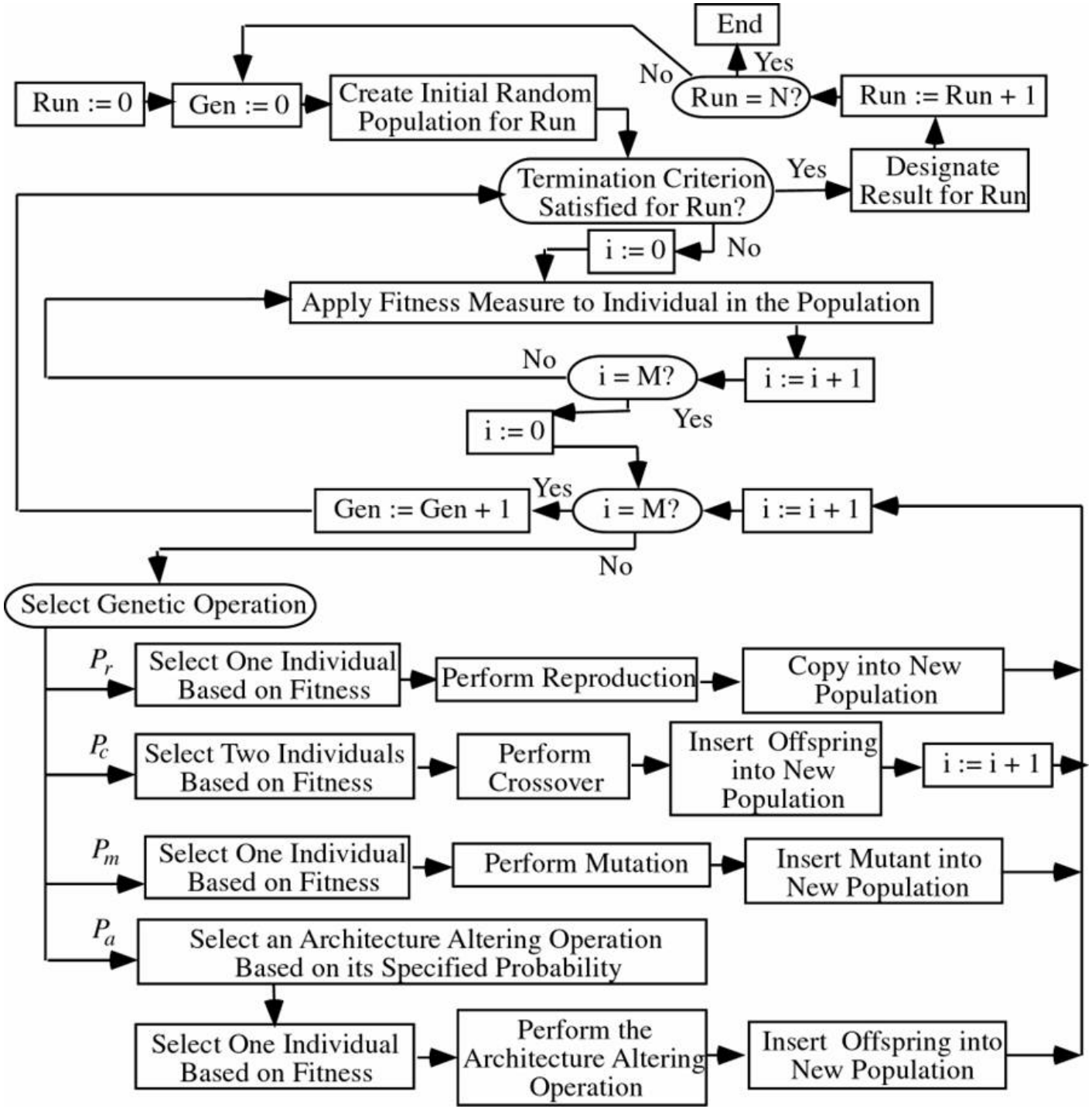
1. Choose a set of functions and terminals for the program you want to evolve:
   - non-terminals e.g.: if, /,* , +, −, sqrt, <, >...
   - terminals e.g.: x, y , $-10, -9, \ldots, 9, 10$
2. Generate an initial random population of trees of maximum depth d
3. Calculate the fitness of each program in the population using the chosen fitness cases.
4. Apply selection, subtree crossover (and subtree mutation) to form a new population.

Example parameter values:

Population size = 10000

Crossover rate = 0.9

Selection: Fitness proportionate


End
Yes
No
Run := 0
Gen := 0
Create Initial Random Population for Run
Run = N?
Run := Run + 1
Designate Result for Run
Termination Criterion Satisfied for Run?
Yes
i := 0
No
Apply Fitness Measure to Individual in the Population
No
i = M?
i := i + 1
Yes
i := 0
Yes
Gen := Gen + 1
i = M?
i := i + 1
No
Select Genetic Operation
P_r
Select One Individual Based on Fitness
Perform Reproduction
Copy into New Population
P_c
Select Two Individuals Based on Fitness
Perform Crossover
Insert Offspring into New Population
i := i + 1
P_m
Select One Individual Based on Fitness
Perform Mutation
Insert Mutant into New Population
P_a
Select an Architecture Altering Operation Based on its Specified Probability
Select One Individual Based on Fitness
Perform the Architecture Altering Operation
Insert Offspring into New Population

Example: GP for Symbolic Regression

Data:

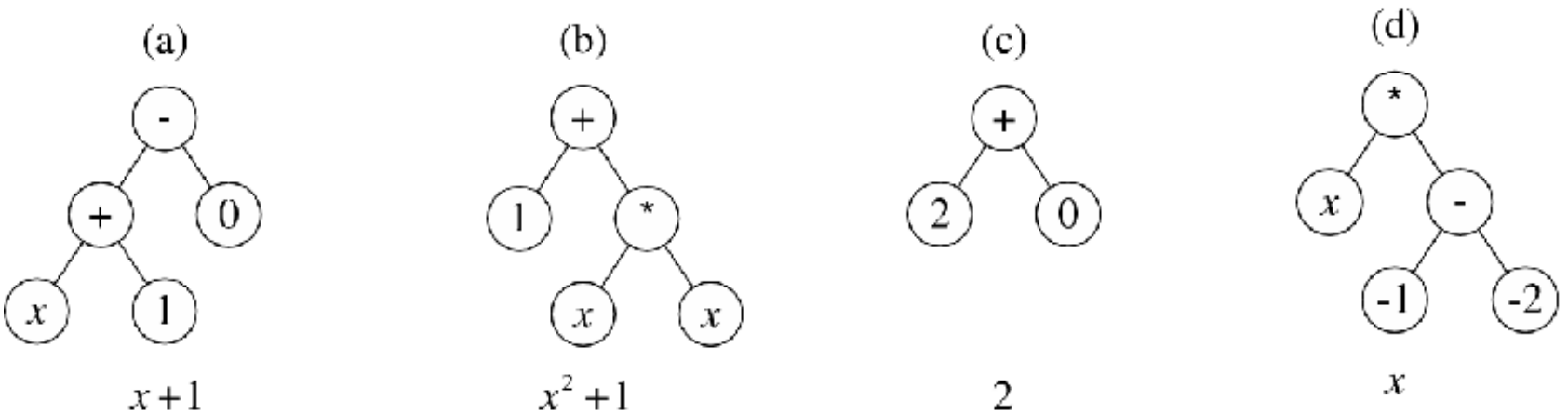
x	y
-1	1.00
-0.8	0.84
-0.6	0.76
-0.4	0.76
-0.2	0.84
0.0	1.00
0.2	1.24
0.4	1.56
0.6	1.96
0.8	2.44
1	3.00

Design:

	Objective:	**Find a computer program with one input (independent variable `X`) whose output equals the given data**
1	**Terminal set:**	`T = {X, Random-Constants}`
2	**Function set:**	`F = {+, -, *, %}`
3	**Fitness:**	**The sum of the absolute value of the differences between the candidate program's output and the given data (computed over numerous values of the independent variable *x* from –1.0 to +1.0)**
4	**Parameters:**	**Population size $M = 4$**
5	**Termination:**	**An individual emerges whose sum of absolute errors is less than 0.1**

From: J. R. Koza: GA and GP (tutorial)

Example: GP for Symbolic Regression

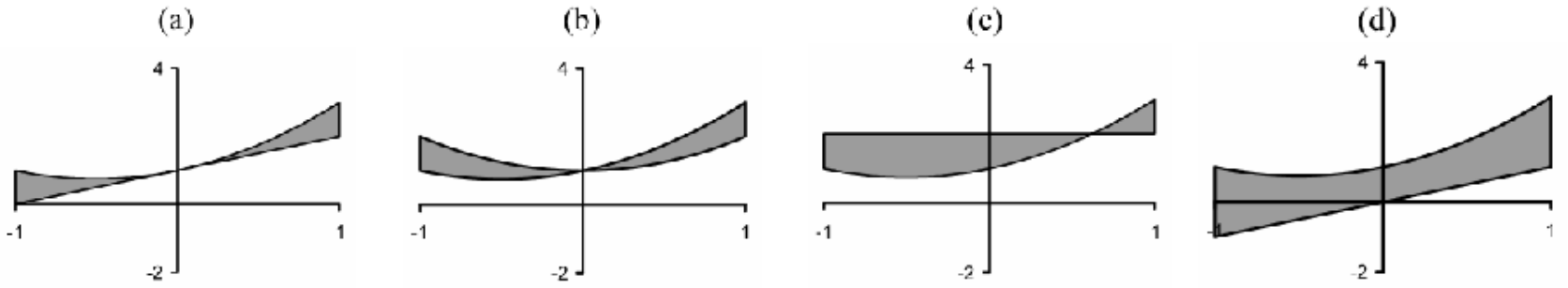


Population of 4 randomly generated individuals for generation 0

Example: GP for Symbolic Regression

Goal function: $f(x) = x^2 + x + 1$

Performance of the individual programs

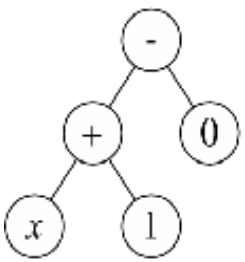


$x + 1$	$x^2 + 1$	2	x
0.67	1.00	1.70	2.67

The algorithm uses only the fitness calculated from the given data.

Example: GP for Symbolic Regression

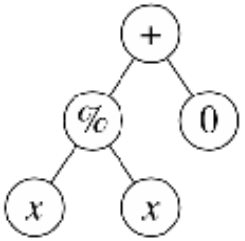
(a)



$x+1$

Copy of (a)

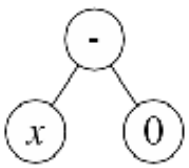
(b)



1

Mutant of (c) picking "2" as mutation point

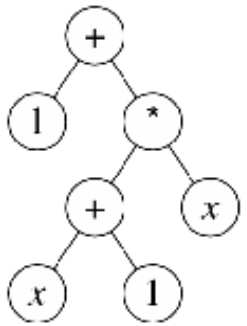
(c)



x

First offspring of crossover of (a) and (b) picking "+" of parent (a) and left-most "x" of parent (b) as crossover points

(d)



x^2+x+1

Second offspring of crossover of (a) and (b) picking "+" of parent (a) and left-most "x" of parent (b) as crossover points

GP: Application Areas

- Problem areas involving many variables that are interrelated in a non-linear or unknown way
- A good approximate solution is satisfactory
  - design, control, classification and pattern recognition, data mining, system identification and forecasting
- Discovery of the size and shape of the solution is a major part of the problem
- Areas where humans find it difficult to write
  - programs parallel computers, cellular automata, multi-agent strategies/distributed AI, FPGAs
- "Black art" problems
  - synthesis of topology and sizing of analog circuits, synthesis of topology and tuning of controllers, quantum computing circuits
- Areas where you simply have no idea how to program a solution, but where the objective (fitness measure) is clear
- Areas where large computerized databases are accumulating and computerized techniques are needed to analyze the data