Genetic Programming

Nuno Antunes Ribeiro

Assistant Professor

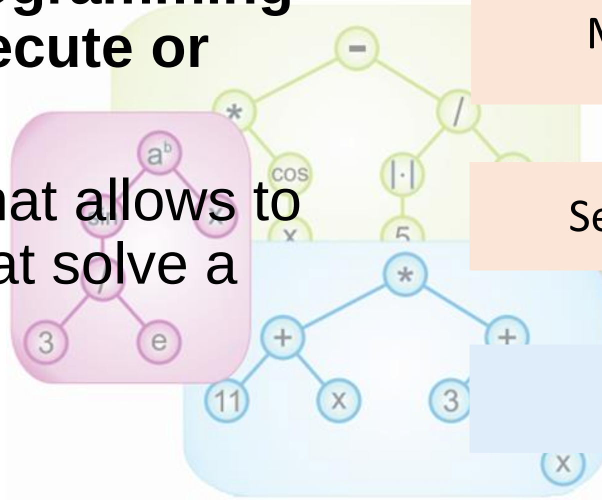
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Engineering Systems and Design

Genetic Programming

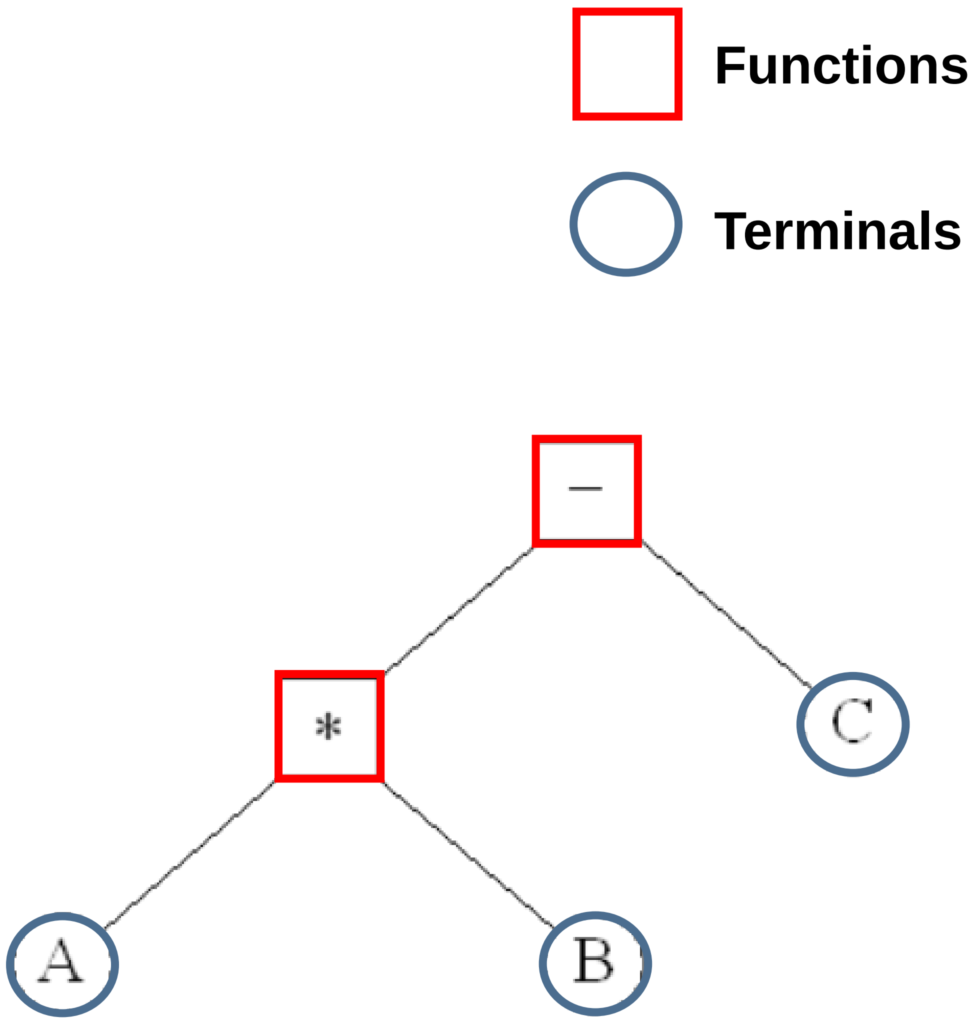
- GP was developed by Jonh Koza (a PhD student of Jonh Holland) around 1992.
- It is a more recent evolutionary approach, which extends the generic model of learning to the space of programs
- Its major variation, with respect to other evolutionary families, is that the evolving individuals are themselves **computer programs** (**sequence of instructions in a programming language that a computer can execute or interpret**)
- GP is a form of program induction that allows to automatically generate programs that solve a given task

Initialize Population
↓
Selection for Reproduction
↓
Crossover Operator
↓
Mutation Operator
↓
Selection for Survival
↓
Termination

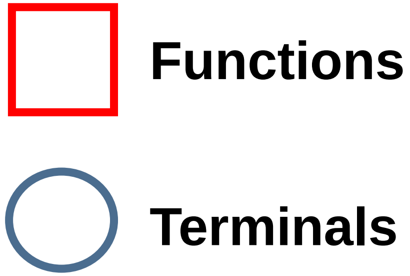


Tree-based Encoding

- **Any computer program is a sequence of operations (functions)** applied to values (arguments).
- **Tree encoding** is often used for hierarchical sequenced optimization problems. In tree encoding, a solution is represented by a tree of some operations /functions
- Computer programs can be coded as a hierarchical sequence of functions

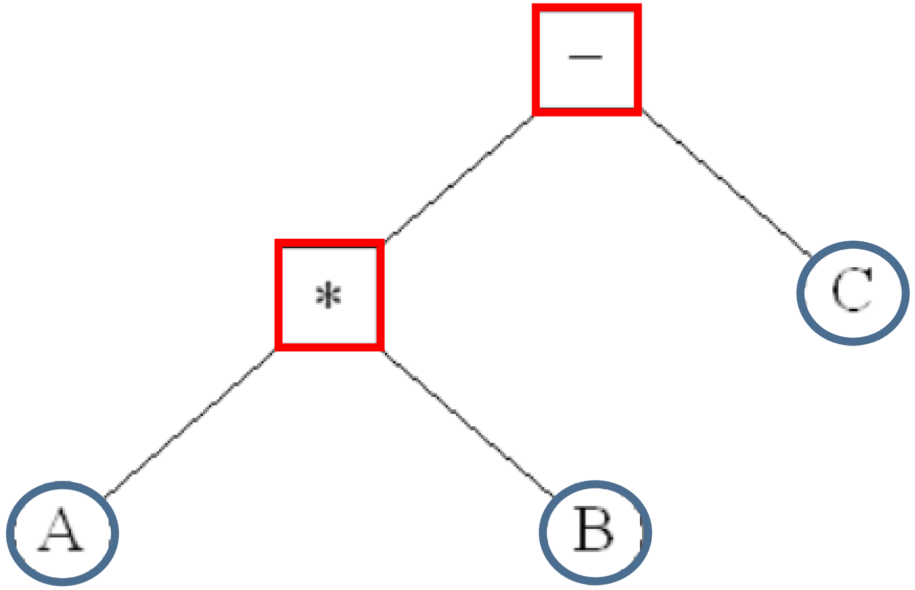


Tree-based Encoding



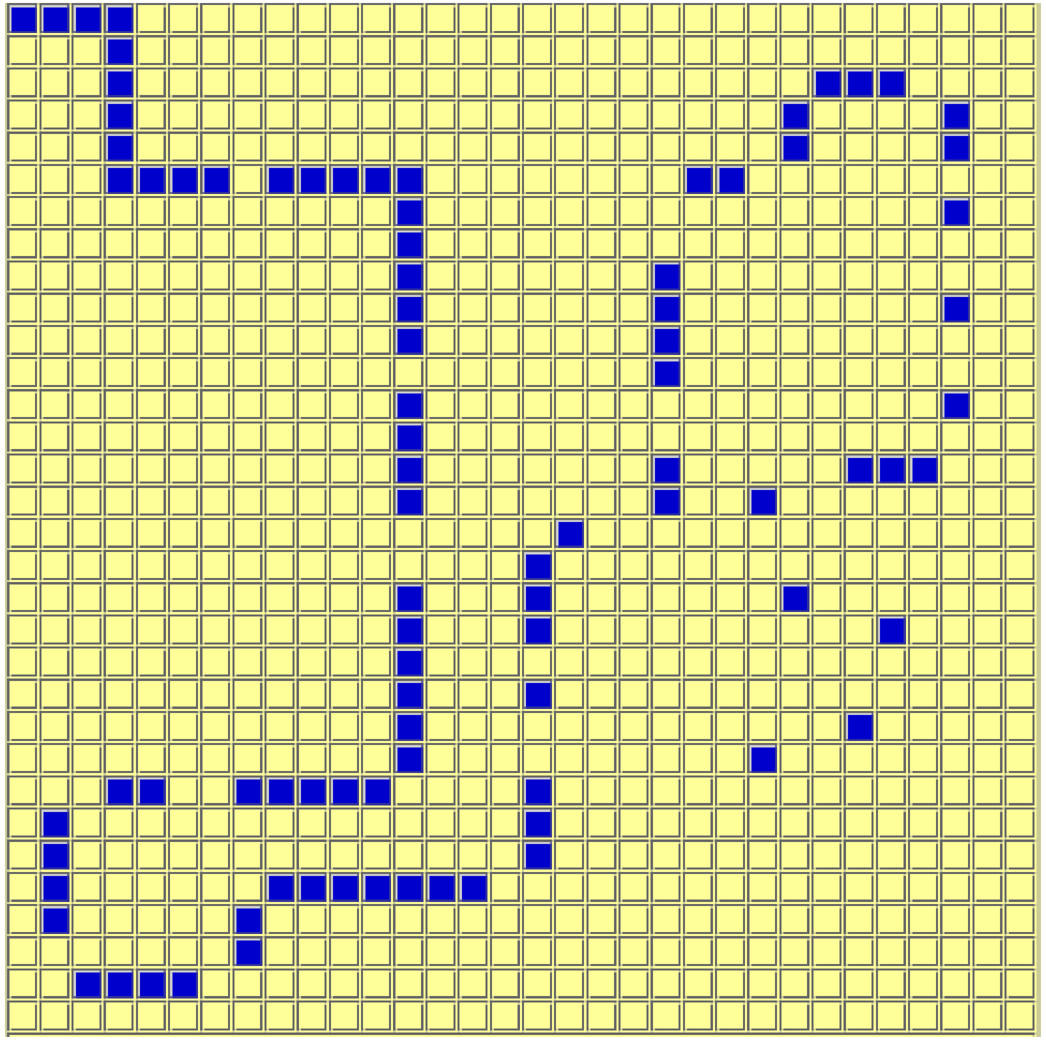
Example: **(- (* A B) C)** $\longrightarrow$ $(A \times B) - C$

- Calls for the application of the subtraction function (-) to two arguments, namely the list (*A B) and the atom C.
- But, first, the multiplication function (*) is applied to A and B.
- Once the list (*A B) is evaluated, the tree applies the subtraction function (-) to the two arguments, and thus evaluates the entire list



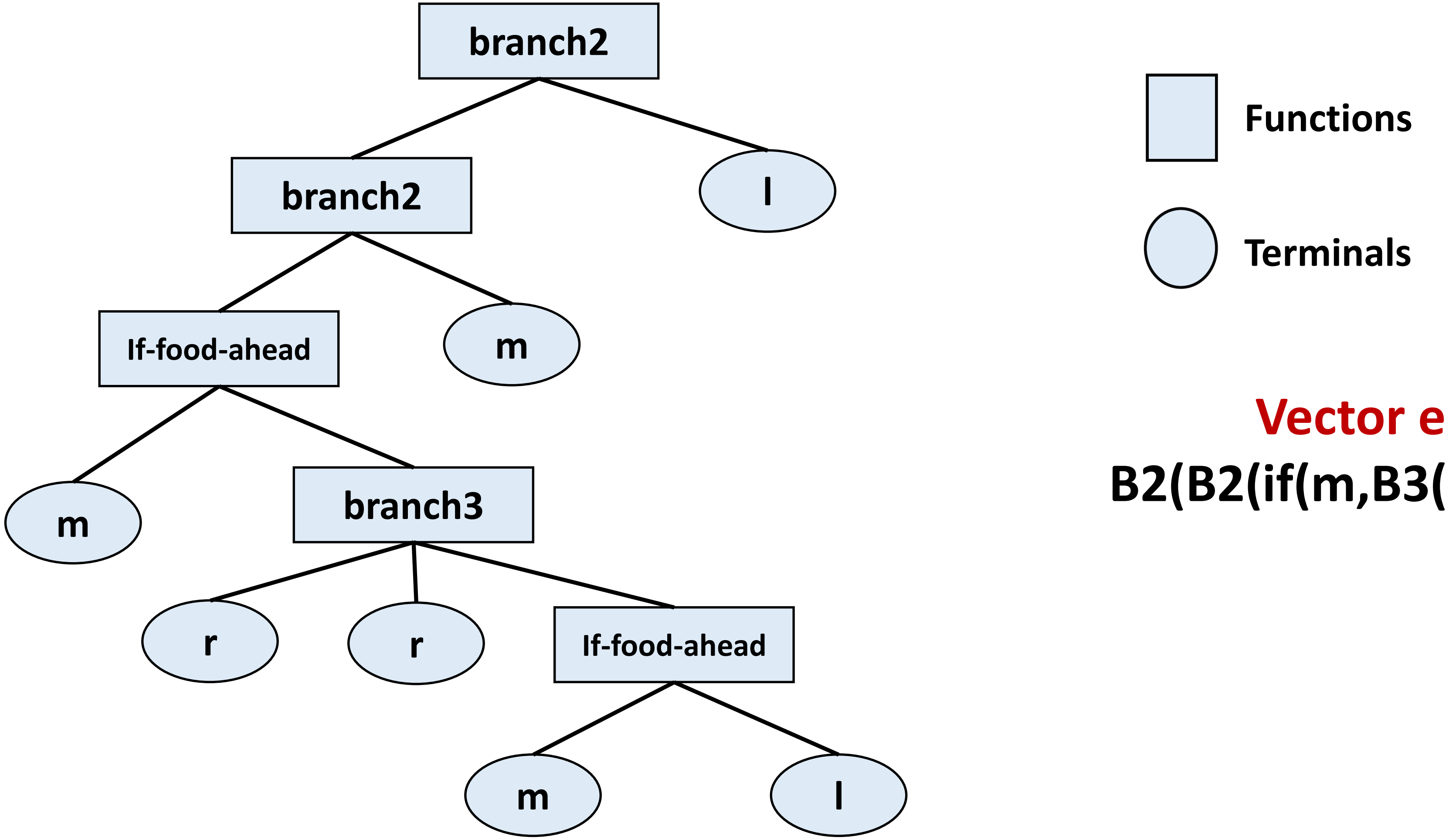
Santa Fe Ant Trail

- The Santa Fe Trail problem is a genetic programming exercise in which artificial ants search for food pellets according to a programmed set of instructions
- Fitness will be measured by the number of food pellets the ant encounters.
- The basic problem considers 6 operators (3 functions and 2 operators):
  - **If-food-ahead** (if); **branch2** (2 branches); **branch3** (3 branches);
  - Terminal nodes: **move straight** (m); **left** (l); **right** (r)



Random Tree

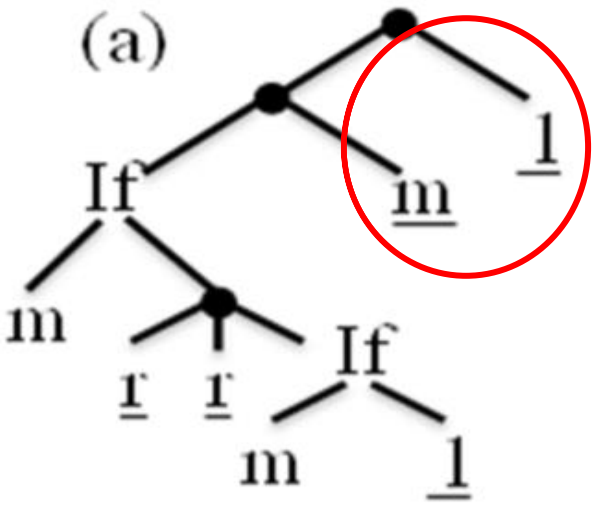
- **Let's start by generating a random tree for the problem.**



Vector encoding

B2(B2(if(m,B3(r,r,if(m,l)))m)l)

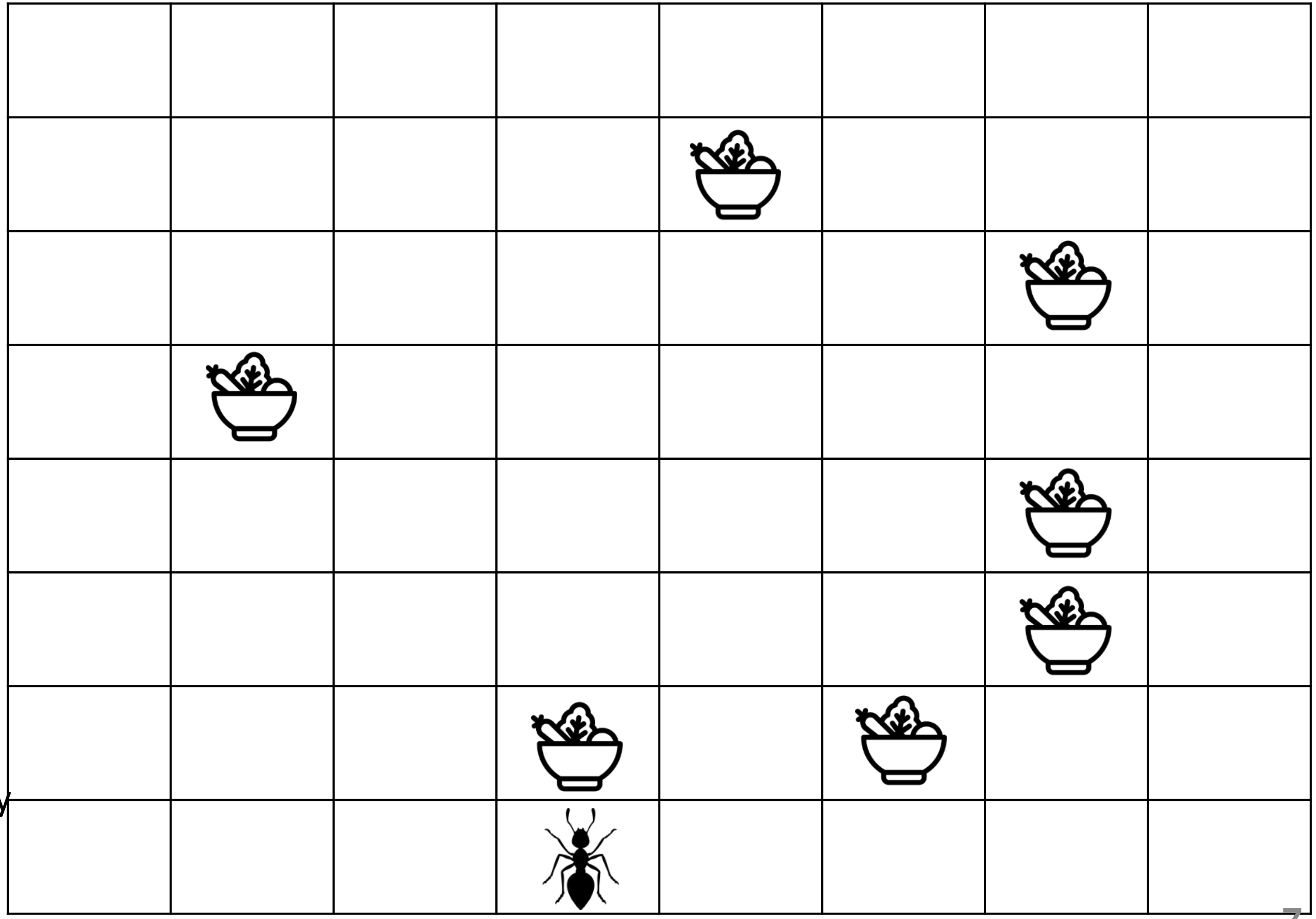
Santa Fe Ant Trail



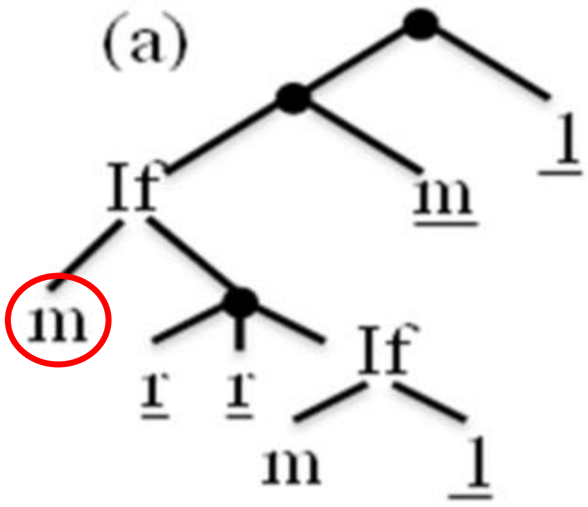
Move:

?ml

m and l moves are always applied. They are the last two moves to be applied in all iterations according to this program

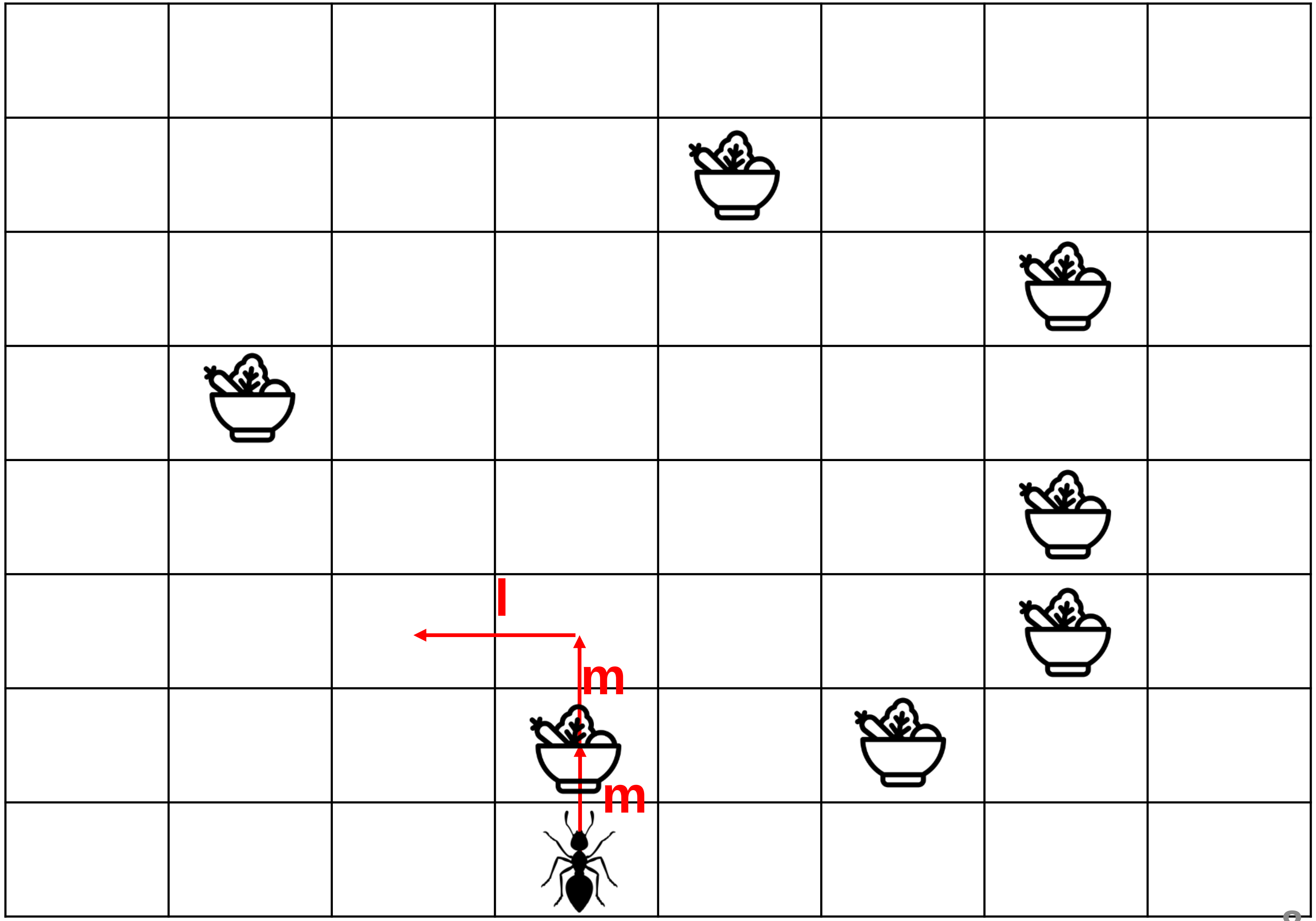


Santa Fe Ant Trail

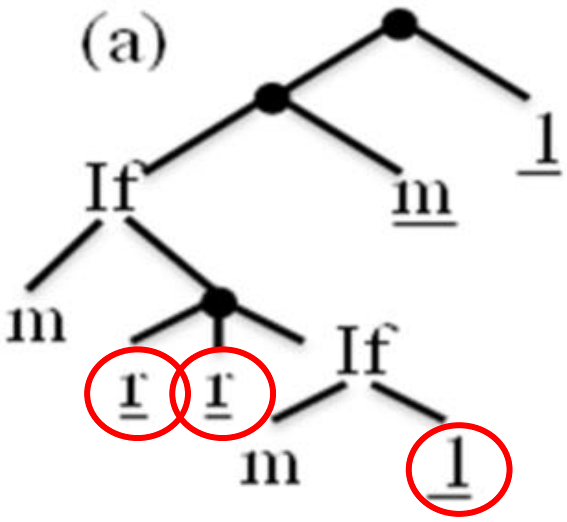


Move:

mml

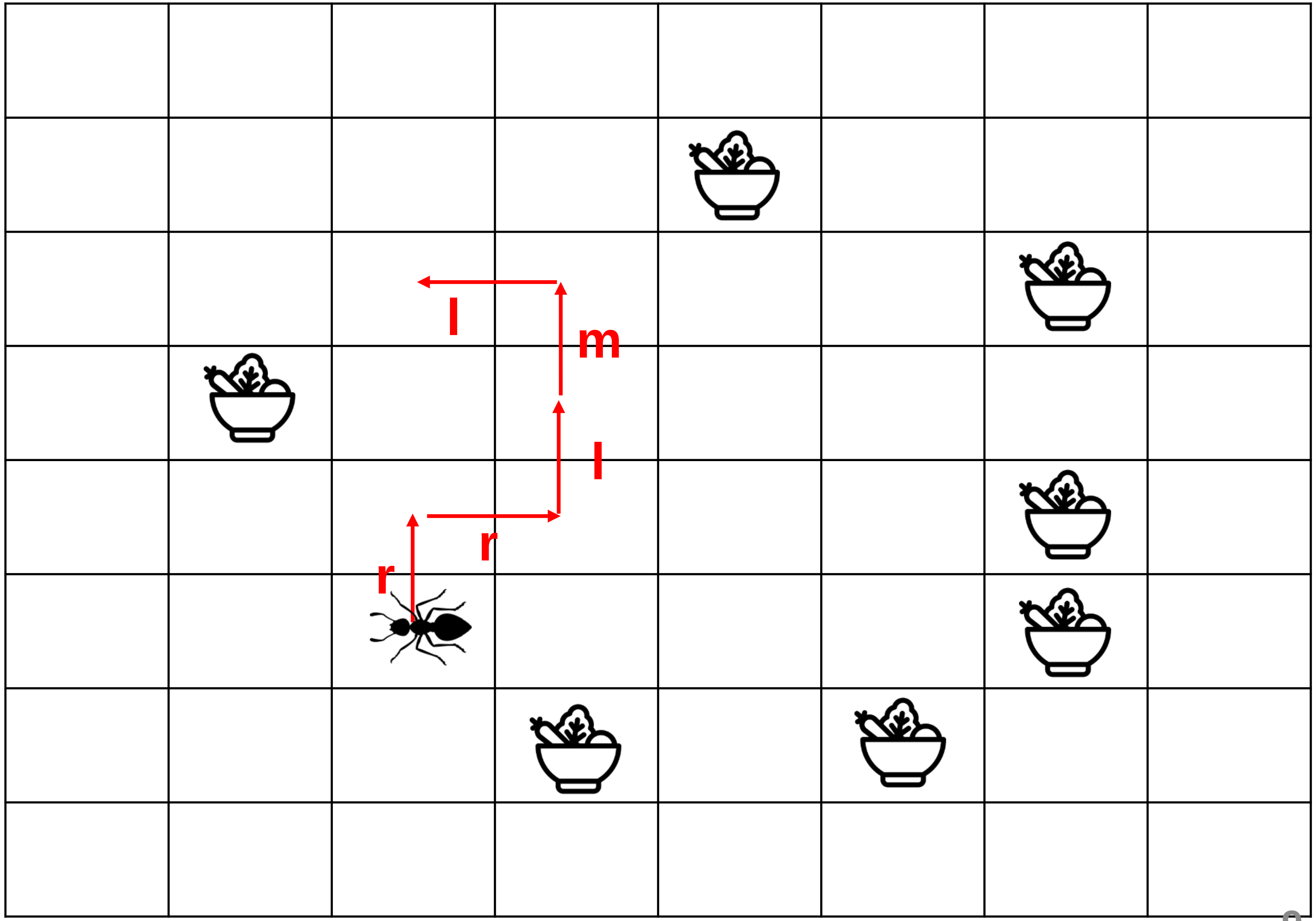


Santa Fe Ant Trail

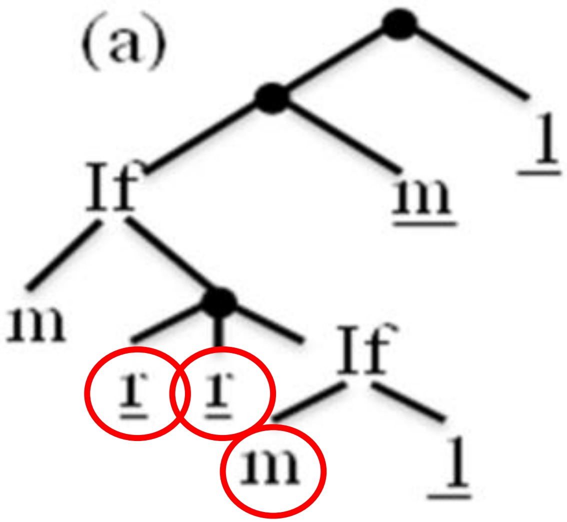


Move:

rrlml

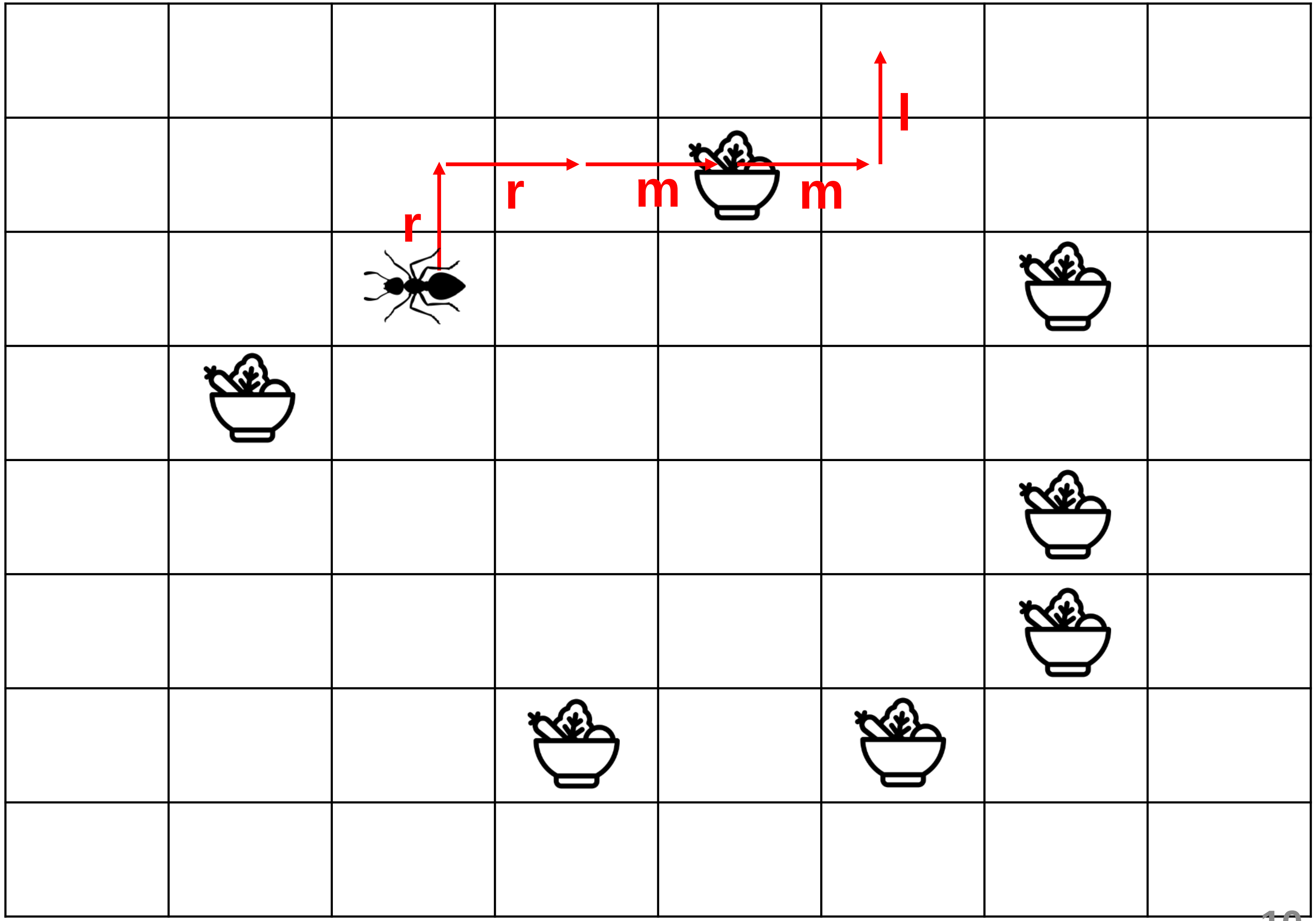


Santa Fe Ant Trail

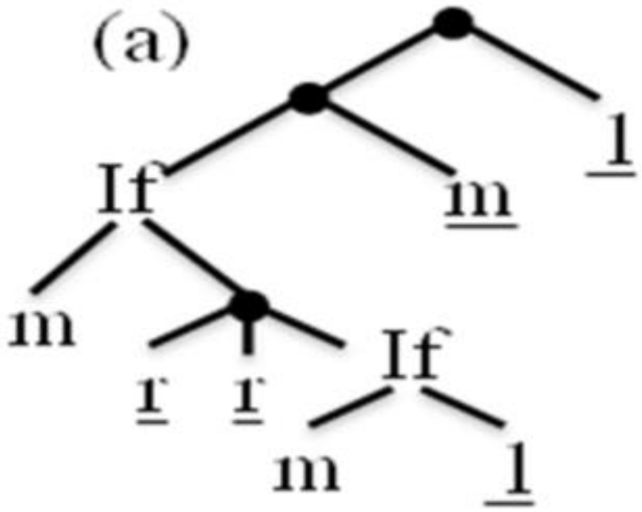


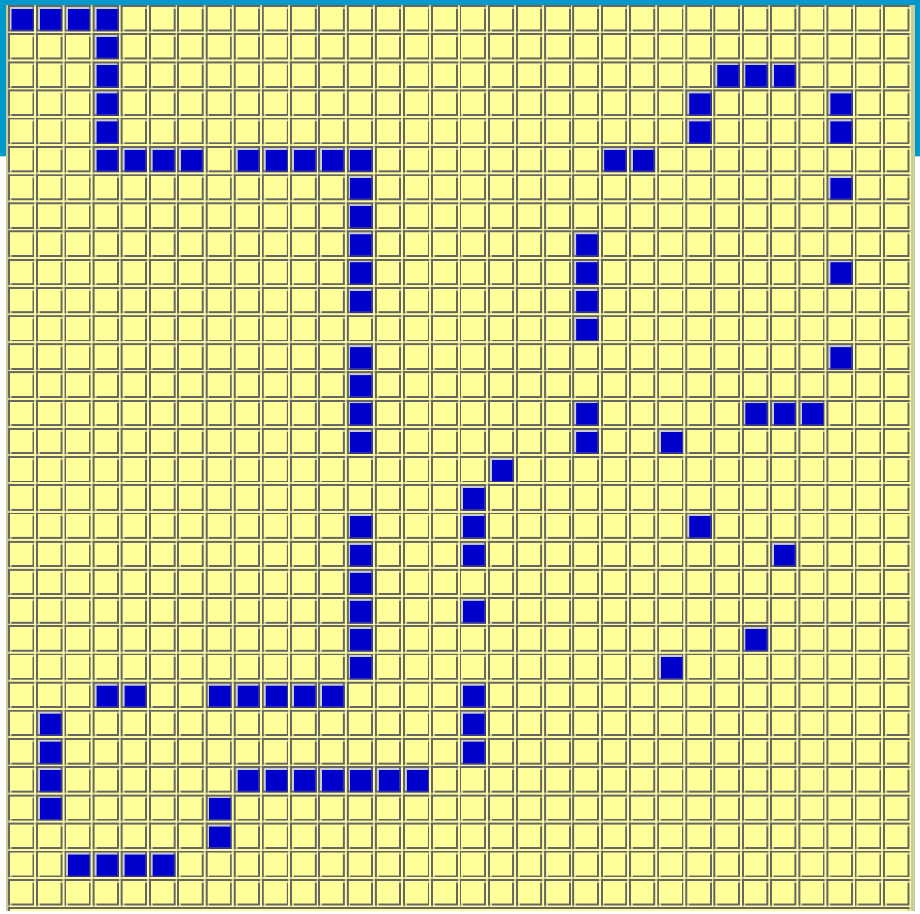
Move:

rrmml



Santa Fe Ant Trail

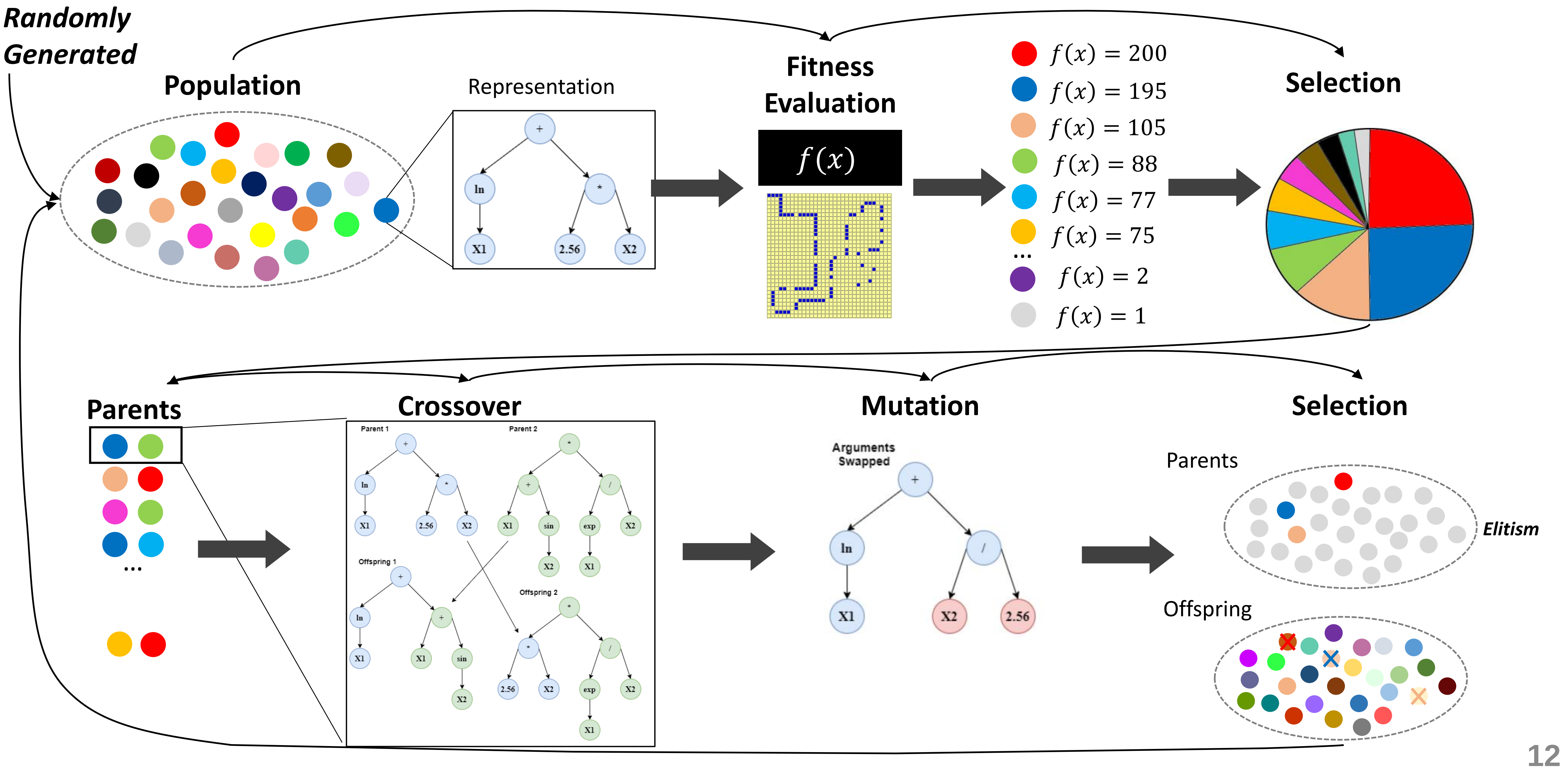




All the Steps

```
mml,rrlml,rrmml,rrlml,rrlml,rrlml,mml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrmml,rrlml,rrlml,rrlml
,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrmml,rrlml,rrlml,
rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,mml,rrlml,rrlml,rrlml,rrlml,mml,rrlml,rrlml,rrlml,rrlml,mml,r
rlml,rrmml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,mml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlm
l,rrlml,rrlml,rrmml,rrlml,rrlml,mml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrmml,rrlml,rrlml,m
ml,rrlml,rrmml,rrlml,rrlml,rrlml,rrmml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrlml,rrmm
l,rrlml,rrlml,rrlml,rrlml,mml,rrlml,rrlml,mml,rrlml,rrlml,rrmml,rrlml,rrlml,rrmml,rrlml,rrlml,mml
```

Evolve our Programme using GP



Crossover

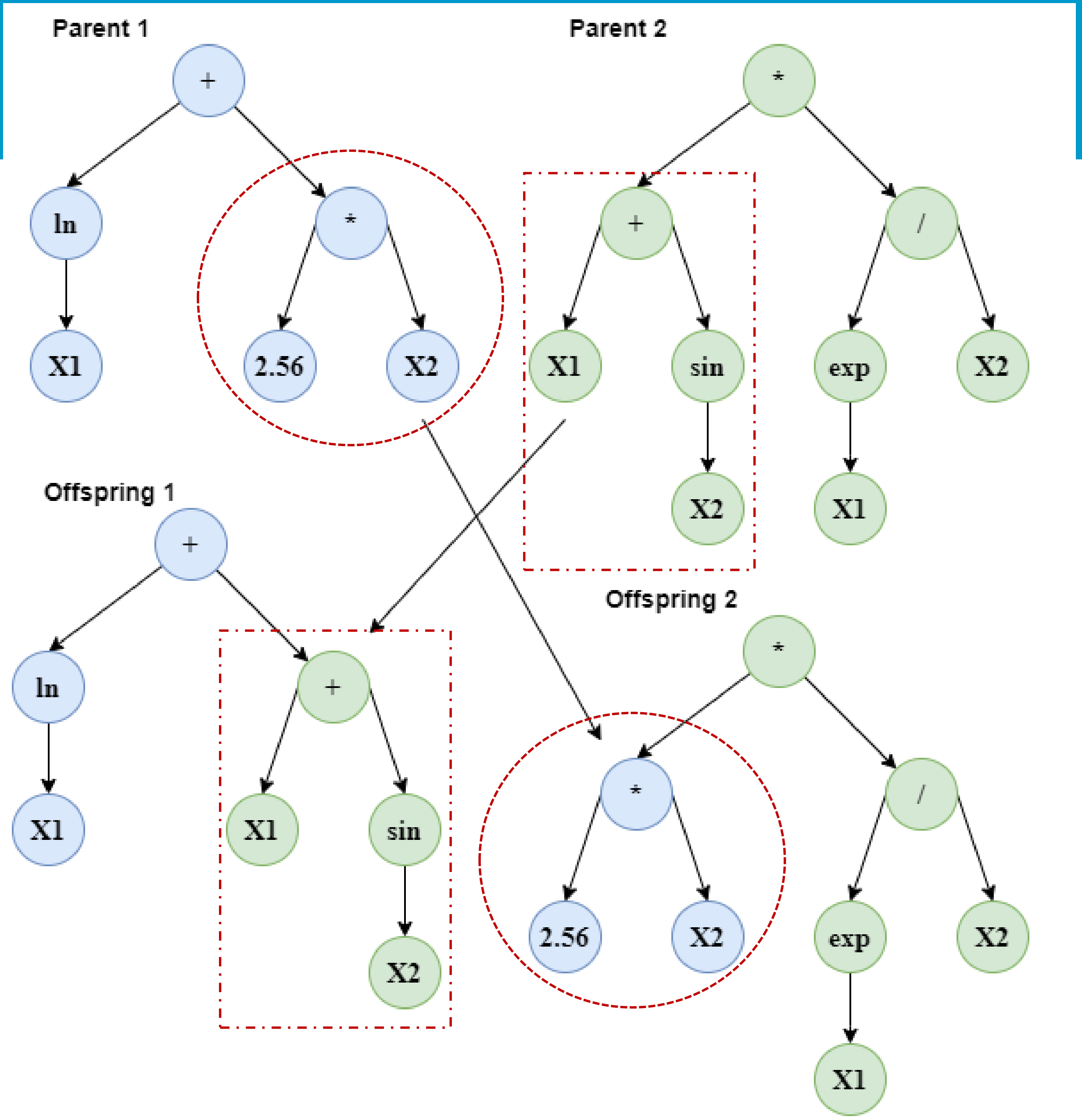
- Crossover operators in genetic programming are quite intuitive. As for other evolutionary algorithms, given two parents, we randomly select a point in each parent and crossover that subtree at that point to create an offspring.

Parent 1 $f(x_1, x_2) = \ln(x_1) + (2.56 \times \mathrm{x}_2)$

Parent 2 $f(x_1, x_2) = (x_1 + \sin(x_2)) \times \left(\frac{e^{x_1}}{x_2}\right)$

Child 1 $f(x_1, x_2) = (2.56 \times \mathrm{x}_2) \times \left(\frac{e^{x_1}}{x_2}\right)$

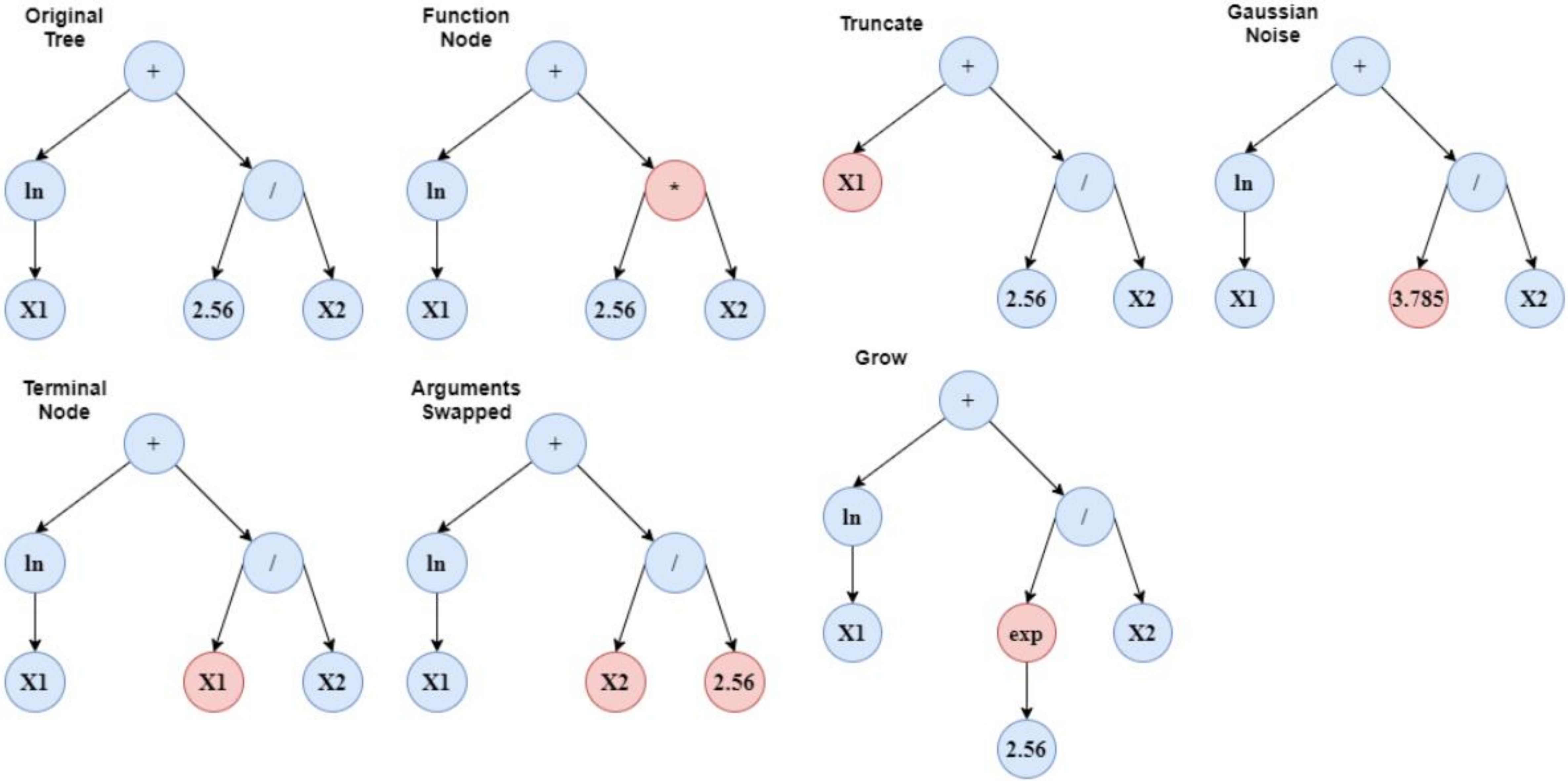
Child 2 $f(x_1, x_2) = \ln(x_1) + (x_1 + \sin(x_2))$



Mutation

- Like crossover, mutation for genetic programming is extremely intuitive, and similar to other evolutionary algorithms.
  - **Function node switching** - switch a random function node to another viable node.
  - **Terminal node switching** - switch a random terminal node with another viable node
  - **Swapping arguments of a terminal** – swap to terminal nodes
  - **Gaussian Noise** - add random gaussian noise to existing numeric values.
  - **Grow** - grow our trees by randomly introducing a single new function node.
  - **Truncation** - shrink a tree by randomly deleting a single function node

Mutation



Original Tree
+
ln
/
X1
2.56
X2
Function Node
+
ln
*
X1
2.56
X2
Truncate
+
X1
/
2.56
X2
Gaussian Noise
+
ln
/
X1
3.785
X2
Terminal Node
+
ln
/
X1
X1
X2
Arguments Swapped
+
ln
/
X1
X2
2.56
Grow
+
ln
/
X1
exp
X2
2.56

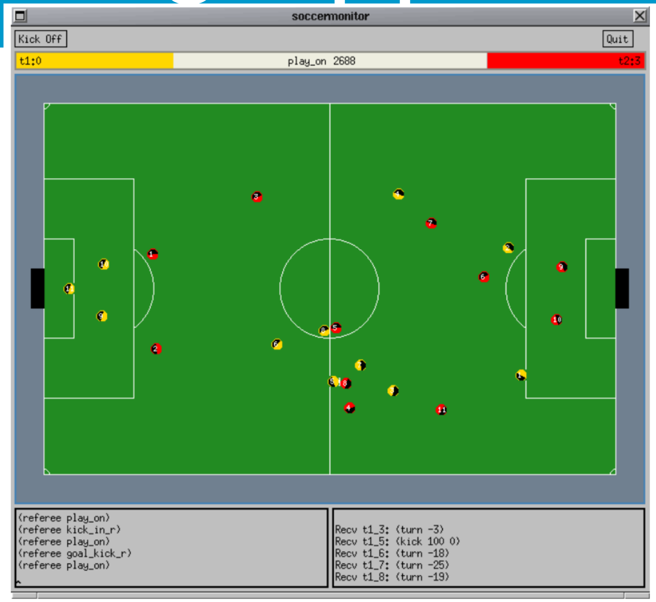
Santa Fe Ant Trail

```
 40:   4 ####
 41:   6 ######
 42:   3 ###
 43:   1 #
 44:   7 #######
 45:   2 ##
 46:   5 #####
 47:   2 ##
 48:   3 ###
 49:   6 ######
 50:   1 #
 51:   2 ##
 53:   4 ####
 54:   1 #
 55:   1 #
 56:   1 #
 57:   1 #
 58:   3 ###
 59:   1 #
 60:   1 #
 61:   1 #
 62:   1 #
 64:   2 ##
 66:   1 #
 67:   1 #
 68:   1 #
 71:   1 #
 78:   1 #
 85:   1 #
Average score: 4.576
```

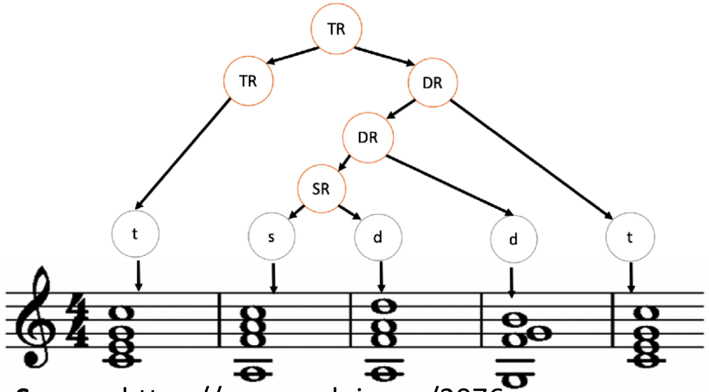
https://www.youtube.com/watch?v=InpbbgpDQkg

Genetic Programming Applications

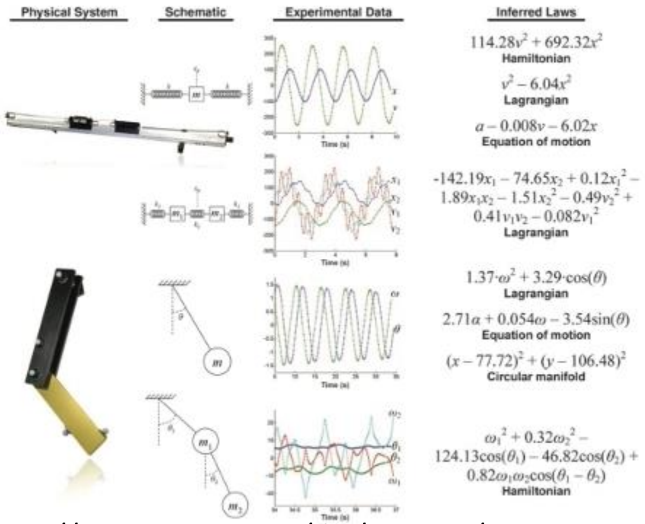
- Robotics ; Game Playing ; Control
- **Machine Learning (Regression and Classification Problems)**
- Maths and Physics
- Systems Security
- Economy and Finances
- Music Creation
- Video Editing
- Etc.



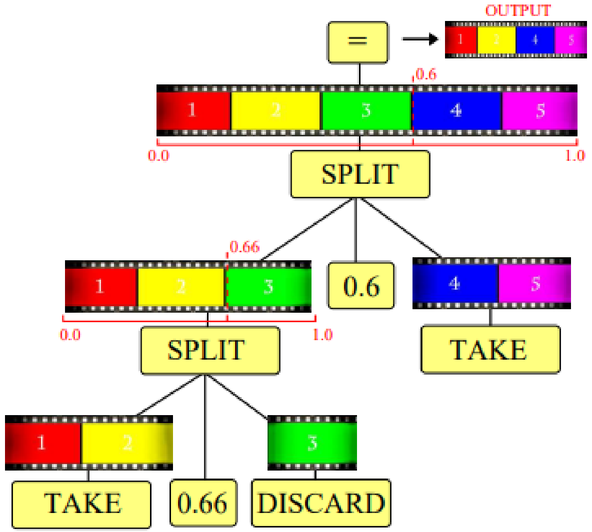
Source: http://www.genetic-programming.com/hc/andretellersoccer.html



Source: https://www.mdpi.com/2076-3417/10/17/6039/htm



Source: https://www.science.org/doi/10.1126/science.1165893



Source: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.9167&rep=rep1&type=pdf

Symbolic Regression

Nuno Antunes Ribeiro

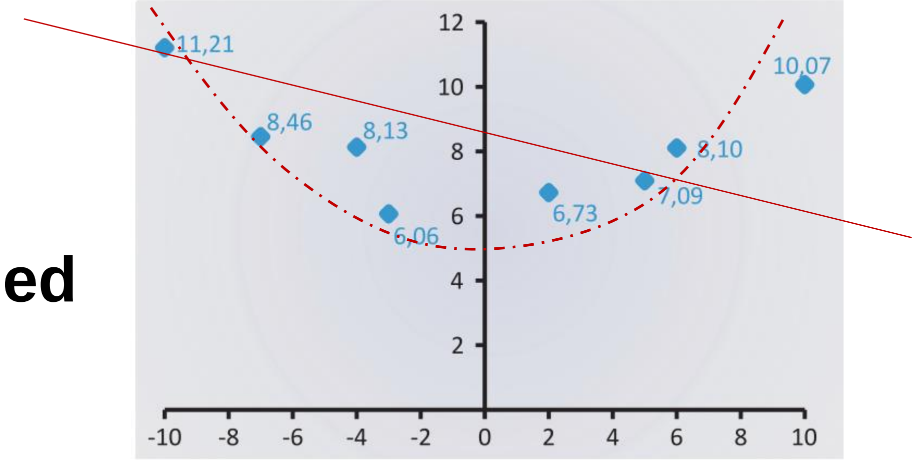
Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Engineering Systems and Design

Genetic Programming - Symbolic Regression

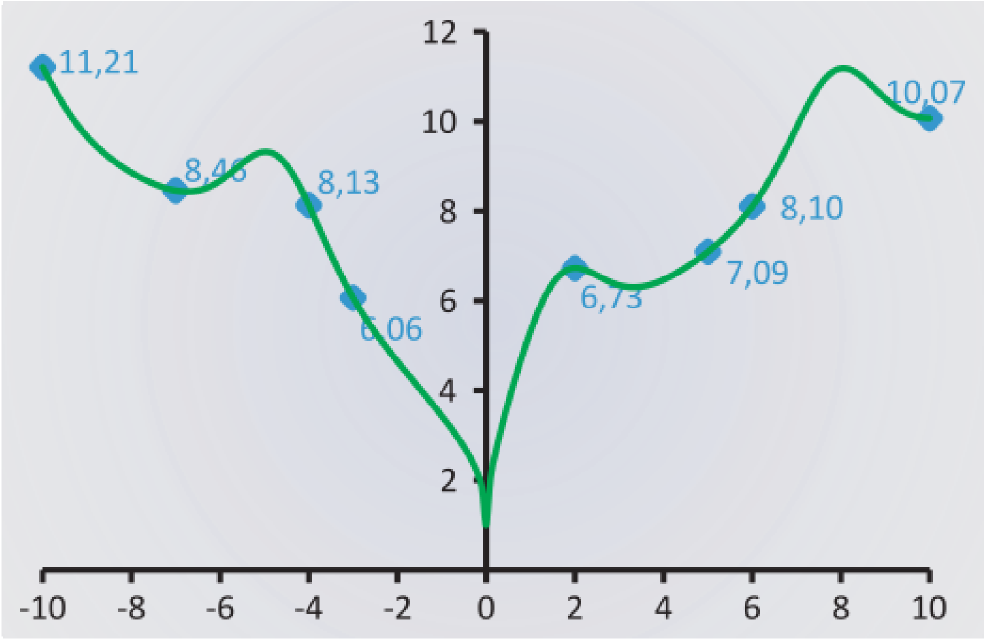
- Linear Regression
- Other regression methods
  - ➢ **Assumption about formula blueprint needed**

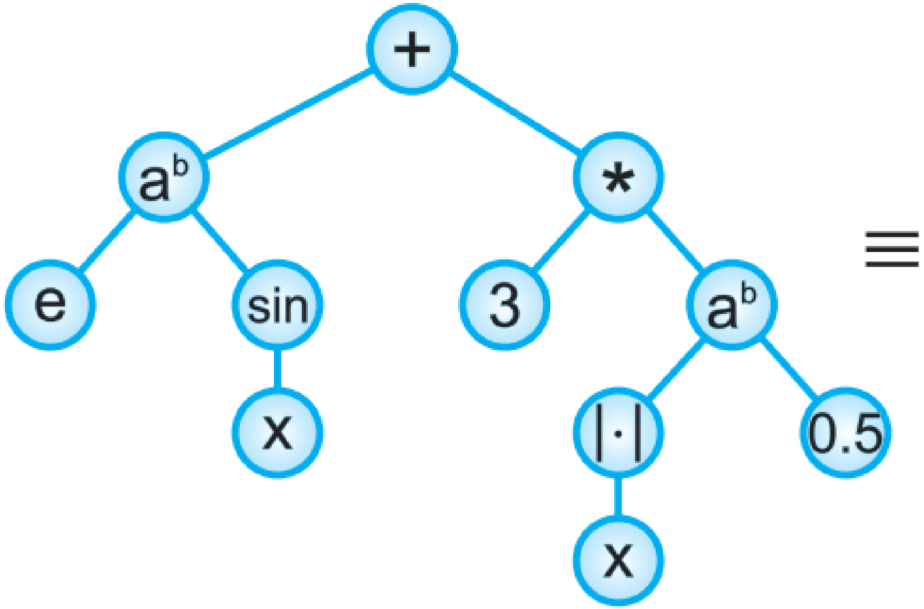


- **Symbolic regression** - new kind of optimization problem:
- We have
  - Given set of points (observations) - $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - A Function Set with elementary functions (e.g. $F = \{+, -, \times, \sqrt{\ }, sin, exp, \ldots\}$)
  - A Terminal Set, i.e. input variables (e.g. $T = \{x \; and \; real \; constants\}$)
- We want to find the best formulation (combining F and T) that best fits the observations S

Genetic Programming - Symbolic Regression

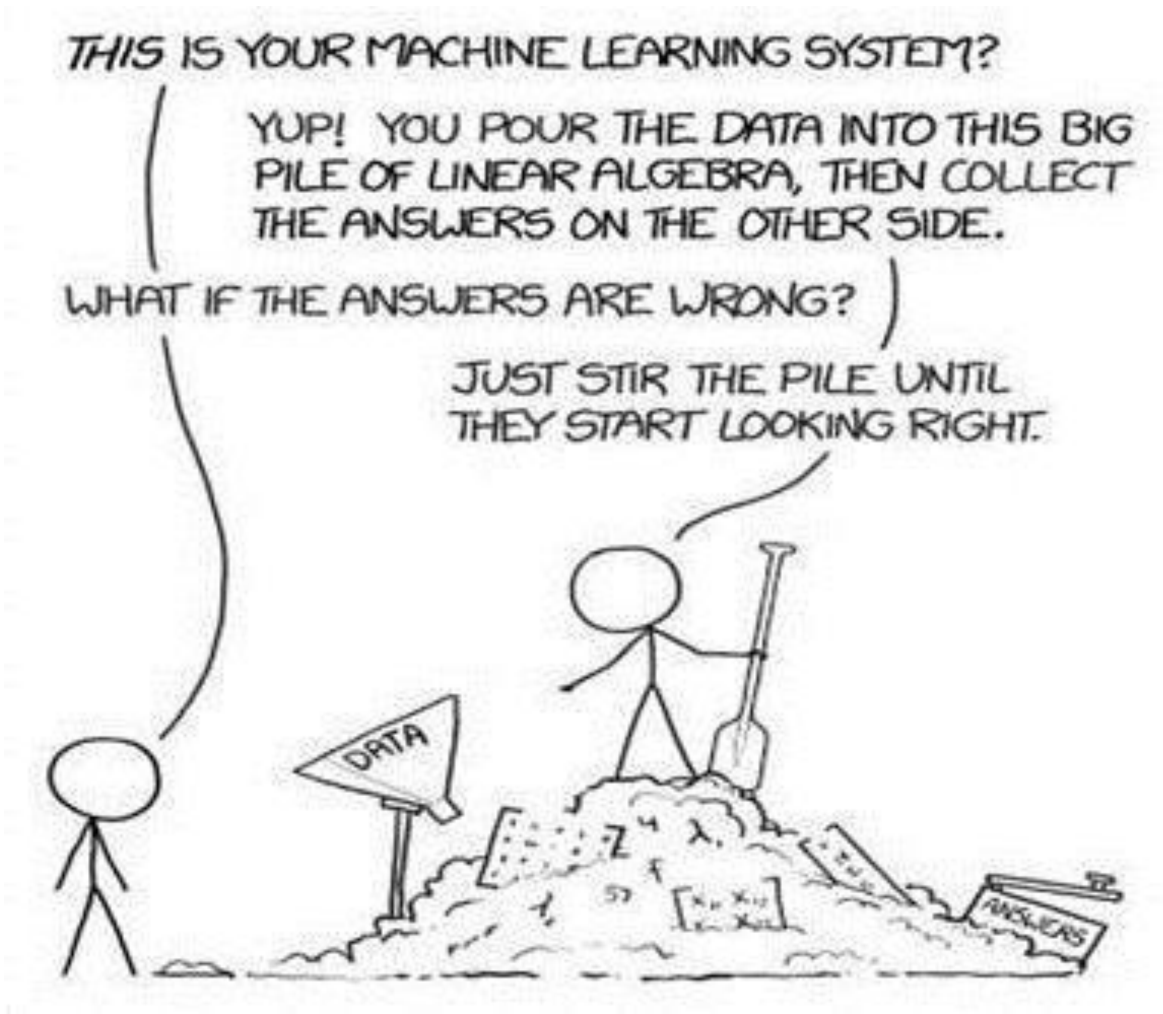
- Symbolic Regression with Genetic Programming:
  - represent formulas as tree data structures
  - Evaluation function: minimize $\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2$
  - Construct $f(x)$ with Genetic Programming!





$\equiv e^{\sin x} + 3\sqrt{|x|}$

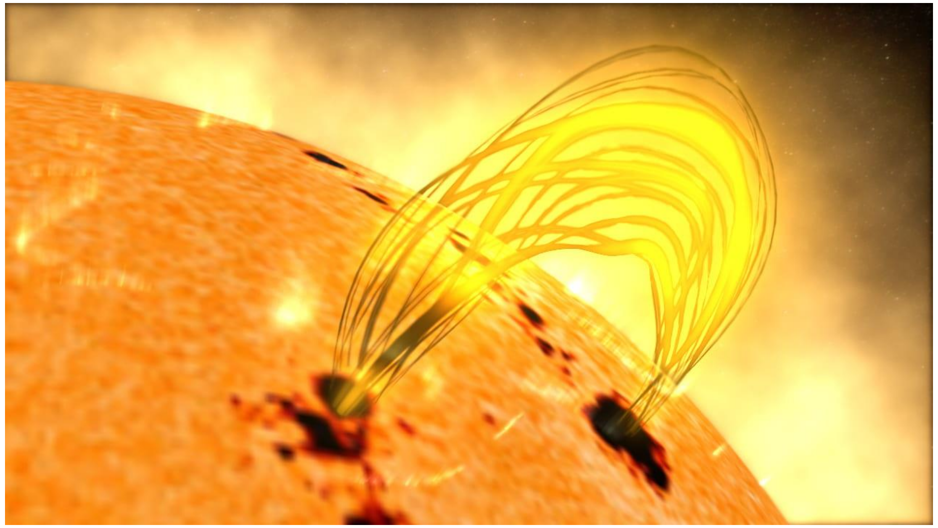
Genetic Programming Algorithm

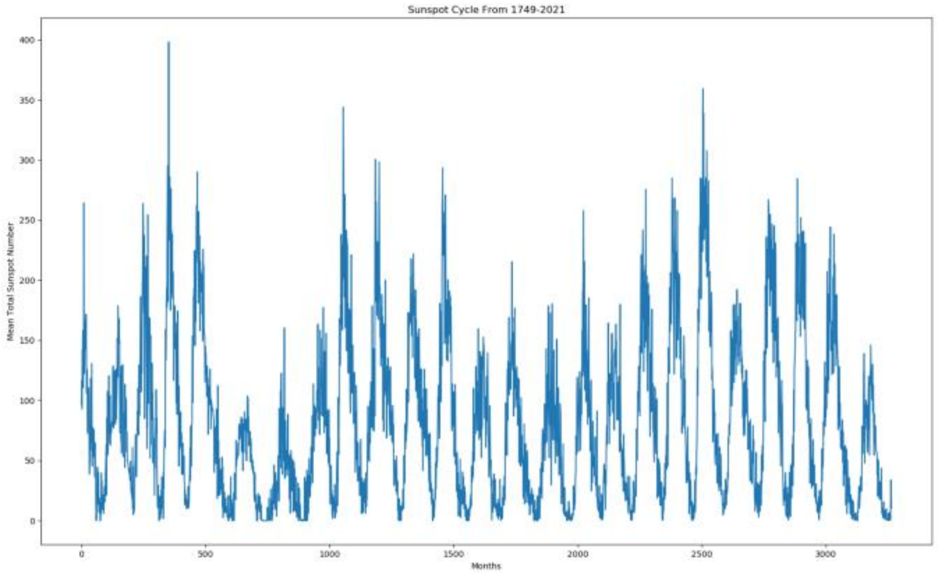


https://meiyi1986.github.io/cec2021-tutorial-xaigp/

Symbolic Regression in Python

- Sunspots have been observed for over four centuries, constituting the longest running, continuous time series of any natural phenomena in the Universe.
- Sunspots spawn severe space weather characterized by solar flares, coronal mass ejections, geomagnetic storms, enhanced radiative, and energetic particle flux
- Endangering satellites, global communication systems, air-traffic overpolar routes, and electric power grids
- Protection of planetary technologies and space situational awareness is therefore enabled by solar activity predictions.





Sunspots.csv

Symbolic Regression in Python

- Programming a genetic programming model from scratch requires a lot of extracurricular preliminaries, such as automata theory, I will not be performing the algorithm from scratch.
- Instead, I will be using **gplearn**, a free python library designed specifically for genetic programming algorithms for both classification and regression.

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

gplearn

https://github.com/trevorstephens/gplearn

Symbolic Regression in Python

```
import numpy as np
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt
from gplearn.genetic import SymbolicRegressor
from sympy import*

#conda install python-graphviz

seed=2

np.random.seed(seed)

df = pd.read_csv("Sunspots.csv")
y = np.asarray(df['Monthly Mean Total Sunspot Number'])
size = len(y)
# 50% of data for training
train_ind = int(size * 0.50)
# 25% of data for validation and other 25% for testing
val_ind = int(size * 0.75)

# testing Genetic Programming algorithm using gplearn

max_window = 8
min_window = 3

best_models = []  # best model from each run of the algorithm per window size
best_fits = []
# randomly shuffle data through indices:
shuffled_indices = np.asarray(range(0, size-max_window))
np.random.shuffle(shuffled_indices)
# loop over each window size
```

Import gplearn package

We split our data into the **training, validation, and testing** sets.
Our algorithms will be trained using the training dataset.
The validation will be used to compare our models.
Lastly, after we've chosen our final model, we will evaluate it's accuracy through the test dataset.
See Slide 36 – Lecture 13

Symbolic Regression in Python

```
# loop over each window size
for vision in range(min_window, max_window + 1):
    input = []
    output = []
    # creates the window length size for each value
    # because the first couple values will not have
    # a full window we skip them, that's why start
    # at i and not 0
    for j in range(vision, size):
        input.append(y[(j - vision):j].tolist())
        output.append(y[j])

    input = np.asarray(input)
    output = np.asarray(output)

    temp = np.column_stack((output, input))

    # instead of shuffle each time here, we shuffle once outside loop
    # so that all window sizes have the same final array
    temp = temp[shuffled_indices]

    output = temp[:, 0]
    input = temp[:, 1:]

    y_train = output[0:train_ind]
    y_val = output[train_ind:val_ind]
    y_test = output[val_ind:size]
    x_train = input[0:train_ind]
    x_val = input[train_ind:val_ind]
    x_test = input[val_ind:size]

    function_set = ['add', 'sub', 'mul', 'div']
    temp_val = []
    temp_models = []
    for i in range(0, 3):
        gp = SymbolicRegressor(population_size=500, metric='mse',
                               generations=20,init_depth=(2, 6),
                               verbose=1, function_set=function_set, parsimony_coefficient=0.4)

        gp.fit(x_train, y_train)
        predictions = gp.predict(x_val)
        predictions = np.where(predictions<0, 0, predictions)
        mse1 = mean_squared_error(y_val, predictions)
        print("  MSE Val: " + str(mse1))
        temp_val.append(mse1)
        temp_models.append(gp)
    best_index = np.argmin(temp_val)
    best_models.append(temp_models[best_index])
    best_fits.append(temp_val[best_index])
```

Loop throughout different window sizes (feature selection)

Prepare explanatory data

Split data into **training, validation, and testing** sets.

Apply Genetic Programming (see next slide)

Symbolic Regression in Python

```
function_set = ['add', 'sub', 'mul', 'div']
temp_val = []
temp_models = []
for i in range(0, 3):
    gp = SymbolicRegressor(population_size=500, metric='mse',
                           generations=20,init_depth=(2, 6),
                           verbose=1, function_set=function_set, parsimony_coefficient=0.4)

    gp.fit(x_train, y_train)
    predictions = gp.predict(x_val)
    predictions = np.where(predictions<0, 0, predictions)
    mse1 = mean_squared_error(y_val, predictions)
    print("  MSE Val: " + str(mse1))
    temp_val.append(mse1)
    temp_models.append(gp)
best_index = np.argmin(temp_val)
best_models.append(temp_models[best_index])
best_fits.append(temp_val[best_index])
```

Set of Functions

Basic gplearn parameters (symbolic regression)

Compute MSE

We generate 3 programmes (i.e. we apply GP three times) to avoid getting stuck in local optima – multistart approach

We then select the best programme from the three to test using the validation data

Symbolic Regression in Python

```
    |   Population Average    |             Best Individual              |
---- ------------------------- ------------------------------------------ ----------
 Gen   Length          Fitness   Length          Fitness      OOB Fitness  Time Left
   0    34.16      1.15449e+42        5          748.416              N/A      6.71s

   1    11.32      4.13481e+10        5          748.416              N/A      4.76s
   2     4.57      4.27409e+06        7          742.902              N/A      4.15s
   3     4.27          4997.45        7          742.902              N/A      3.80s
   4     5.90          5845.34        9          741.201              N/A      3.70s
   5     6.44      6.13391e+07       13          707.885              N/A      3.45s
   6     8.22      5.68907e+07       15          706.454              N/A      3.32s
   7    11.74      7.37752e+07       27          668.907              N/A      3.13s
   8    13.64      3.09287e+07       25          668.894              N/A      2.97s
   9    18.06      5.05252e+07       21          663.364              N/A      2.80s
  10    27.14      2.80803e+07       23          658.551              N/A      2.79s
  11    25.20      2.37904e+07       23          658.551              N/A      2.41s
  12    24.99      3.74852e+13       21          658.327              N/A      2.34s
  13    22.15      1.08825e+07       21          658.327              N/A      1.76s
  14    19.26      4.72406e+07       21          658.327              N/A      1.42s
  15    18.78      2.39752e+07       21          658.327              N/A      1.12s
  16    18.12      2.42965e+07       21          658.217              N/A      0.85s
  17    18.49         5.56e+07       19           658.01              N/A      0.57s
  18    16.93      3.17151e+07       19           658.01              N/A      0.28s
  19    16.97      3.03626e+07       17          649.785              N/A      0.00s
 MSE Val: 625.7448334070784
```

Gplearn output

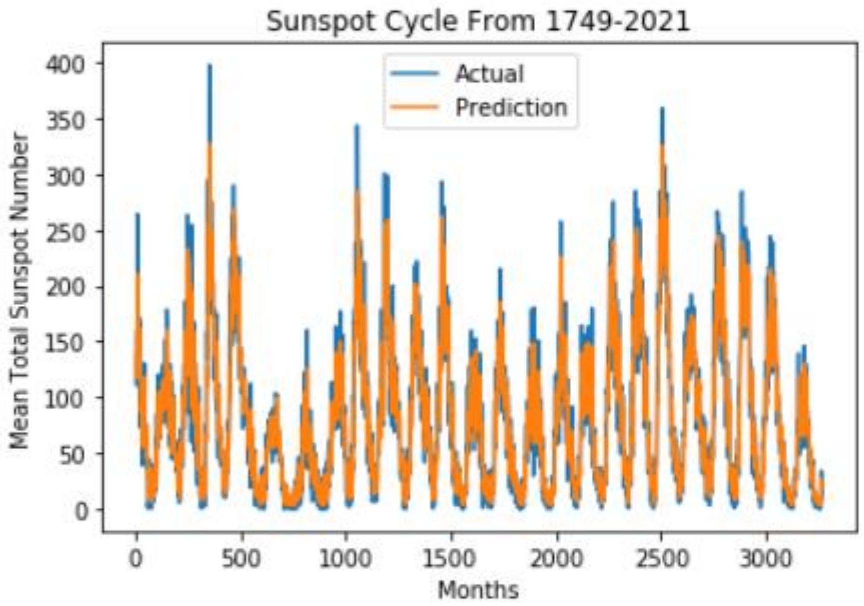
GP estimation of the time left to complete the calibration

MSE for the validation data

The length of a program is the number of elements in the formula which is equal to the total number of nodes

GP Application - Time Series Analysis

```
Best Validation Fitness Values Per Window Size:
Window Size: 3 - Validation MSE: 707.5851095958388
Window Size: 4 - Validation MSE: 625.7448334070784
Window Size: 5 - Validation MSE: 603.3792897649598
Window Size: 6 - Validation MSE: 695.9561339158797
Window Size: 7 - Validation MSE: 705.650323050849
Window Size: 8 - Validation MSE: 730.6631145004401
Validation Error: Mean w/ std: 678.1631340391745+-46.60890223275122
Best Model:
 Window Size : 5
 MSE for Test Data Set : 667.0923572895032
```




```
print(best_model._program)
```

```
add(div(sub(X2, X4), sub(sub(0.492, div(0.604, -0.420)), div(0.604, -0.420))), X4)
```

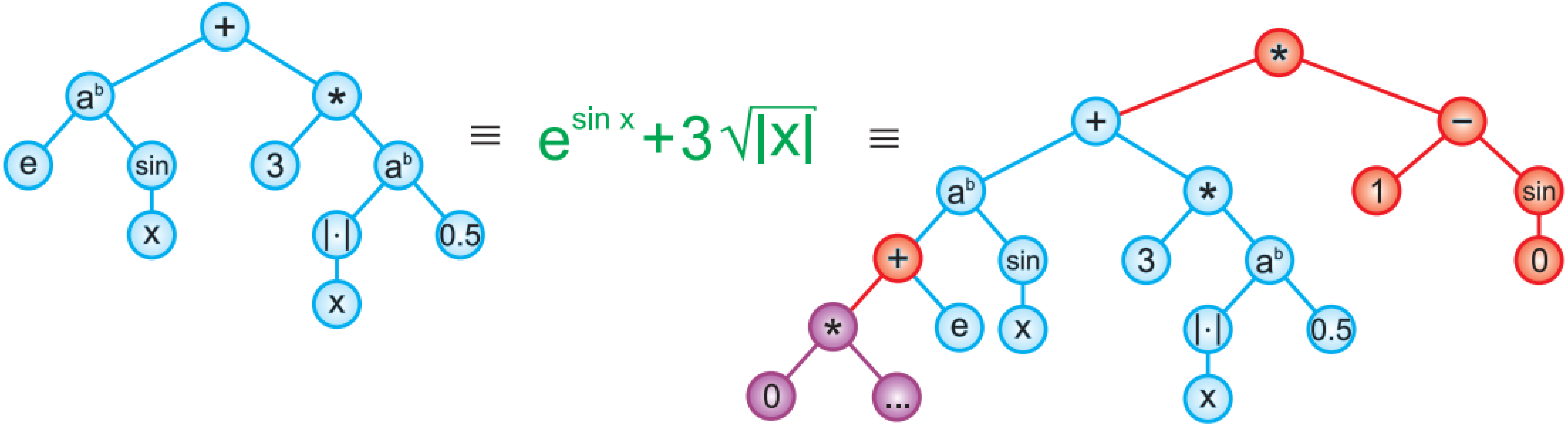
Results

Genetic Algorithms + NN	Backpropagation + NN	Genetic Programming
Testing	**Testing**	**Testing**
613	615	608
674	661	667
659	601	642
700	734	645
626	636	615
705	657	751
670	693	694
677	689	704
607	662	787
576	574	559

Symbolic regression is a great tool to be aware of.
It is not perfect for every kind of approach, but it gives you another ML option which can be really useful as the outcome is readily understandable.

Bloat in Genetic Programming

- Bloat: uncontrolled growth of programs
- Intron: useless part of program, one type of bloat



Bloat in Genetic Programming

- Bloat: uncontrolled growth of programs
- Why is it bad?
  - Elegant solutions are always simple and small
  - Larger programs = longer processing time for both, reproduction operations and evaluation
  - Larger programs = danger of overfitting
  - Larger programs occupy more memory
- What can we do against it?
  - Use multi-objective optimization: minimize also program size
  - Use **penalties in single-objective optimization**
  - Set a conservative upper bound for program size
  - Use specialized mutation and crossover operators which minimize bloat

Bloat in Genetic Programming

- Bloat can be fought in gplearn in several ways. The principal weapon is using a penalized fitness measure during selection where the fitness of an individual is made worse the larger it is.
- In this way, should there be two programs with identical fitness competing in a tournament, the smaller program will be selected and the larger one discarded.
- The parsimony_coefficient parameter controls this penalty and may need to be experimented with to get good performance.

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

```
gp = SymbolicRegressor(population_size=500, metric='mse',
                       generations=20,init_depth=(2, 6),
                       verbose=1, function_set=function_set, parsimony_coefficient=0.4)
```

Solving Classification Problems using Genetic Programming

Nuno Antunes Ribeiro

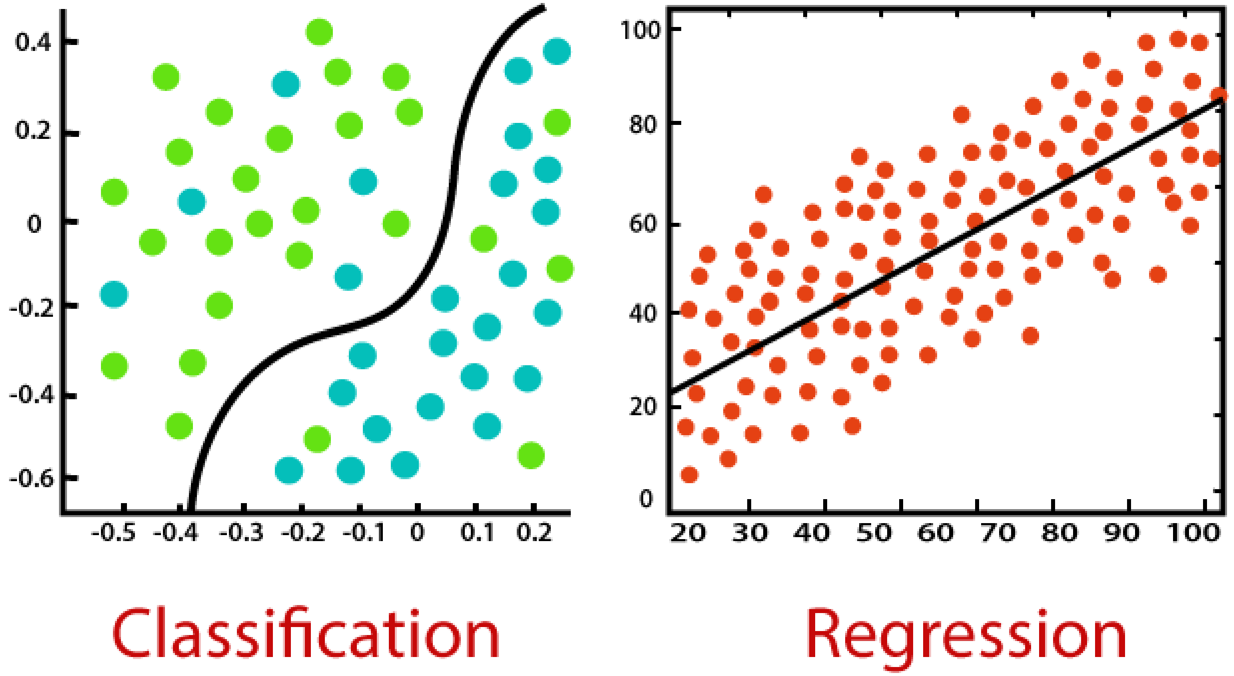
Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Engineering Systems and Design

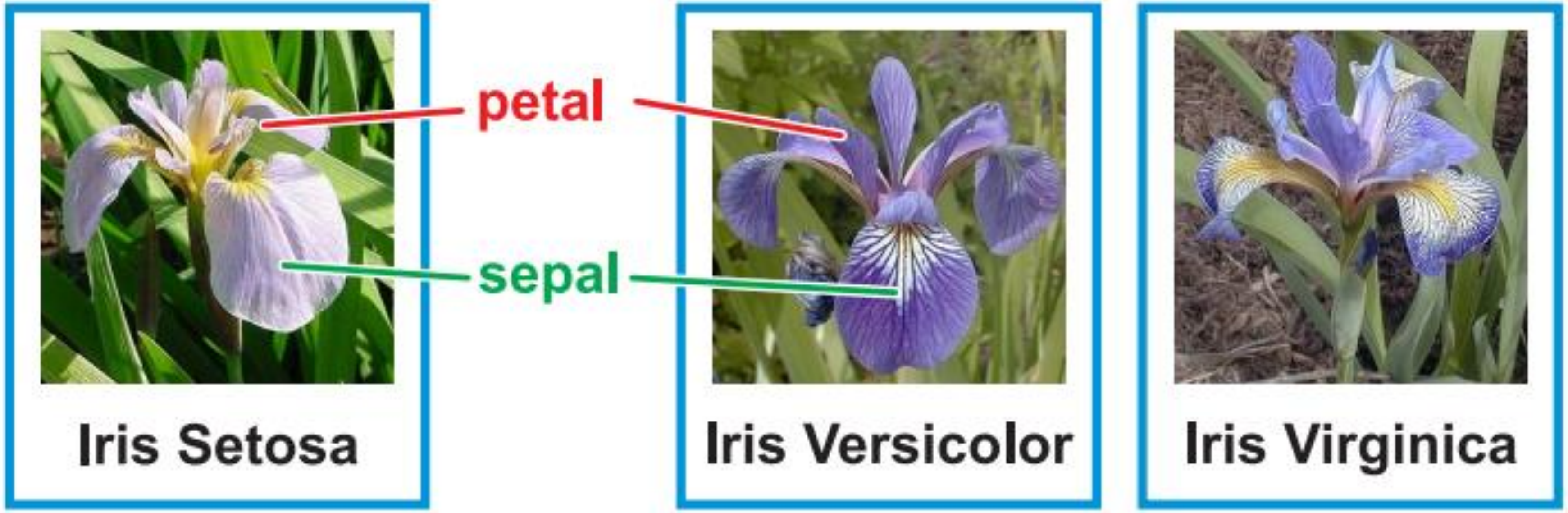
Classification Problems

- Classification is a task that requires the use of machine learning algorithms that learn how to assign a class label to examples from the problem domain. An easy to understand example is classifying emails as “spam” or “not spam.”
- There are many different types of classification tasks that you may encounter in machine learning and specialized approaches to modeling that may be used for each.



Example: Classify Irises

- Most classical example of a classification problem
- The petals and sepals of different iris flowers have been measured
- Can we use this data to find a program which tells us to what type a flower belongs on basis of petal and sepal measurements?



Example: Classify Irises

- Data samples $t = (t_1, t_2, \ldots, t_n)$; $t_i \in \mathbb{R}$ belongs to classes $k\ in\ K$
- Supervised learning: we use samples $t \in A$ with known classes $class(t) \in K$ to learn a function $f(t_i)$

Sepal Length	Sepal Width	Petal Length	Petal Width	Class
5.1	3.5	1.4	0.2	iris setosa
4.9	3.0	1.4	0.2	iris setosa
7.0	3.2	4.7	1.4	iris versicolor
6.3	3.3	6.0	2.5	iris virginica
...	...	...	...	...
6.4	3.2	4.5	1.4	iris versicolor

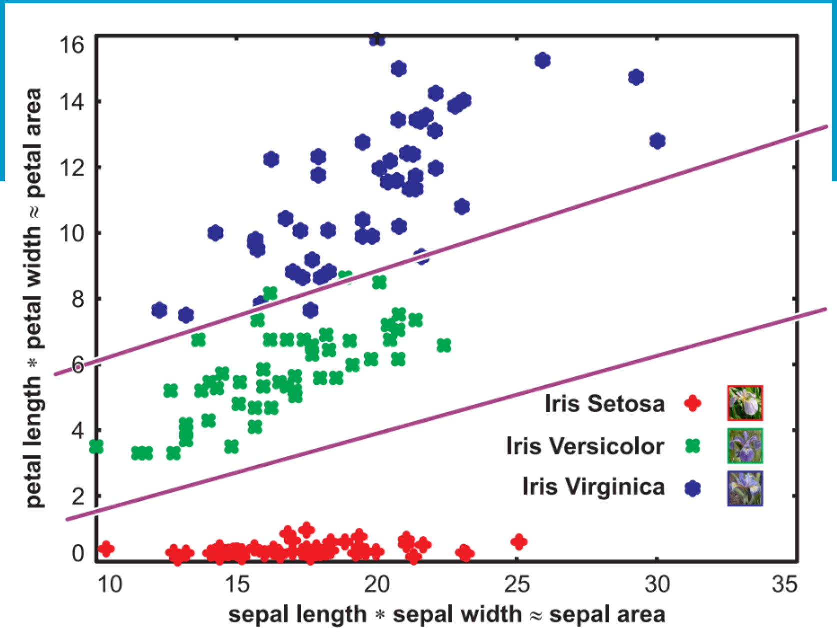
A

$t \in A$

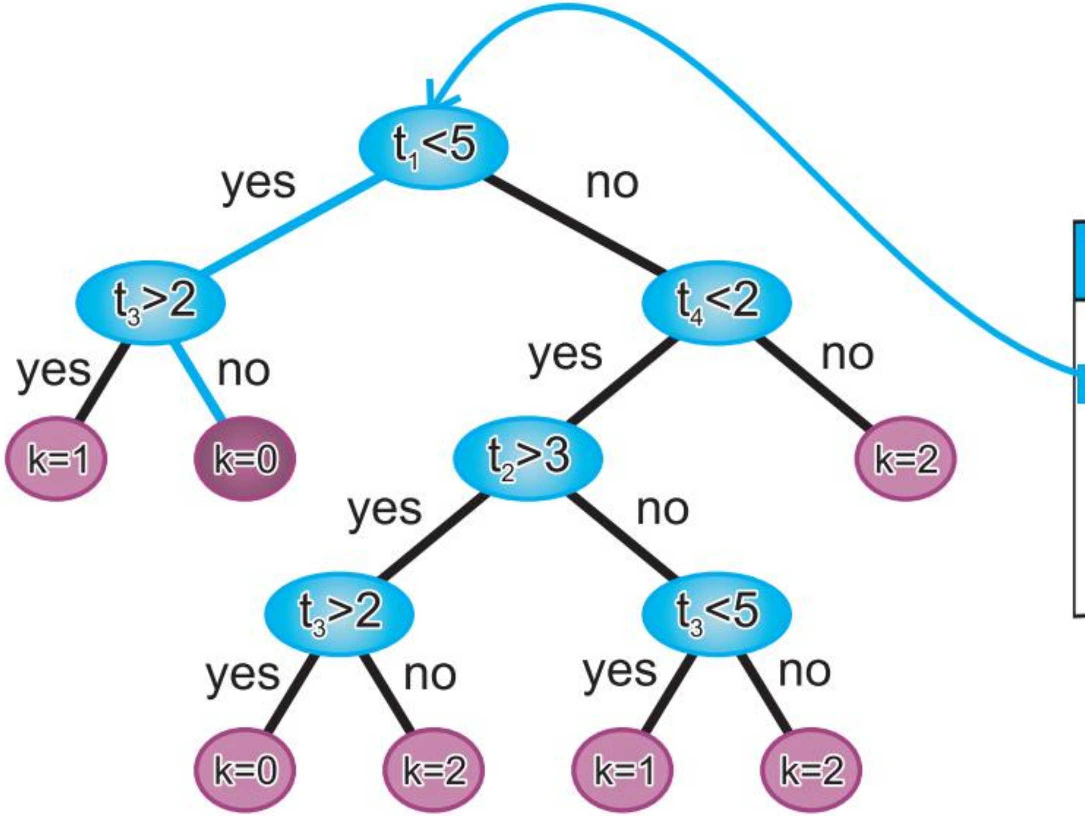
$k = class(t)$

$t_1 \in \mathbb{R}$ $t_2 \in \mathbb{R}$ $t_3 \in \mathbb{R}$ $t_4 \in \mathbb{R}$ K = {*setosa*, *versicolor*, *virginica*}

Example: Classify Irises



- Common Approach: Decision Trees

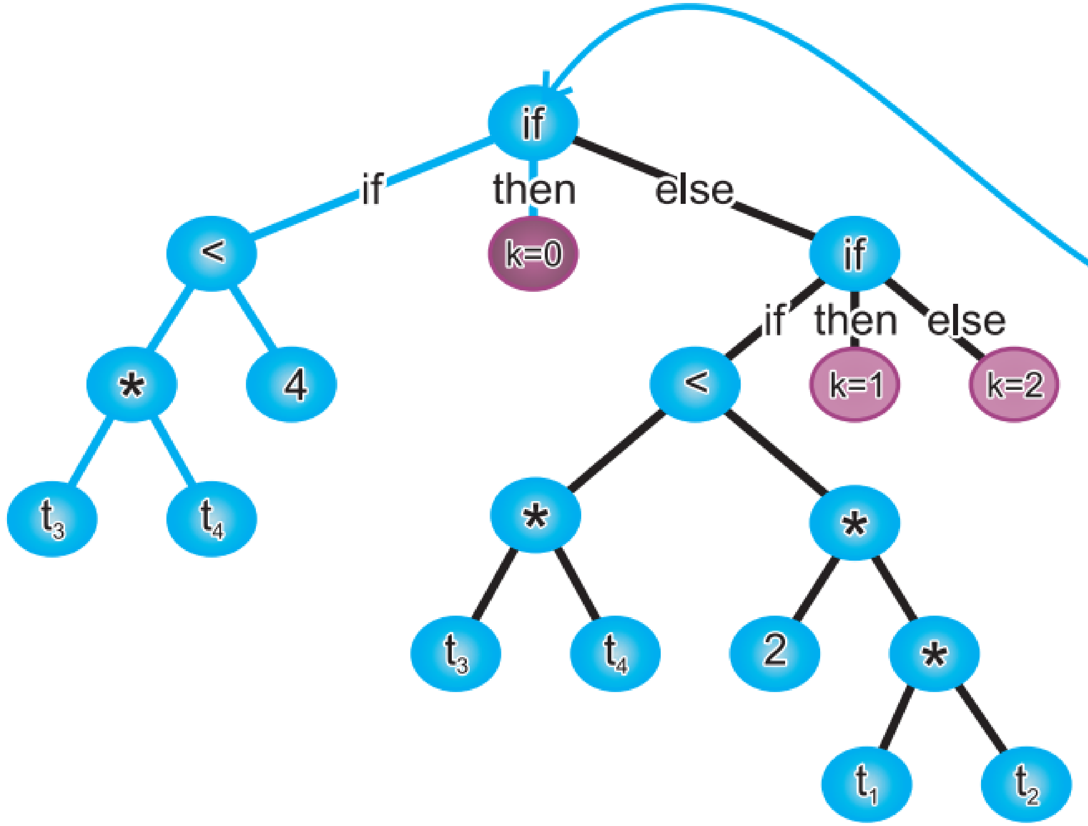


t_1	t_2	t_3	t_4	k
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
7.0	3.2	4.7	1.4	1
6.3	3.3	6.0	2.5	2
...	...	...	...	...
6.4	3.2	4.5	1.4	1

How decision trees work :
https://www.youtube.com/watch?v=ZVR2Way4nwQ

Example: Classify Irises

- In Genetic Programming decisions and tree shapes are not limited to a certain shape



t_1	t_2	t_3	t_4	k
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
7.0	3.2	4.7	1.4	1
6.3	3.3	6.0	2.5	2
...	...	...	...	...
6.4	3.2	4.5	1.4	1

`if` $t_3 * t_4 < 4 \Rightarrow$ class 0
`else if` $t_3 * t_4 < 2 * t_1 * t_2 \Rightarrow$ class 1
`else` class 2