Metaheuristic Optimization: Genetic Programming (GP)

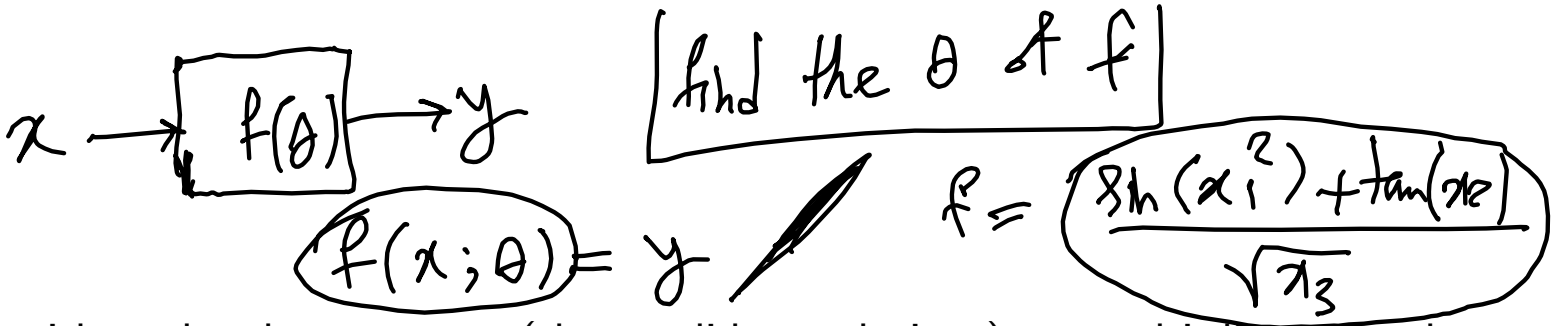
Adaptive and Cooperative Algorithms (ECE 457A)

ECE, MME, and MSCI Departments,
University of Waterloo, ON, Canada

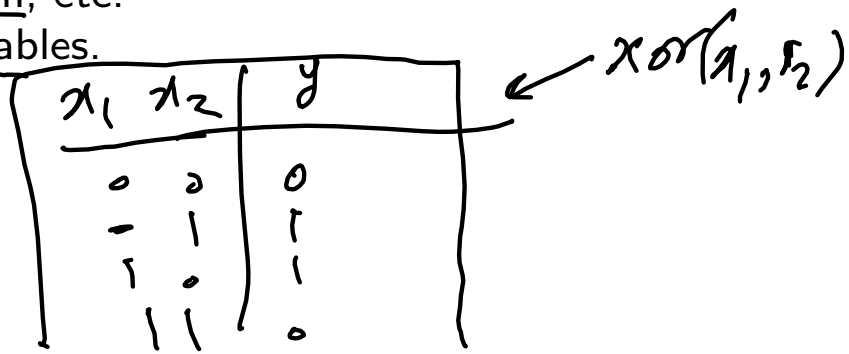
Course Instructor: Benyamin Ghojogh
Fall 2023

Idea and Concepts

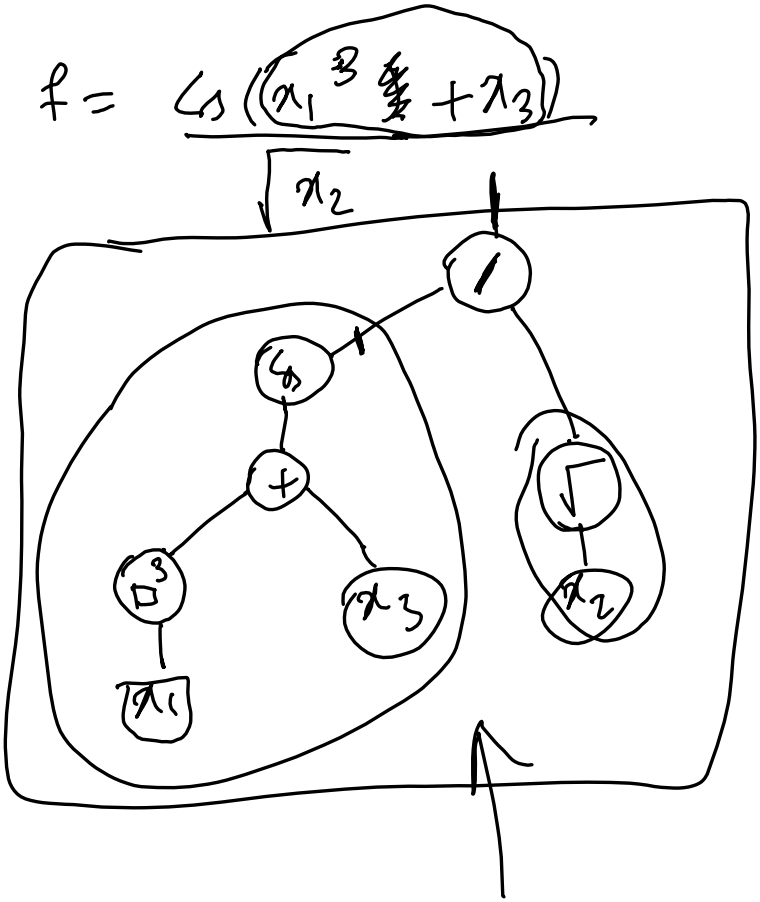
The Idea



- In genetic algorithm, the chromosomes (the candidate solutions) are multi-dimensional vectors or matrices.
- In genetic programming, the chromosomes (the candidate solutions) are tree data structures.
- Genetic programming was first proposed in 1985 [1].
- It tries to extend computational intelligence to programming and function generation.
- It can be used for logical expressions:
  - Tree nodes, except the terminal nodes, are logical operators such as AND, OR, NOT, etc.
  - Terminal nodes are are the binary variables.
- It can be used for function expressions:
  - Tree nodes, except the terminal nodes, are functions such as multiplication, division, addition, subtraction, sine, cosine, logarithm, etc.
  - Terminal nodes are are the continuous variables.



$f = \sin(x_1^3 + x_3)$

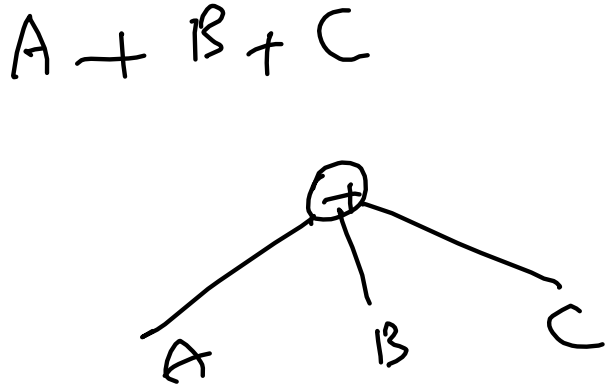


$\{x_1, x_2, x_3, x_4\}$

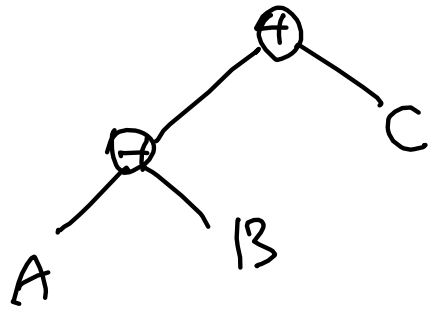
$\{\sin, \square^3, +, \sqrt{}, /\}$

Concepts in Genetic Programming

- A **tree** has some nodes starting with a root node.
- A node may or may not have a **child** or multiple **children**. When drawing the tree, every node is connected to its children by lines.
- A node without any children is a **terminal node**.
- **Branching factor**: The number of children at every tree node. The larger the branching factors, the more the width of the tree and usually the less the depth of the tree.
- **Terminal set**: the set of variables which are represented as the terminal nodes.
- **Function set**: the set of functions or operators which are represented as the nodes excluding the terminal nodes.
- **Semantic rules**: the rules for operations or functions which restrict the structure of tree from being any structure.



Example Trees



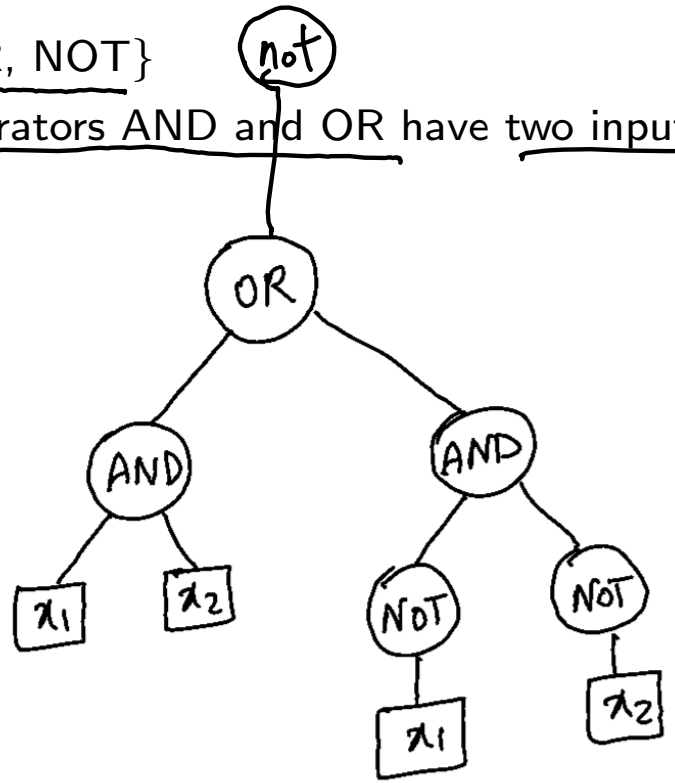
Logic Example

- Desired: X_1 XOR x_2
- According to the truth table in logic, XOR can be stated as:

$$X_1 \text{ XOR } x_2 = (X_1 \text{ AND } x_2) \text{ OR } ((\text{NOT } X_1) \text{ AND } (\text{NOT } X_2))$$

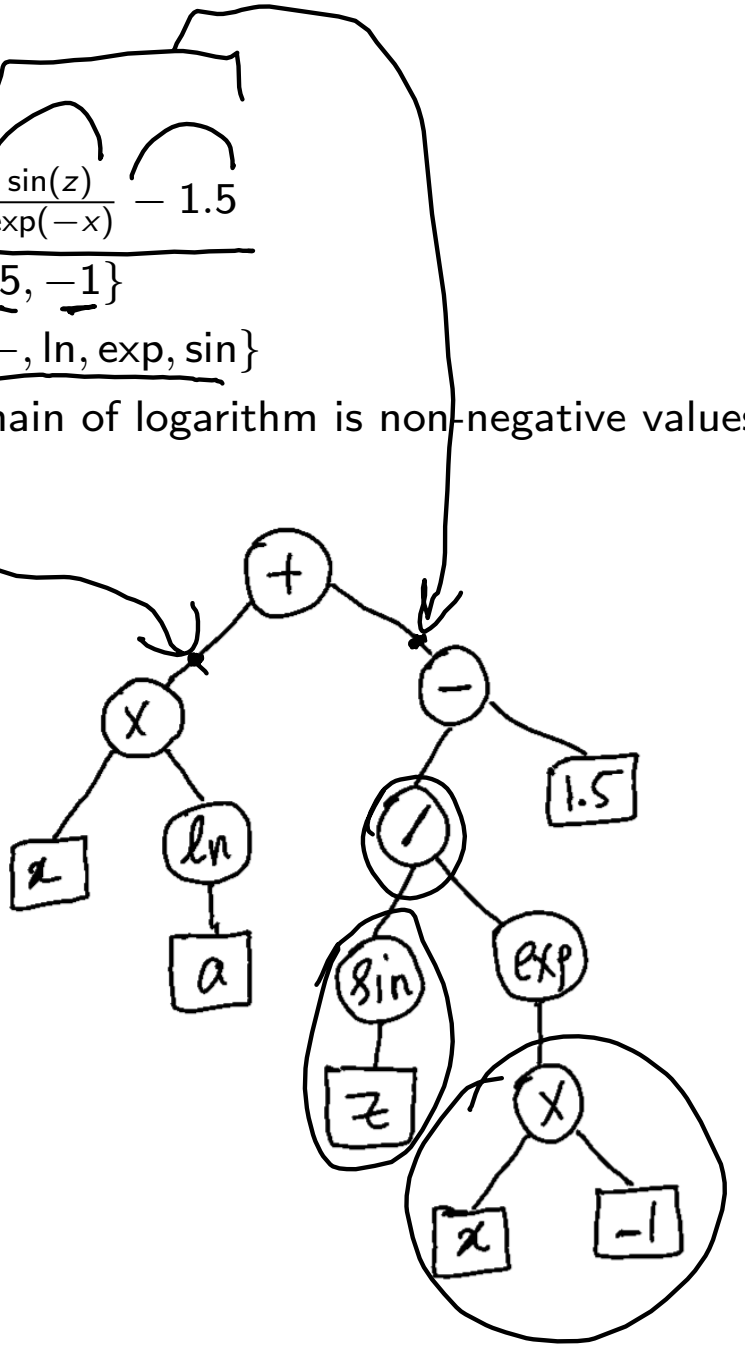
- Terminal set: $\{x_1, x_2\}$
- Function set: {AND, OR, NOT}
- Semantic rules: The operators AND and OR have two inputs each and the operator NOT has one input.

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Function Example

- Desired: $y = x \ln(a) + \frac{\sin(z)}{\exp(-x)} - 1.5$
- Terminal set: $\{x, a, z, 1.5, -1\}$
- Function set: $\{\times, /, +, -, \ln, \exp, \sin\}$
- Semantic rules: the domain of logarithm is non-negative values and we should not have division by zero.



Genetic Programming Algorithm

Genetic Programming Algorithm: Goal

- Goal of genetic programming:
  - We have a dataset (also called truth table) where every row has the input values and the corresponding ground-truth label or output value.
  - The goal is to find an expression of functions/operators which fits to this table and can predict the labels as accurate as possible.
  - It is similar to regression or classification problem in machine learning where we fit a learning model to predict as desired.
  - However, the big difference from machine learning is that genetic programming finds an expression (combination of functions or operators) to fit the dataset but machine learning learns some parameters of a model to fit the dataset!

function is fixed
parameters of function are tunable

Genetic Programming Algorithm: Initial Trees

- For generating every initial chromosome (candidate solution), we select a random root node from the function set.
- For generating the initial trees (chromosomes):
  - For every node, we first decide whether it is a function node or terminal node.
    - ★ If the depth at that node has reached the maximum depth, it must be a terminal node.
    - ★ If the depth at that node has not reached the maximum depth yet, we have a probability $p \in [0, 1]$ for whether the node is a function node or a terminal node. The probability for being a terminal node (i.e., a variable node) is $1 - p$.
  - For every node, we can let the branching factor b at every node be a random variable with Poisson distribution:

$$b \sim \mathbb{P}(b; \lambda) = \frac{\lambda^b e^{-\lambda}}{b!}, \tag{1}$$

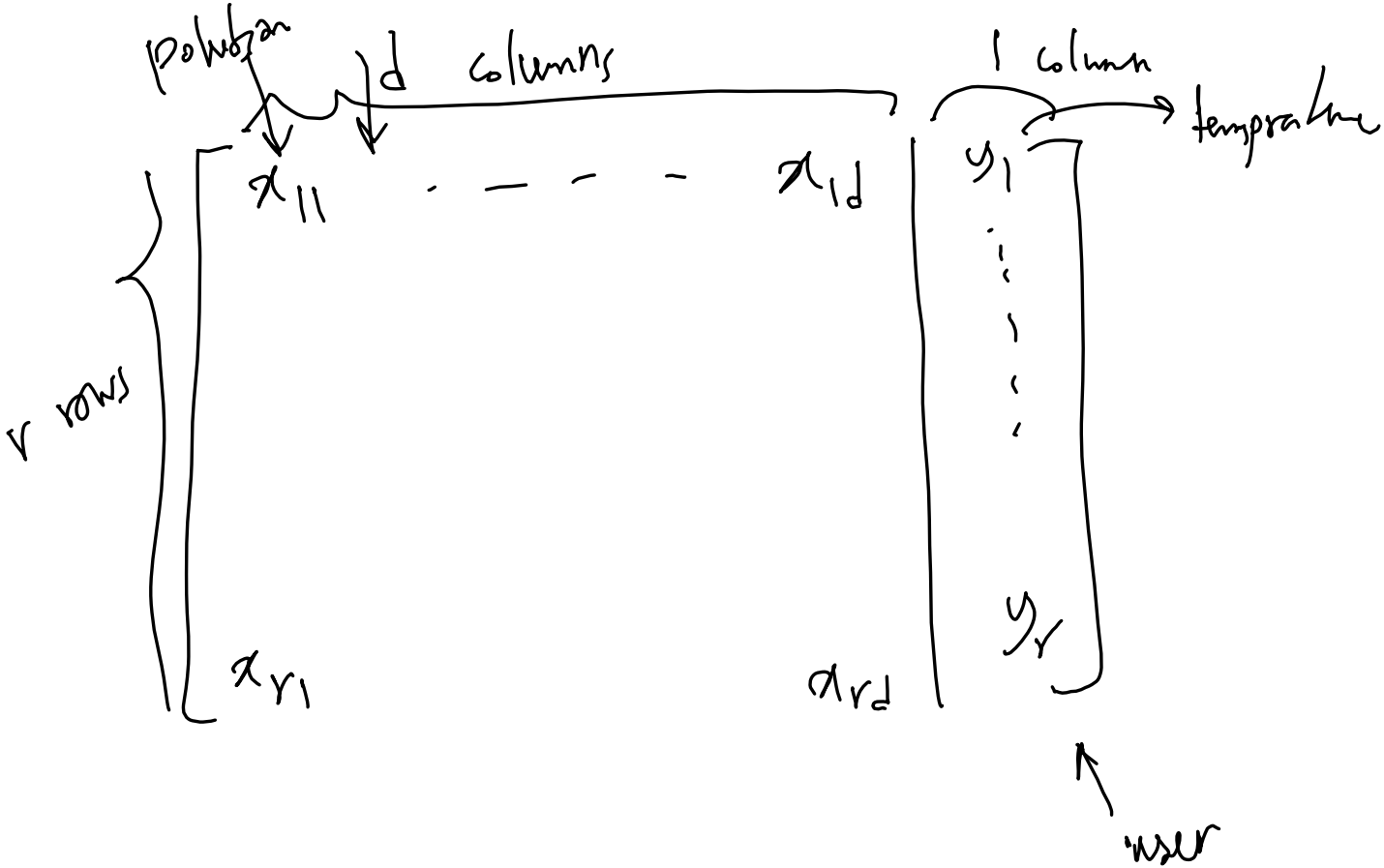
where the random variable b is a non-negative integer and $\lambda > 0$ is the hyperparameter of the distribution (e.g., $\lambda = 3$). The larger the λ, the larger the mean of distribution so the more probable it becomes to draw larger integers.

Genetic Programming Algorithm: Cost/Fitness

- Every tree structure, i.e., every combination of functions/operators, has some fitness or cost value.
- For a tree structure, the cost value can be calculated as the following:
  - Assume the dataset has r instances $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$ and the ground-truth labels are $\{y_1, \ldots, y_r\}$. Every data instance $\boldsymbol{x}_i$ has d features $\{x_{i1}, \ldots, x_{id}\}$.
  - Every tree structure represents an function expression where the variables of function are all or a subset of the d features.
  - We feed the every instance of dataset, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$, to the function expression and it outputs the predicted labels $\{\widehat{y}_1, \ldots, \widehat{y}_r\}$ corresponding to the instances.
  - The cost for the tree structure can be the mean squared error between the ground-truth and predicted labels:

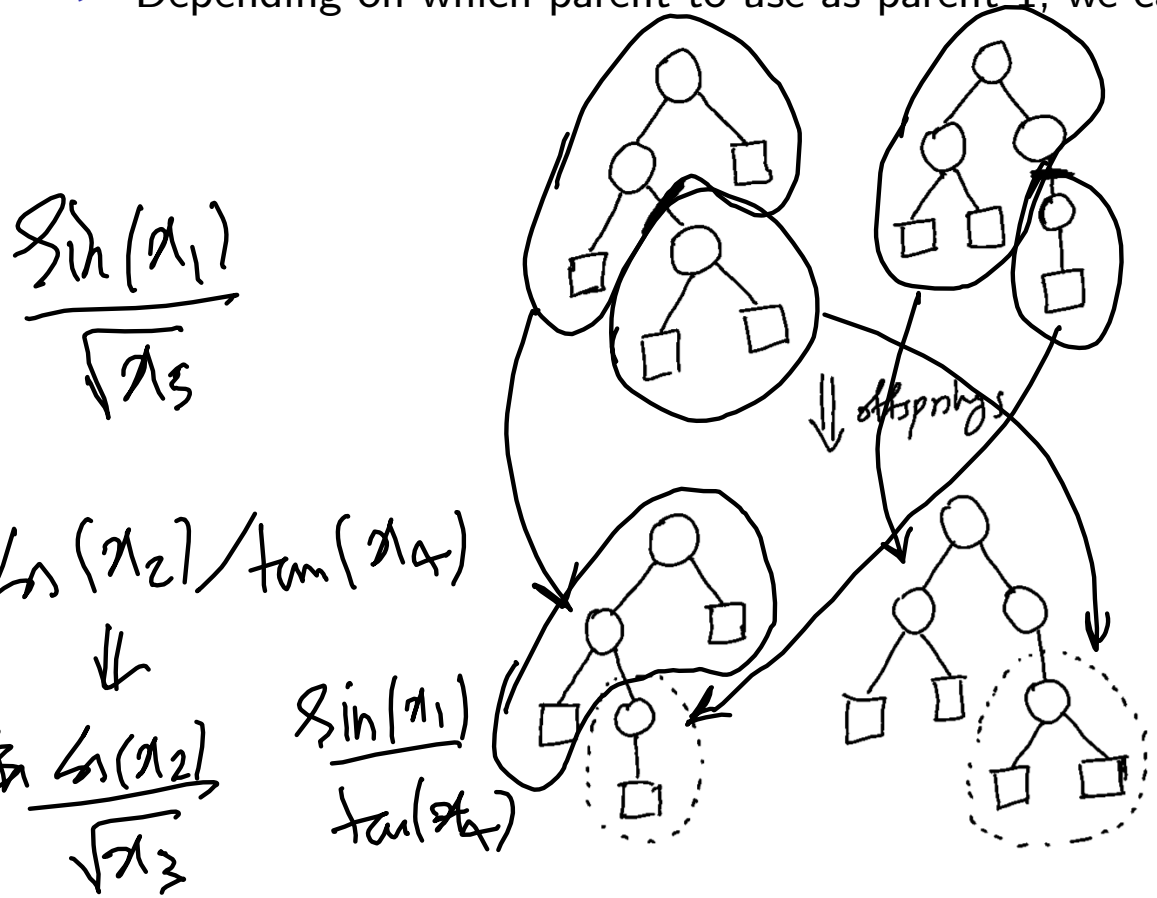
$$f(\text{tree}) = \frac{1}{r}\sum_{i=1}^{r}(\widehat{y}_i - y_i)^2. \tag{2}$$

  - If the number of instances in the dataset is too large, i.e., $r \gg 1$, we may use a subset of rows for calculating the cost. The subset can be the randomly sampled rows from the table for evaluating the cost of the tree.


pollutant
d columns
1 column
temperature
x_{11}
x_{1d}
y_1
r rows
y_r
x_{r1}
x_{rd}
user

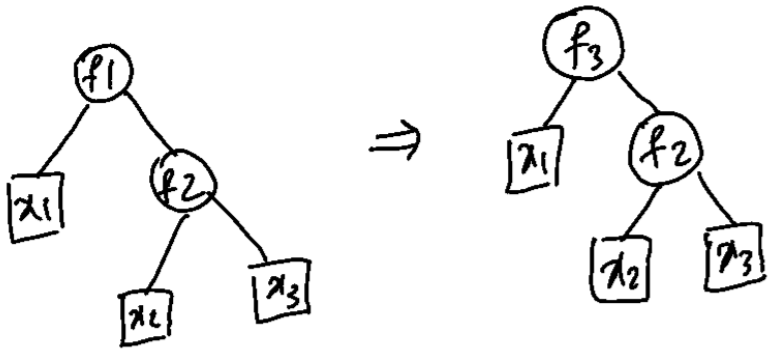
Genetic Programming Algorithm: Crossover

- For cross-over:
  - For every parent tree, we divide the tree into two partitions from one of the lines in the tree.
  - We take the partition containing the root from one of the parents, call it parent 1. We take the partition not containing the root from the other parent, call it parent 2.
  - We **transplant** the partition pf parent 2 to the partition of parent 2 to generate the offspring.
  - Depending on which parent to use as parent 1, we can have two offsprings.

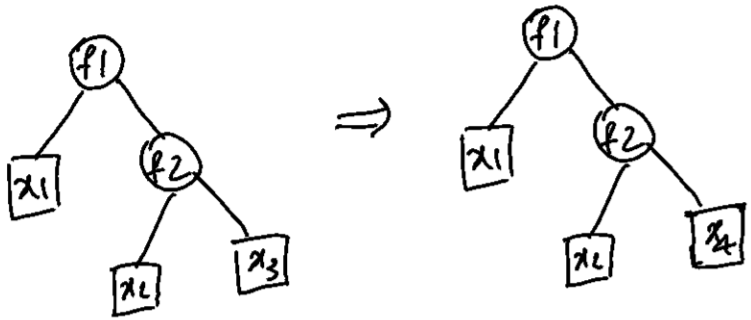


Genetic Programming Algorithm: Mutation

- For mutation, we can perform any of the following mutation methods:
  - Function mutation: we replace one or several function nodes with other functions in the function set.

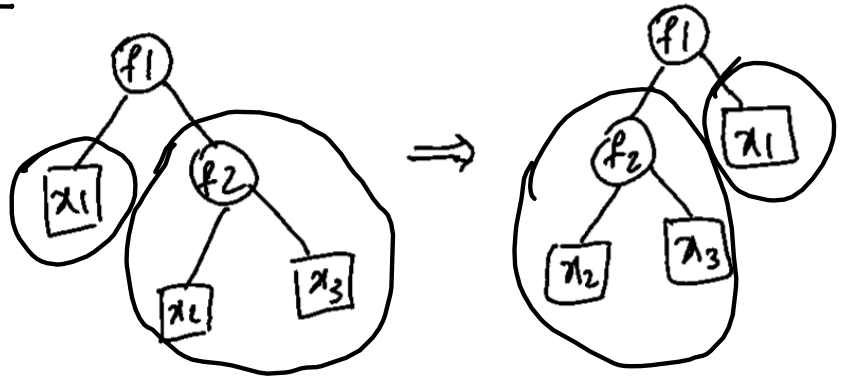


  - Terminal mutation: we replace one or several terminal nodes with other variables in the terminal set.

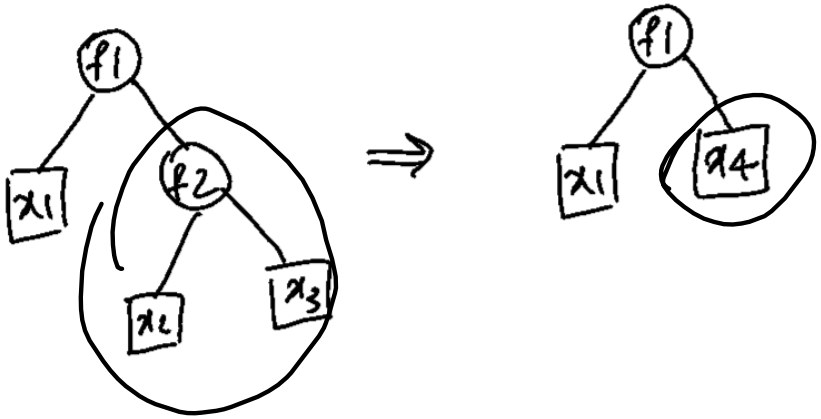


Genetic Programming Algorithm: Mutation

- For mutation, we can perform any of the following mutation methods:
  - Swapping mutation: for the functions sensitive to order of variables, we can swap their sub-trees.

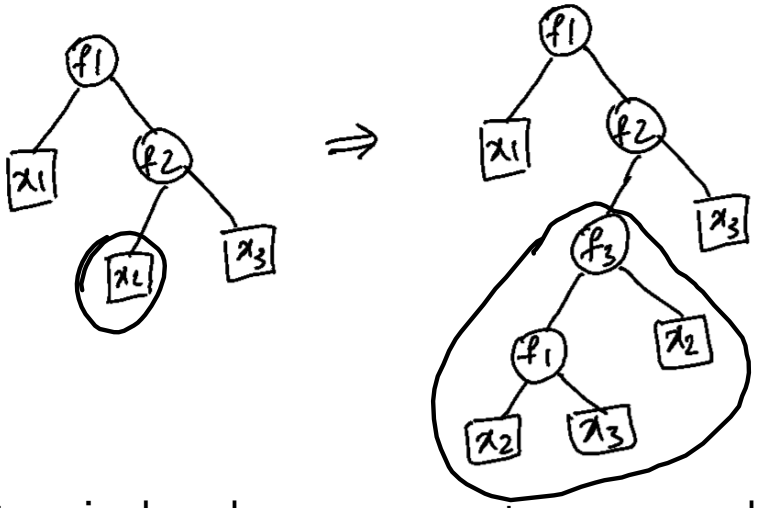


  - Truncation (or pruning) mutation: we consider a sub-tree in the tree, remove it, and replace it with a random terminal node from the terminal set.

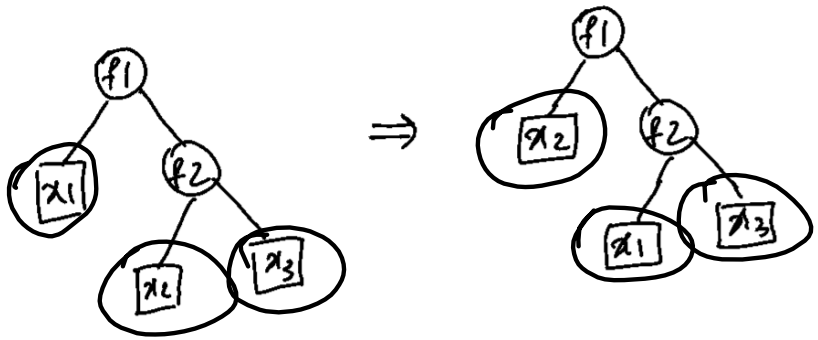


Genetic Programming Algorithm: Mutation

- For mutation, we can perform any of the following mutation methods:
  - Growing mutation: we consider a terminal node in the tree and grow it with some other random functions from the function set and random terminal nodes from the terminal set.

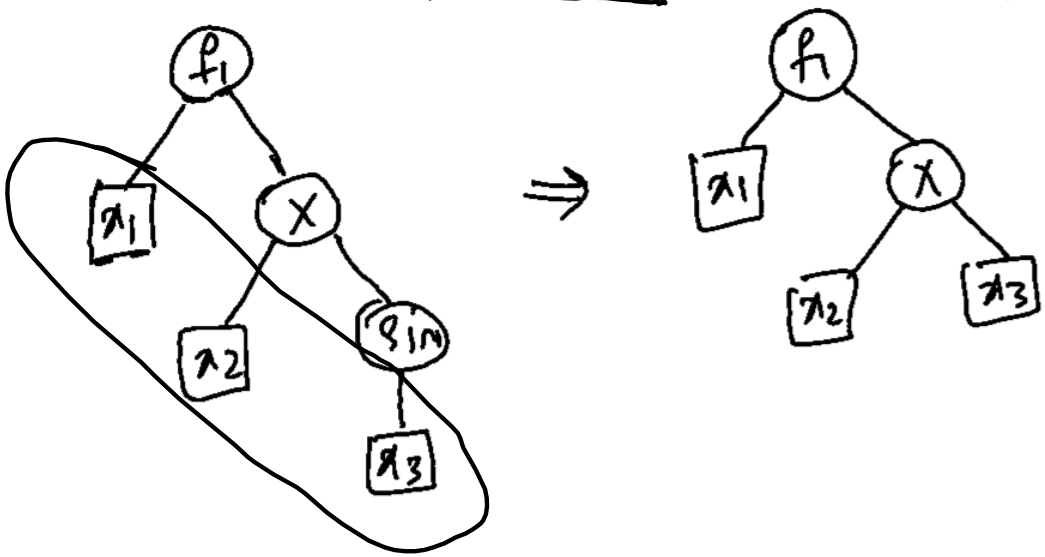


  - Permutation of terminal nodes: we permute some or all the terminal nodes randomly.



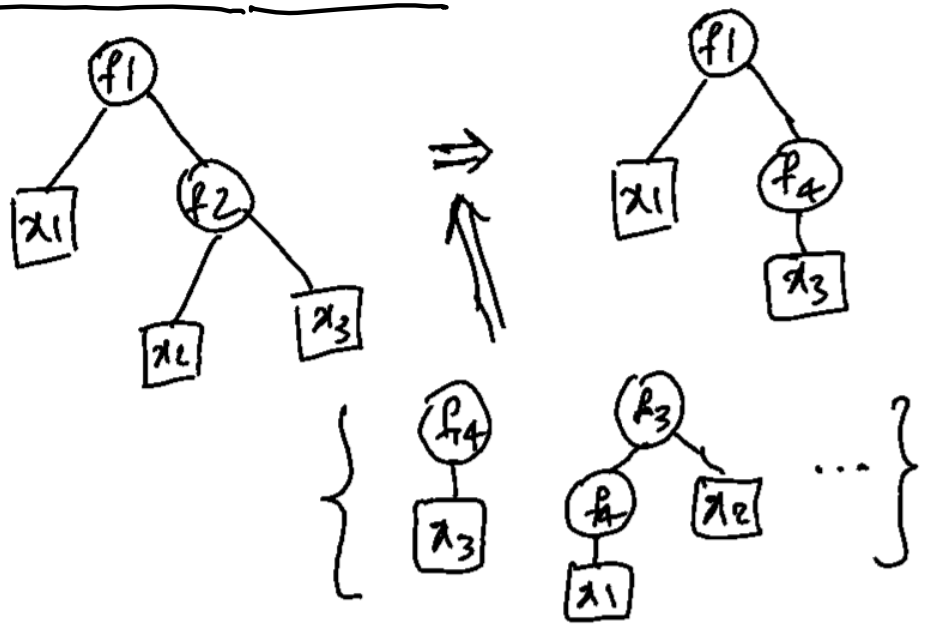
Genetic Programming Algorithm: Mutation

- For mutation, we can perform any of the following mutation methods:
  - Editing mutation: wherever we see a sub-tree which we know is not useful, we can edit it to a useful sub-tree. For this, we need to have hard rules for detecting useless functions. Example: replace $x_2 \times \sin(x_3)$ gets replaced by $x_2 \times x_3$.



Genetic Programming Algorithm: Mutation

- For mutation, we can perform any of the following mutation methods:
  - Building block operators: if we have a list of correct and useful building blocks (sub-trees) of operators, we can replace a sub-tree from the tree with a building block of operators. Note that there is also a variant of genetic programming, named building block genetic programming [2], which makes the trees from the building blocks or sub-tress from scratch.



Genetic Programming Algorithm

Algorithm Genetic Programming

Input: the truth table, the terminal set, the function set, the semantic rules

//Initialize the chromosomes (tree structures) $\{T_i\}_{i=1}^{n}$:
for *tree* $T_i \in \{T_1, \ldots T_n\}$ **do**
    $T_i \leftarrow$ Generate_Tree(depth_max, $\hat{\lambda}$, $\hat{p}$)
    Evaluate the cost of tree T_i
    if $cost(T_i) < best\_cost$ **then**
        best_cost $\leftarrow$ cost(T_i)
        best_solution $\leftarrow$ T_i

//Search iteratively:
while *not converged* **do**
    for *tree* $T_i \in \{T_1, \ldots T_n\}$ **do**
        Evaluate the cost of tree T_i
        if $cost(T_i) < best\_cost$ **then**
            best_cost $\leftarrow$ cost(T_i)
            best_solution $\leftarrow$ T_i
    Natural selection of best parents
    Cross over of parents
    Mutation of some trees
Return best_solution T

that feature is very important.

if it uses a feature several times

* GP does feature selection implicitely & intelligently.

* the idea of GP itself can be implemented by other algorithms.

Acknowledgment

- Some slides of this slide deck are inspired by teachings of Prof. Saeed Sharifian at the Amirkabir University of Technology, Department of Electrical Engineering.

References

[1] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp. 183–187, Psychology Press, 1985.

[2] N. F. McPhee, B. Ohs, and T. Hutchison, "Semantic building blocks in genetic programming," in *Genetic Programming: 11th European Conference, EuroGP, Naples, Italy*, pp. 134–145, Springer, 2008.