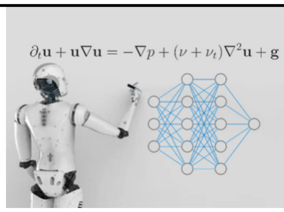


Hands on Machine Learning
for Fluid Dynamics

7 – 11 February 2022

von KARMAN INSTITUTE FOR FLUID DYNAMICS

Lecture 7
Genetic Programing

Dominique Joachim

Joachim.dominique@vki.ac.be

Table of Contents

1. Introduction to Genetic programing

2. Simple example on symbolic regression

3. Write the code using DEAP

Table of Contents

1. Introduction to Genetic programing

2. Simple example on symbolic regression

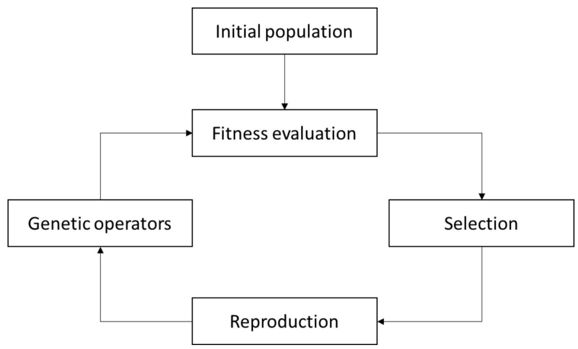
3. Write the code using DEAP

1.1 What is Genetic programing?

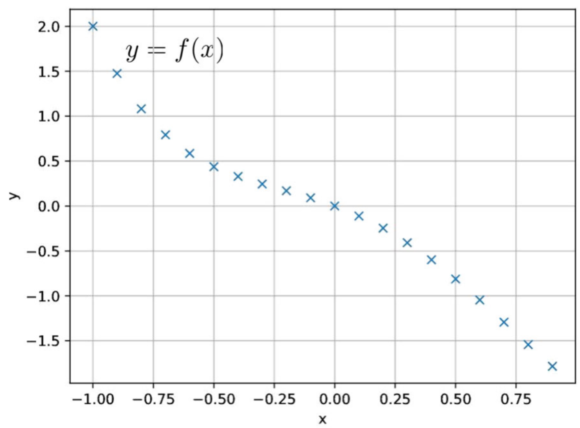
Genetic Programing **(GP)** belong to the wider class of **Genetic Algorithms.** It is an optimization algorithm that mimic natural selection by survival of the fittest.

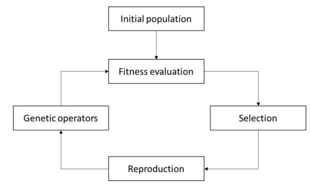
It uses a **population of individuals, select** the individuals according to **fitness** and introduce **genetic variation** using one or more **genetic operators.**

In **GP**, the individuals are **computer programs.**



1.2 Example





Using the GA framework

Objective of the Genetic Programing : discover $f(x)$

1.3 Syntax Tree/ Expression Tree

Primitive set

Function set = $[+, -, \times, \sqrt{}, /]$

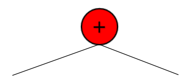
Terminal set = $[\mathrm{a,d,x,y,1.0}]$

1.3 Syntax Tree/ Expression Tree

Primitive set

Function set = $[+, -, \times, \sqrt{}, /]$

Terminal set = $[\mathrm{a,d,x,y,1.0}]$



2 inputs

Linear string

$\boxed{+}$

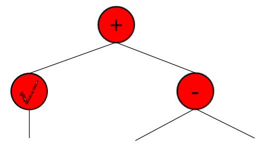
2 inputs

1.3 Syntax Tree/ Expression Tree

Primitive set

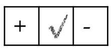
Function set = $[+, -, \times, \sqrt{}, /]$

Terminal set = $[a, d, x, y, 1.0]$



3 inputs

Linear string



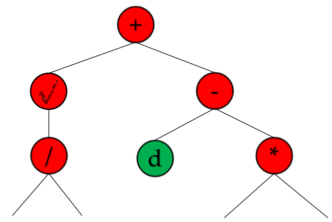
3 inputs

1.3 Syntax Tree/ Expression Tree

Primitive set

Function set = $[+, -, \times, \surd, /]$

Terminal set = [a,d,x,y,1.0]



4 inputs

Linear string

+	√	-	/	d	×

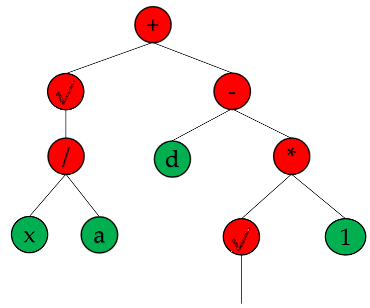
4 inputs

1.3 Syntax Tree/ Expression Tree

Primitive set

Function set = $[+, -, \times, \surd, /]$

Terminal set = [a,d,x,y,1.0]



1 inputs

Linear string

+	√	-	/	d	×	x	a	√	1

1 inputs

1.3 Syntax Tree/ Expression Tree

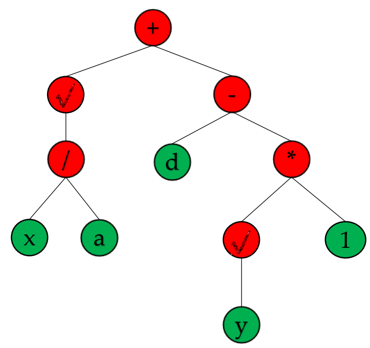
Primitive set

Function set = $[+, -, \times, \surd, /]$

Terminal set = $[\mathrm{a,d,x,y,1.0}]$

Linear string

+	√	-	/	d	×	x	a	√	1	y



1.3 Syntax Tree/ Expression Tree

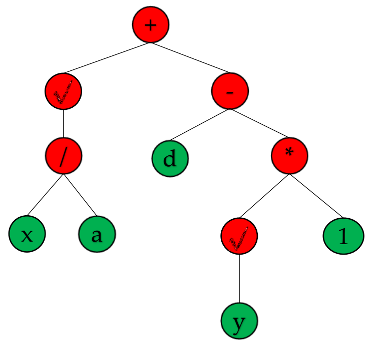
Primitive set

Function set = $[+, -, \times, \surd, /]$

Terminal set = $[a, d, x, y, 1.0]$

Linear string

+	√	-	/	d	×	x	a	√	1	y



add(root(div(x,a)),sub(d,mul(root(y),1.0)))

$$f_{ind}(x, y) = \sqrt{\frac{x}{a}} + d - \sqrt{y}$$

1.3 Syntax Tree/ Expression Tree

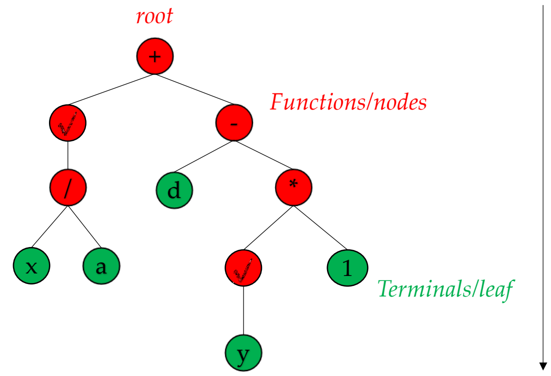
Primitive set

Function set = $[+, -, \times, \surd, /]$

Terminal set = $[a,d,x,y,1.0]$

Linear string

+	√	-	/	d	×	x	a	√	1	y



Depth = 5

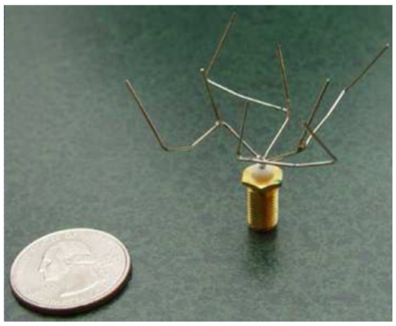
Length = 11

add(root(div(x,a)),sub(d,mul(root(y),1.0)))

$$f_{ind}(x,y) = \sqrt{\frac{x}{a}} + d - \sqrt{y}$$

1.3 Applications

Design NASA satellite antenna



Lohn, J. D., Hornby, G. S., & Linden, D. S. (2005). An evolved antenna for deployment on nasa's space technology 5 mission. In *Genetic Programming Theory and Practice II* (pp. 301-315). Springer, Boston, MA.

Propose Chess end game strategies

Hauptman, A., & Sipper, M. (2007, April). Evolution of an efficient search algorithm for the mate-in-N problem in chess. In *European Conference on Genetic Programming* (pp. 78-89). Springer, Berlin, Heidelberg.

1.3 Applications

- **Curve fitting and regression**
- Image and signal processing
- Financial trading, time series and economical modelling
- Industrial process control
- Medicine, Biology
- Entertainment, computer games
- Compression
- …

For more interesting applications check http://www.human-competitive.org/awards for winners of humies competition up to 5000 $ cash prize

Table of Contents

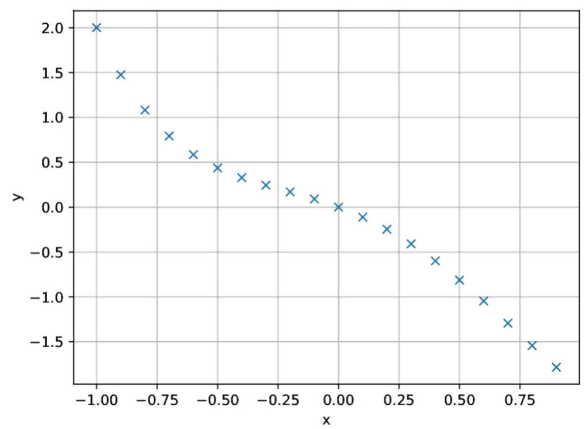
1. Introduction to Genetic programing

2. Simple example on symbolic regression

3. Write the code using DEAP

2.1 Simple symbolic regression problem

Objective function: $y = x^4 - x^3 - x^2 - x$



2.2 Define Primitive set

Objective function: $y = x^4 - x^3 - x^2 - x$

Function set = $[+, -, \times, /]$

Terminal set = $[\mathrm{x}, 1.0, ?]$

Closure :

- *Type consistency :* Any subtree crossover can mix and join nodes arbitrarily
- *Evaluation safety* : Function evaluation cannot fail

Sufficiency

- It is possible to express a solution to the problem at hand using the elements of the primitive set

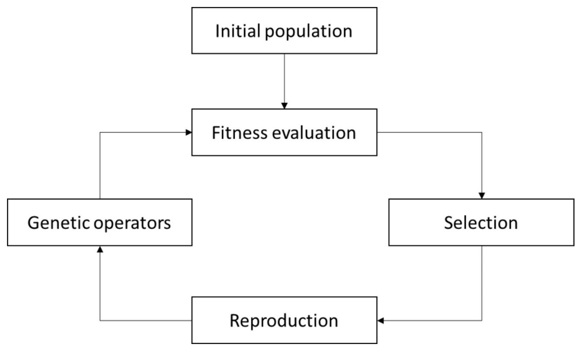
2.3 Chose the fitness function

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_{ind}(x_i) - y_i)^2$$

Fitness a measure of the quality of the program express as :

- **Error between output desired output**
- Amount of time to accomplish a task
- The accuracy of the task
- The payoff that a given strategy produce
- …

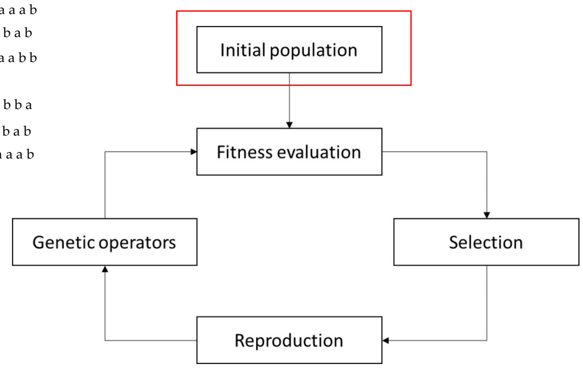
2.4 Set evolutionary algorithm parameters



Generation of a population of N individuals:

$\times N$

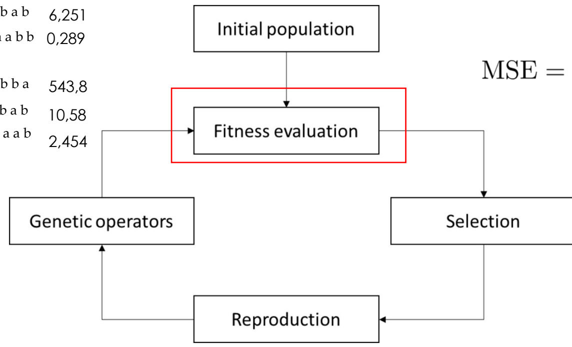
```
+Q-\b*aaQbaabaabbaaab
ab \b+ax+baabbbbaabab
xb aQ-aQ+aabbabbaaabb
...
\a+-Qba+baababababba
---ab*aa+babbabaaabab
+--\b*aQabaabaabbaaab
```



Generation of a population of N individuals:

$\times N$

	Fit:
+Q-\b*aaQbaabaabbaaab	0,125
ab \b+ax+baabbbbaabab	6,251
xb aQ-aQ+aabbabbaaabb	0,289
…	
\a+-Qba+baabababbba	543,8
---ab*aa+babbabaaabab	10,58
+--\b*aQabaabaabbaaab	2,454



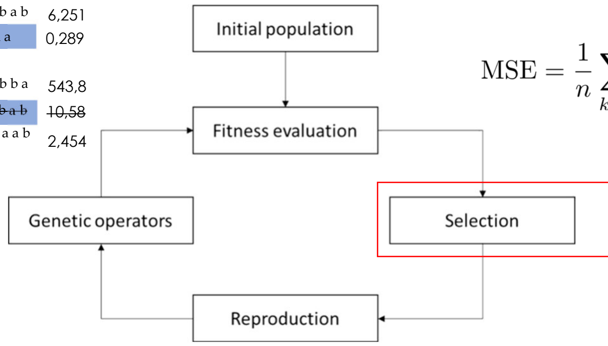
$$\text{MSE} = \frac{1}{n}\sum_{k=1}^{n}(f_{ind}(x_k, y_k) - f_k)^2$$

22

Generation of a population of N individuals:

$\times N$

	Fit:
+Q-\b*aaQbaabaabbaaab	0,125
ab \b+ax+baabbbbaabab	6,251
xb aQ-aQ+aabbabbaaa bb	0,289
…	
\a+-Qba+baababababba	543,8
~~---ab*aa+babbabaaabab~~	~~10,58~~
+--\b*aQabaabaabbaaab	2,454



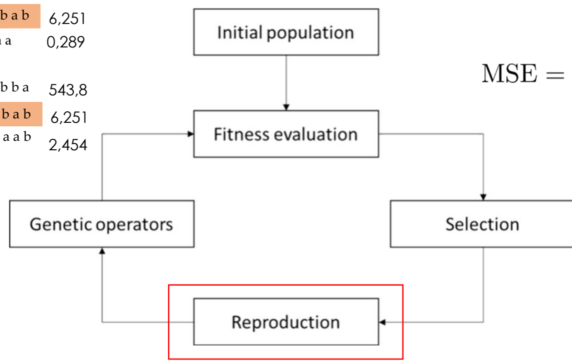
$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(f_{ind}(x_k, y_k) - f_k)^2$$

23

Generation of a population of N individuals:

$\times N$

	Fit:
+Q-\b*aaQbaabaabbaaab	0,125
ab \b+ax+baabbbbaabab	6,251
xb aQ-aQ+aabbabbaaa bb	0,289
…	
\a+-Qba+baabababababba	543,8
ab \b+ax+baabbbbaabab	6,251
+--\b*aQabaabaabbaaab	2,454



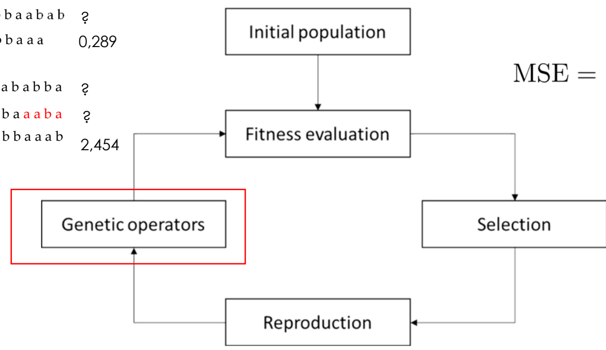
$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(f_{ind}(x_k, y_k) - f_k)^2$$

24

It is a non-deterministic process

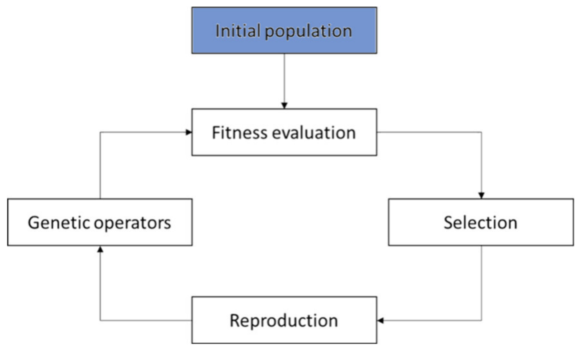
Generation of a population of N individuals:

$\times N$		Fit:
	+Q-\b*aaQbaabaabbaaab	0,125
	Qb \b+ax+baabbbbaabab	?
	xb aQ-aQ+aabbabbaaa bb	0,289
	…	
	\a+-Qba+bababbababba	?
	ab \b+ax+baabbbbaaaba	?
	+--\b*aQabaabaabbaaab	2,454



$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(f_{ind}(x_k, y_k) - f_k)^2$$

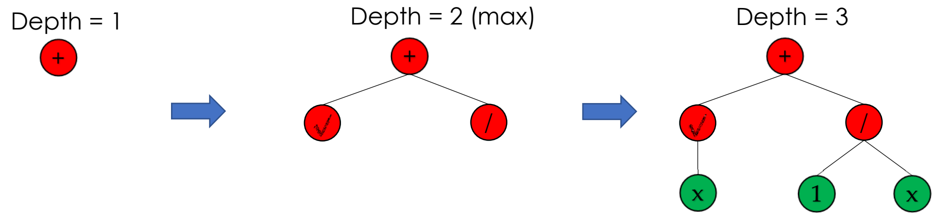
25


Initial population
Fitness evaluation
Selection
Reproduction
Genetic operators

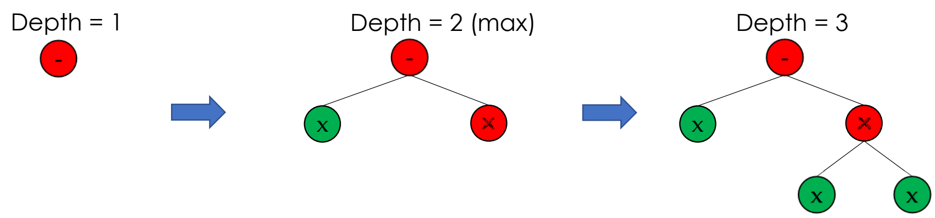
Slide 26

2.4.1 Initial population

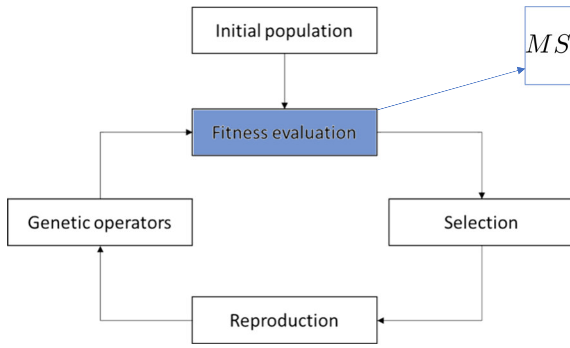
The full method : Nodes are chosen randomly in the function set until a maximum tree depth. Then the leafs are filled randomly with terminals



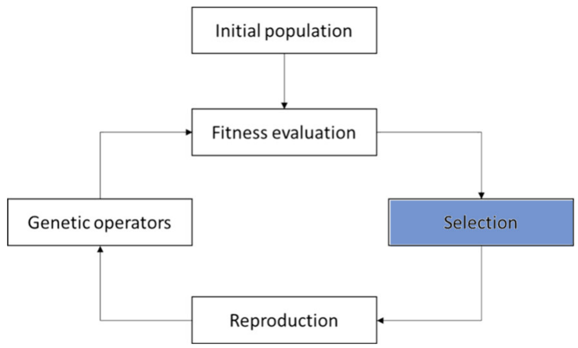
The grow method : Nodes are chosen randomly from the function and terminal set until a maximum tree depth. Then the leafs are filled randomly with terminals



The ramped half-half method : Half the population is generated with the full method and half with the grow method with a range of depth limit



$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(f_{ind}(x_i) - y_i\right)^2$$

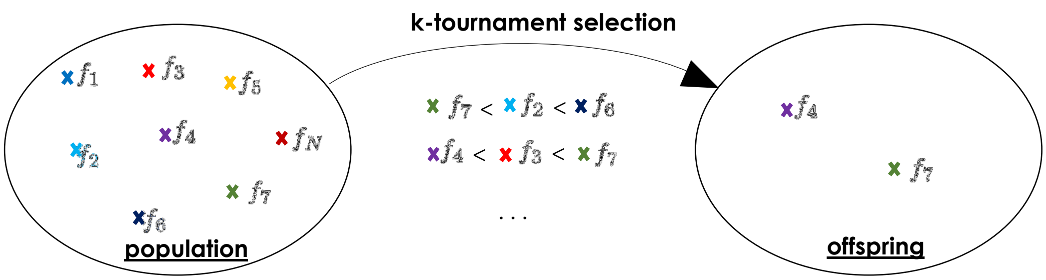

Initial population
Fitness evaluation
Genetic operators
Selection
Reproduction

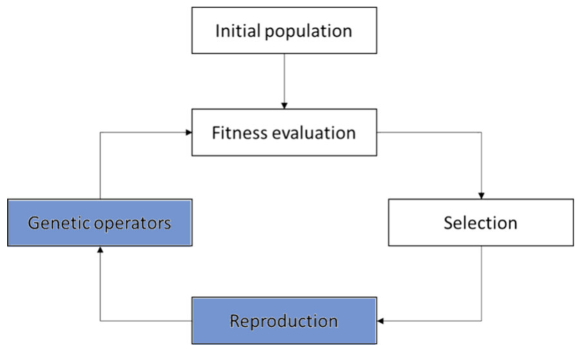
2.4.2 Selection

Fitness proportional : the selection probability of an individual is proportional to its fitness

Fitness based : the candidate has a probability of being selected that depends on his fitness

Competition based : the objectives values of individuals are compared to one another



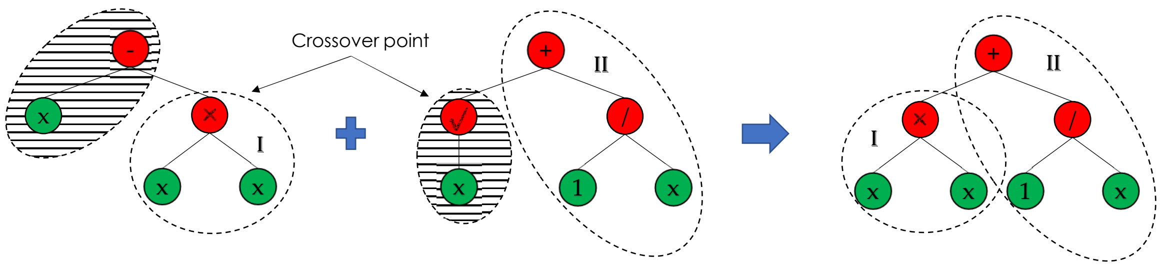

Initial population
Fitness evaluation
Genetic operators
Selection
Reproduction

Slide 31

2.4.3 Recombination and mutations

Crossover operators which use multiple individuals to generate a new one

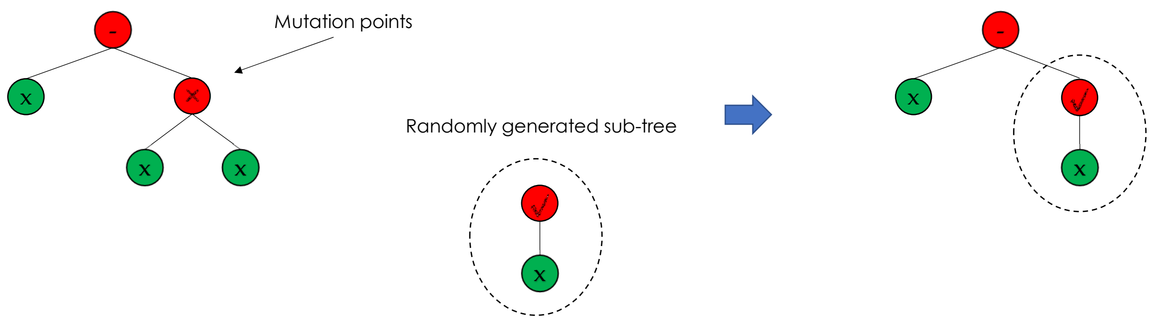
Subtree crossover : Two parents randomly select a crossover point. The branch after the crossover point of the second parent is replaced by the branch of the first parent to create a new individual



2.4.3 Recombination and mutations

mutation operators which modify an existing individual

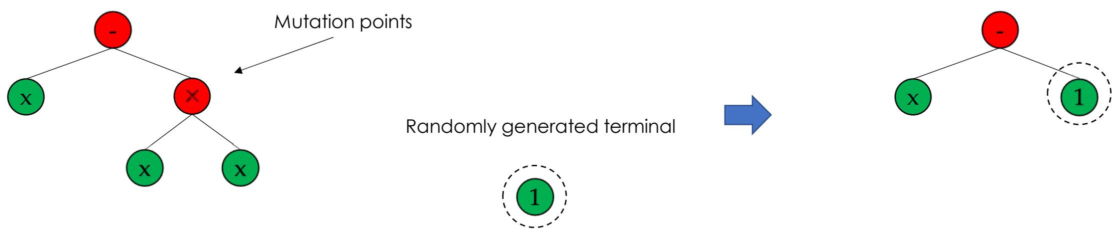
Point mutation: a random node is selected, and the branch is replaced by a randomly generated subtree



2.4.3 Recombination and mutations

Shrink or hoist mutation operators which purpose is to reduce the length of an existing individual

Shrink mutation: a random node is selected and the branch is replaced by a randomly selected terminal



2.4.3 Recombination and mutations

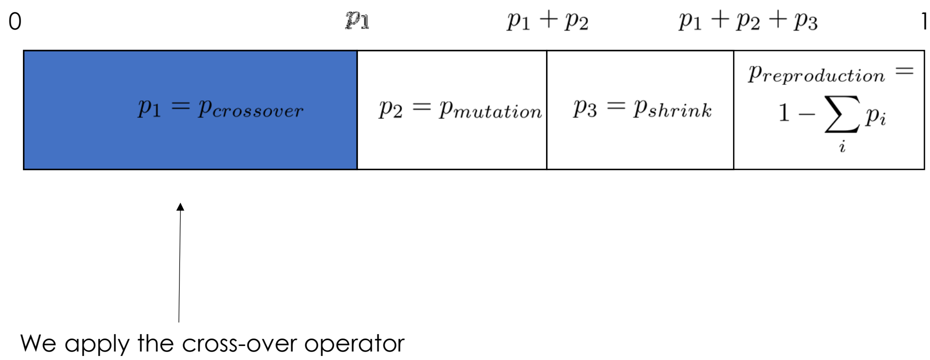
We apply only genetic operator per individual at each generation

0 p_1 $p_1 + p_2$ $p_1 + p_2 + p_3$ 1

$p_1 = p_{crossover}$	$p_2 = p_{mutation}$	$p_3 = p_{shrink}$	$p_{reproduction} = 1 - \sum_i p_i$

2.4.3 Recombination and mutations

We apply only genetic operator per individual at each generation



0 **< Random number generator <** p_1

2.4.3 Recombination and mutations

We apply only genetic operator per individual at each generation

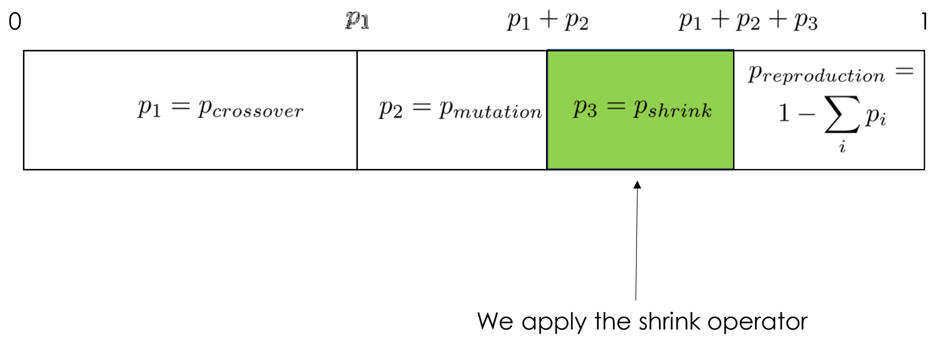
0	p_1	$p_1 + p_2$	$p_1 + p_2 + p_3$	1
$p_1 = p_{crossover}$	$p_2 = p_{mutation}$	$p_3 = p_{shrink}$	$p_{reproduction} = 1 - \sum_i p_i$	

We apply the mutation operator

p_1 **< Random number generator <** $p_1 + p_2$

2.4.3 Recombination and mutations

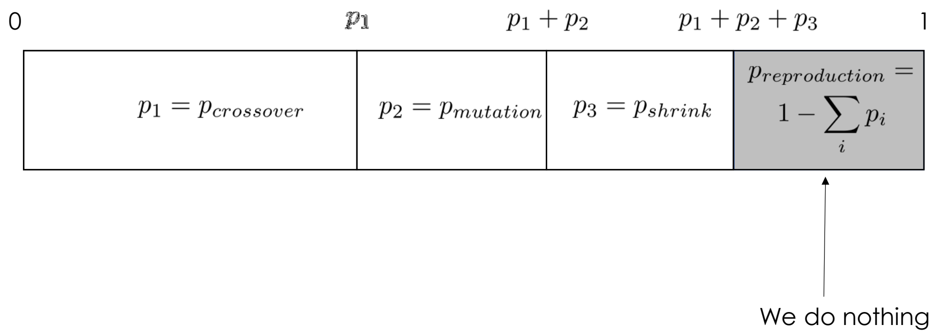
We apply only genetic operator per individual at each generation



$p_1 + p_2$ **< Random number generator <** $p_1 + p_2 + p_3$

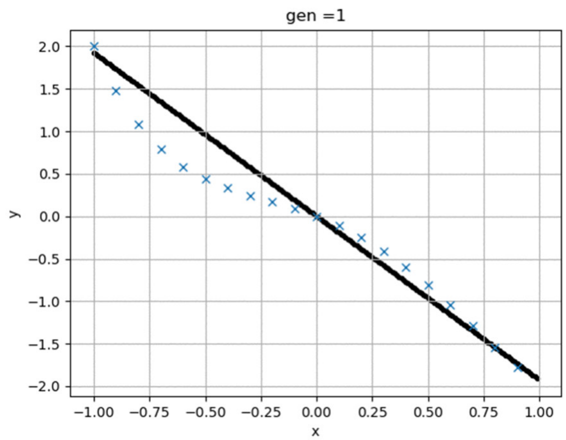
2.4.3 Recombination and mutations

We apply only genetic operator per individual at each generation



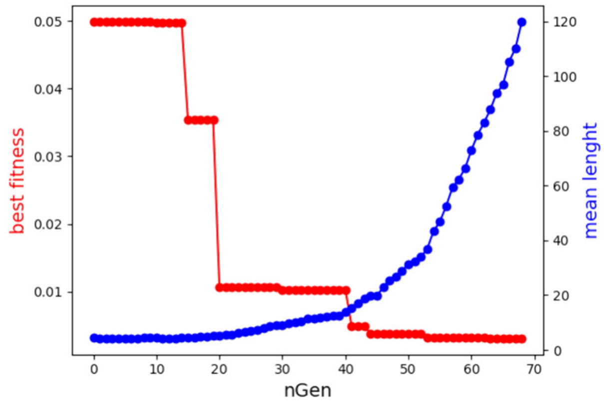
$p_1 + p_2 + p_3$ **< Random number generator <** 1.0

2.5 Let's try!


gen =1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

2.6 The bloat problem

It is normally defined as the increase in mean program size without a corresponding improvement in fitness.



2.6 Why does bloat exist ?

They are several theories for bloat :

1. *Nature of program search space theory :* the number of long program for a given fitness is greater than the number of short program
2. *The replication accuracy theory:* the success of an individual is based on its ability to have offspring that are functionally similar to its parents
3. *The removal bias theory :* Inactive modes are usually present in the low parts of the tree. Replacing an inactive nodes by an inactive subtree tends to increase the program length without improving its fitness
4. *And many more*

2.6 how to avoid bloat?

Bloat control strategies:

1. **Size and depth limits** on the offspring.

2. **Anti-bloat genetic operators**. Implement genetic operators that prevent tree from growing by design.

3. **Anti-bloat selection.** Add a penalty term on the fitness depending on the size of the program. This is the concept of the well-know parsimony pressure method where :

$$F(x) = f(x) - c \times l(x)$$

New fitness *Old fitness* *Penalty strength* *Individual length*

2.7 numerical constants

The creation of floating-point constants is necessary to do symbolic regression in evolutionary computation (Koza 1992)

1. Include numerical constants in the terminal set

Terminal set = $[x, 1.0, 2.0, 10.0, 2.5]$

2. Use ephemeral constants

Terminal set = $[x, \Re]$

Table of Contents

1. Introduction to Genetic programing

2. Simple example on symbolic regression

3. Write the code using DEAP

3.1 DEAP

DISTRIBUTED
EVOLUTIONARY
ALGORITHMS IN
PYTHON

DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent.

But they are other libraries available such as :

Genetic Programming in Python,
with a scikit-learn inspired API:
gplearn

Pyevolve
Genetic Algorithms

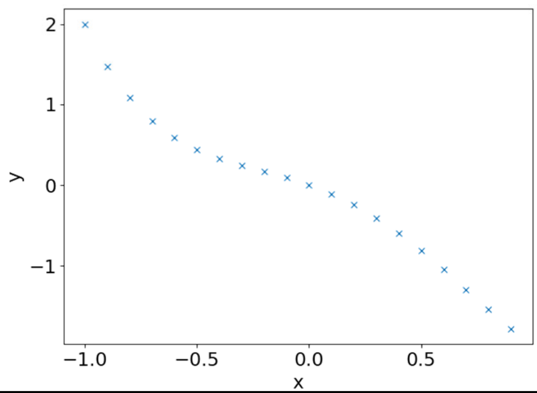
3.2 Checkout the coding tutorial video

3.3 Consider Today's dataset

```python
xpoints = [x/10. for x in range(-10,10)] #select 20 points between -1 and 1

def TrainFunction(x):
    y  = x**4 - x**3 - x**2 - x
    return y

#plot the training points
plt.figure()
plt.plot(np.array(xpoints),TrainFunction(np.array(xpoints)),marker='x',linestyle='')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



3.4 Define new sets of primitive

```
#########################################################################
# 1 - Chose primitive set (slide 10)
#########################################################################

# terminal set - one variable named x
pset = gp.PrimitiveSet("MAIN", 1)
pset.renameArguments(ARG0='x')

# terminal set - add ephemeral constant
name='rand'+str(random.randint(0, 10000))
pset.addEphemeralConstant(name,lambda: random.uniform(-1, 1))

# terminal set - add a constant of value 1.0
pset.addTerminal(1.0)

# function set - add operators
# multiplication, addition, soustraction, division

pset.addPrimitive(operator.add, 2)
```

Add the subtraction operator
Add the multiplication operator

Add the division operator

3.4 Define new sets of primitive

```
########################################################################
# 1 - Chose primitive set (slide 10)
########################################################################

# terminal set - one variable named x
pset = gp.PrimitiveSet("MAIN", 1)
pset.renameArguments(ARG0='x')

# terminal set - add ephemeral constant
name='rand'+str(random.randint(0, 10000))
pset.addEphemeralConstant(name,lambda: random.uniform(-1, 1))

# terminal set - add a constant of value 1.0
pset.addTerminal(1.0)

# function set - add operators
# multiplication, addition, soustraction, division

pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)

#attention the division needs to be protected to avoid division by 0
def protectedDiv(left, right):
    try:
        return left / right
    except ZeroDivisionError:
        return 1
pset.addPrimitive(protectedDiv, 2)
```

Add the subtraction operator

Add the multiplication operator

Add the division operator

3.5 Define new mutation operators

```
# Begin the generational process
for gen in range(1, Ngeneration + 1):
    # Select the next generation individuals
    select = toolbox.select(pop, len(pop)-1)
    # Select the best individual and save it in elite. This is elitism
    elites = tools.selBest(pop, k=1)
    save_elite.append(toolbox.clone(elites))

    #clone offspring
    off = [toolbox.clone(ind) for ind in select]

    # Apply mutation
    #this is to make sure we apply only one mutation to each individual (slide 18)
    cumpb = 
    for i in range(len(off)):
        pb = random.random()



        elif pb<cumpb[2]:
            off[i], = toolbox.mutate(off[i])
            del off[i].fitness.values
```

Change the cumulative probability

Add cross-over

Add shrinking

Add mutation of numerical constants

3.5 Define new mutation operators

```python
# Begin the generational process
for gen in range(1, Ngeneration + 1):
    # Select the next generation individuals
    select = toolbox.select(pop, len(pop)-1)
    # Select the best individual and save it in elite. This is elitism
    elites = tools.selBest(pop, k=1)
    save_elite.append(toolbox.clone(elites))

    #clone offspring
    off = [toolbox.clone(ind) for ind in select]

    # Apply mutation
    #this is to make sure we apply only one mutation to each individual (slide 18)
    cumpb = np.cumsum([cxpb,mutsh,mutpb,mut_eph])
    for i in range(len(off)):
        pb = random.random()
        if pb<cumpb[0] and i>0:
            off[i-1], off[i] = toolbox.mate(off[i-1],off[i])
            del off[i-1].fitness.values, off[i].fitness.values
            # print('off N=%i, crossover'%(i))
        elif pb<cumpb[1]:
            off[i], = toolbox.shrink(off[i])
            del off[i].fitness.values
            #print('off N=%i, schrink'%(i))
        elif pb<cumpb[2]:
            off[i], = toolbox.mutate(off[i])
            del off[i].fitness.values
        elif pb<cumpb[3]:
            off[i], = toolbox.mut_eph(off[i])
            del off[i].fitness.values
            #print('off N=%i, mutate'%(i))
        # else:
        #     print('off N=%i, reproduce'%(i))
```

Change the cumulative probability

Add cross-over

Add shrinking

Add mutation of numerical constants

3.6 Let's play with the meta parameters

```
# parameters for the GA
Npopulation = 1000 #size of the population
Ngeneration = 50 #number of generation
```

```
#selection tournament
# -> here you can change the selection process (slide 14)
toolbox.register("select", tools.selTournament, tournsize=2)# <- here

# -> here you can change the probability for the cross-over (slide 15)
toolbox.register("mate", gp.cxOnePoint)
cxpb = 0.4 # <- here

# -> here you can change the probability for the mutation (slide 16)
toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)
mutpb = 0.1 # <- here

# -> here you can change the probability for the shrink (slide 17)
toolbox.register("shrink", gp.mutShrink)
mutsh = 0.1 # <- here

# -> here you can change the probability for the mutation of ephemeral constants
toolbox.register("mut_eph", gp.mutEphemeral, mode='all')
mut_eph = 0.05 # <- here
```

Play with those parameters and try to obtain the best match to the data

Summary

1. Introduction to Genetic programing
2. Simple example on symbolic regression
3. Write the code using DEAP

Take Home Messages

Genetic and Swarm Intelligence algorithms are particularly interesting because…

☑ 1. Extremely simple coding (almost no mathematical background required!)

☑ 2. They can be easily parallelized (this is the TSC assignment!)

☑ 3. They are 'global optimizers' and can handle extremely complex cost functions

☑ 4. Easy generalization and no extra cost at higher dimensions

On the other hand, you should consider that…

☒ 1. Very Poor 'Sample Efficiency': many cost function evaluations are needed

☒ 2. The good setting of hyperparameters is strongly test case dependent

☒ 3. Performances are highly sensitive to hyperparameters (collaboration vs egoism!)

☒ 4. You might be looking for one good solution… not a population/swarm of solutions!

Take Home Messages

Genetic and Swarm Intelligence algorithms are particularly interesting because…

- ☑ 1. Extremely simple coding (almost no mathematical background required!)
- ☑ 2. They can be easily parallelized (this is the TSC assignment!)
- ☑ 3. They are 'global optimizers' and can handle extremely complex cost functions
- ☑ 4. Easy generalization and no extra cost at higher dimensions

Same as Lecture 6

On the other hand, you should consider that…

- ☒ 1. Very Poor 'Sample Efficiency': many cost function evaluations are needed
- ☒ 2. The good setting of hyperparameters is strongly test case dependent
- ☒ 3. Performances are highly sensitive to hyperparameters (collaboration vs egoism!)
- ☒ 4. You might be looking for one good solution… not a population/swarm of solutions!

Take Home Messages

Genetic and Swarm Intelligence algorithms are particularly interesting because…

1. Extremely simple coding (almost no mathematical background required!)
2. They can be easily parallelized (this is the TSC assignment!)
3. They are 'global optimizers' and can handle extremely complex cost functions
4. Easy generalization and no extra cost at higher dimensions
5. Output somewhat easily interpretable function

On the other hand, you should consider that…

1. Very Poor 'Sample Efficiency': many cost function evaluations are needed
2. The good setting of hyperparameters is strongly test case dependent
3. Performances are highly sensitive to hyperparameters (collaboration vs egoism!)
4. You might be looking for one good solution… not a population/swarm of solutions!
5. Bloat issue !!!

The end!

Slide 58