Lecture 11 - Genetic Programming

Yuhui Shi

CSE, SUSTech

Review of the Last Lecture

- Penalty Methods
- Stochastic Ranking
- Repair Functions
- Specialised Representations
- Decoder Functions

Outline of This Lecture

- Introduction
- Main Steps of GP and Its Operators
- Evolving Boolean N-multiplexer Functions using GP
- Multi-objective Genetic Programming (MOGP) Approaches
- Evaluation of MOGP Fitness Schemes

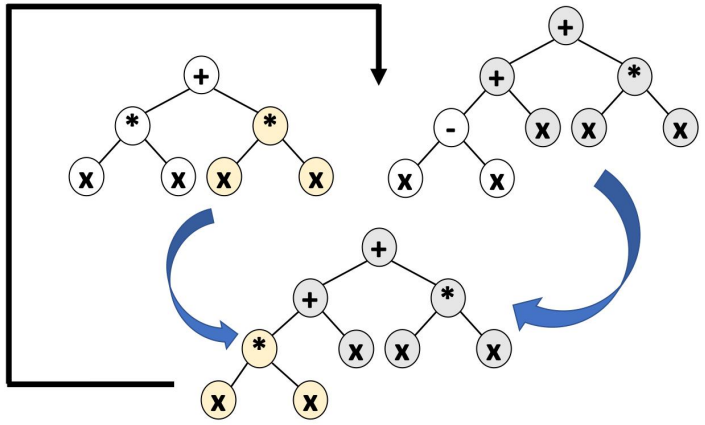
Outline of This Lecture

- **Introduction**
- Main Steps of GP and Its Operators
- Evolving Boolean N-multiplexer Functions using GP
- Multi-objective Genetic Programming (MOGP) Approaches
- Evaluation of MOGP Fitness Schemes

Genetic Programming (GP)

First used by de Garis to indicate the evolution of artificial neural networks [3], but used by Koza to indicate the application of GP to the evolution of computer programs [6].

1. Trees (especially Lisp expression trees) are often used to represent individuals.
2. Both crossover and mutation are used.

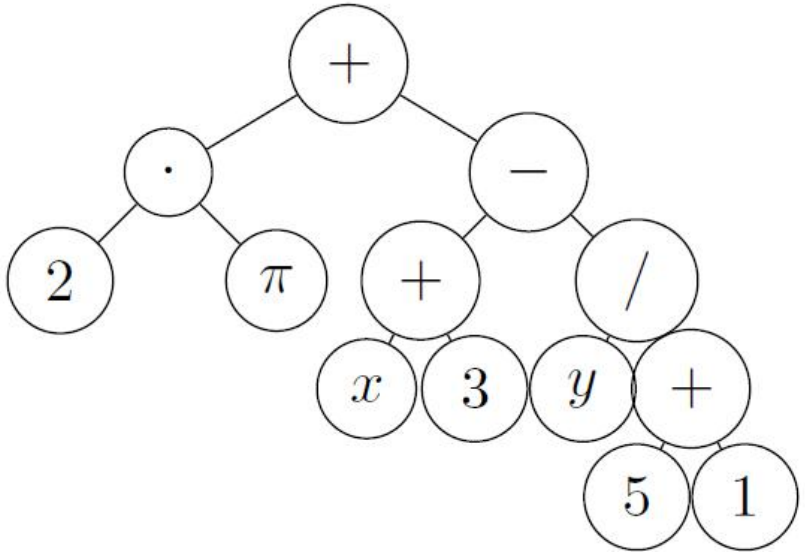


Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Figure 1: Left: illustration of GP. Right: Table 6.4 of [4].

Tree Representation: An Example

- Arithmetic formula: $2 \cdot \pi + ((x + 3) - y/(5+1))$
- Prefix notation
  - ✓Polish notation: +(·(2, π),−(+(x, 3), /(y, +(5, 1))))
  - ✓LISP notation: (+(· 2 π) (−(+ x 3)(/ y (+ 5 1))))
- Parse tree:
  - ✓Nodes (or points) indicate the instructions to execute.
    - ➢Internal nodes: functions.
    - ➢Leaves: terminals.
  - ✓Links indicate the arguments for each instrucption.



Outline of This Lecture

- Introduction
- **Main Steps of GP and Its Operators**
- Evolving Boolean N-multiplexer Functions using GP
- Multi-objective Genetic Programming (MOGP) Approaches
- Evaluation of MOGP Fitness Schemes

Preparatory Steps [7]

Before all, let's see what you need to prepare first . . .

1. The set of primitive functions F. → Define the search space.
   - ✓Example: arithmetic functions and conditional branching operators, or actions of a robot.
2. The set of terminals T.→ Define the search space.
   - ✓Example: independent variables (program's external input), zero-argument functions and numerical constants.
3. The (explicit or implicit) fitness function.→ Define the goal.
4. The algorithm control parameters.
   - ✓Example: population size and maximum size for programs.
5. The termination criterion/criteria.
   - ✓Example: maximum number of generations, or problem-specific success predicate.

Algorithm 1 Main Steps of GP [7].

Input: a set of functions F
Input: a set of terminals T
Input: a fitness measure
Input: control parameters
Randomly initialise a population of individuals pop
$t = 0$
while termination criterion is not met **do**
  for program $\in pop$ **do** ▶ Evaluation
    Execute individual program and obtain its fitness
  end for
  Select one or two individual(s) from pop with a probability based on fitness (with re-selection allowed)
  Create new individual(s) for the population by applying reproduction, crossover, mutation and architecture-altering operations with specified probabilities
end while
Return the best-so-far individual

Now, let's go through initialisation, crossover, mutation and architecture-altering operators one by one.

Applications of GP [10]

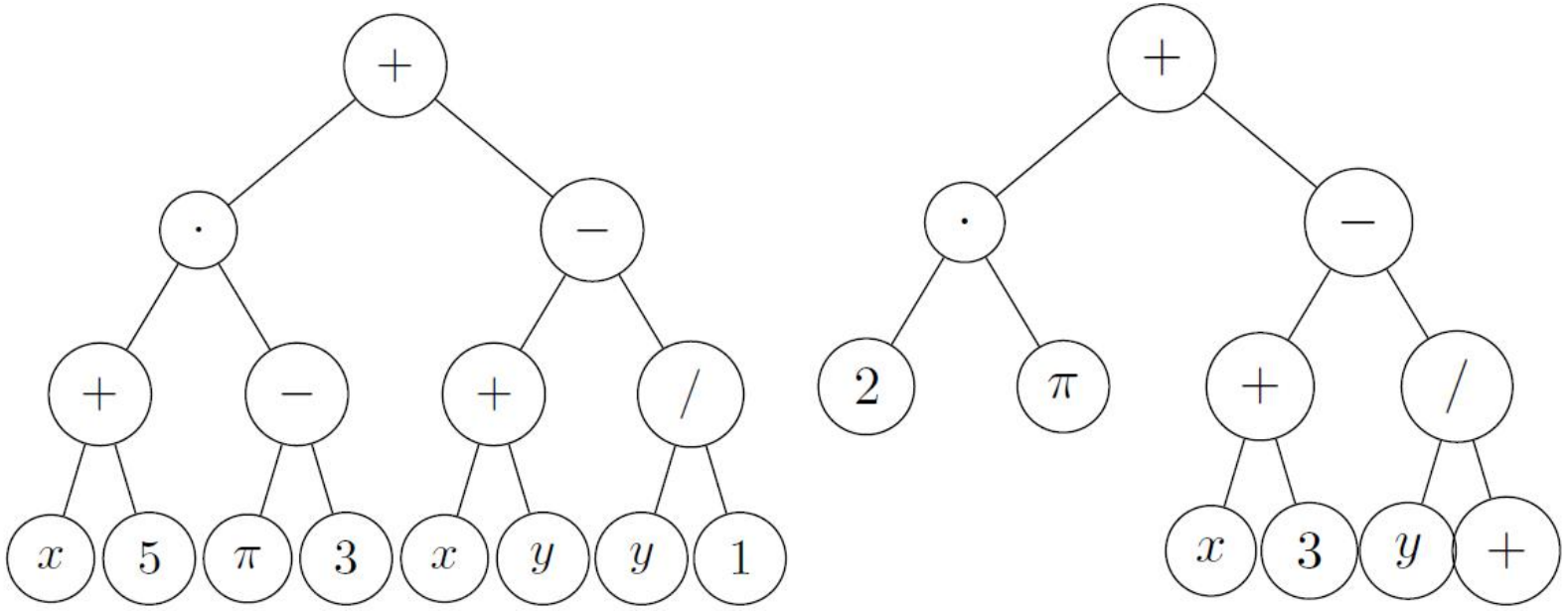
- Systems Modelling (e.g., approximation of models).
- Design (e.g., evolve structure of monomer/circuit).
- Control (e.g., robot control).
- Optimisation and scheduling (e.g., GP based facility layout scheme).
- Signal processing (e.g., evolve adaptive digital signal processing algorithms).

→ Seek models with maximum fit; evolve the syntax of arithmetic expressions, formulas in first-order predicate logic, or programs.

John R Koza, Forrest H Bennett III, and David Andre. Method and apparatus for automated design of complex structures using genetic programming. US Patent 5,867,397. 1999

Initialisation Methods I

- Full method (left): all the branches has depth D_{max}.
- Grow method (right): the branches may have different depths.



Initialisation Methods II

- Ramped half-and-half (混合法):
  - ✓Half of the population are generated using the full method, while using grow method for generating the others.
  - ✓Sometimes, each individual is generated using either method with equal probability.

Algorithm 2 Ramped half-and-half.

Input: a set of functions F
Input: a set of terminals T
Input: maximum depth D_{max}
for $i \in \{1, \ldots, \mu\}$ **do**
  if rand < 0.5 **then** ▶ Full method
    for $d \in \{1, \ldots, D_{max} - 1\}$ **do**
      The contents of nodes at depth d are chosen from F ▶ Select functions
    end for
    The contents of nodes at depth D_{max} are chosen from T ▶ Select terminals at leaves
  else ▶ Grow method
    The tree is constructed beginning from the root, with the contents of a node being chosen stochastically from $F \cup T$ if $d < D_{max}$
  end if
end for

Crossover & Mutation

- Crossover: Create new offspring by recombining randomly chosen parts from two selected parents.
- Mutation: Create one new offspring by randomly mutating a randomly chosen part of one selected parent.

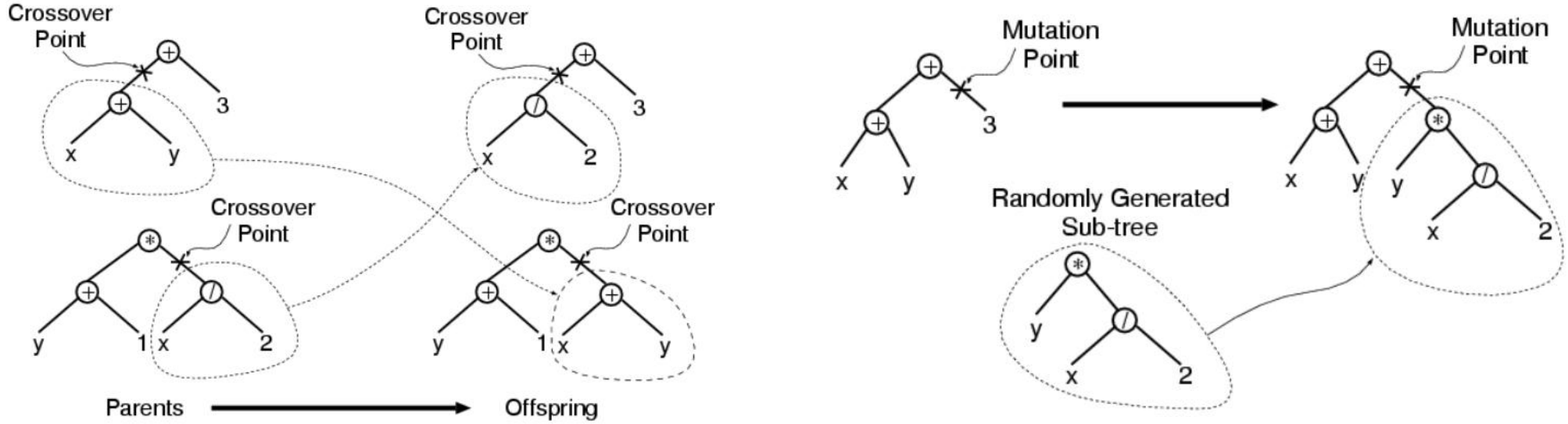


Figure 2: Figures 5.10 and 5.11 of [7].

Selection I

- Over-selection
  - ✓Often used to deal with the typically large population sizes.
  - ✓Large populations are not unusual in GP.
  - ✓Steps:
    1. Rank the population.
    2. Divide the population into two groups: G_A contains the top α% and G_B contains the (100 − α)%.
    3. Parent selection: 80% come from G_A while the rest come from G_B.
  - ✓How to select α?
    - ➢ α is found empirically.
    - ➢ α depends on the population size.
    - ➢ The selection pressure increases dramatically for larger populations.

      → #individuals from which the majority of parents are chosen stays a low constant value.

Selection II

- Bloat (膨胀): Average tree sizes tend to grow along the evolution.
  - ✓Also called the code growth or Survival of the fattest.
  - ✓Questions: Why does bloat happen? Is bloat good?
  - ✓Main techniques to control bloat:
    1. Parsimony pressure: Penalty term to reduce the fitness of large trees.
    2. Set a fixed limit on the size or depth of the tree.

       → Reject a tree if it is over-sized (consider as an infeasible solution). → Or set fitness to 0 if over-sized.
    3. Modify search operators.
    4. Multi-objective techniques (fitness and size).

  All sounds like Constraint Handling techniques!

Outline of This Lecture

- Introduction
- Main Steps of GP and Its Operators
- **Evolving Boolean N-multiplexer Functions using GP**
- Multi-objective Genetic Programming (MOGP) Approaches
- Evaluation of MOGP Fitness Schemes

Evolving Boolean N-multiplexer Functions

- Input: k address bits a_i and 2^k data bits d_i, where $N = k + 2^k$.

$$a_{k-1}, \ldots, a_1, a_0, d_{2^k-1}, \ldots, d_1, d_0.$$

- Output of Boolean multiplexer is the particular data bit that is singled out by the address bits (e.g., see Figure).

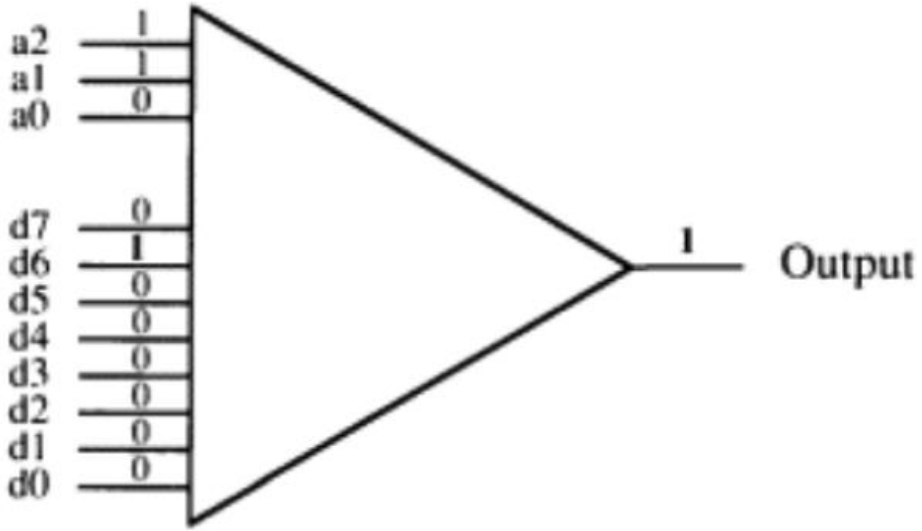


Figure 3: Figure 7.27 of [7]: Boolean 11-multiplexer with input of 11001000000 and output of 1, as the address bits are 110, which refers to d6.

Steps for Evolving a Boolean 11-multiplexer

1. Select the set of functions:

   $$F = \{AND, OR, NOT, IF\}$$

2. Select the set of terminals: 3+8=11 arguments.

   $$T = \{a_0, a_1, a_2, d_0, d_1, \ldots, d_7\}.$$

3. Identify the fitness measure for any evolved multiplexer m:
   - ✓ $fitness_{raw}$(m) = the number of x that m(x) == TrueValue(x).
4. Select the values of control parameters: PopSize = 4000.
5. Specify the criterion for designating a result and the termination criterion.

Summary Tableau for the Boolean 11-multiplexer

Objective:	Find a Boolean S-expression whose output is the same as the Boolean 11-multiplexer function.
Terminal set:	`A0, A1, A2, D0, D1, D2, D3, D4, D5, D6, D7.`
Function set:	`AND, OR, NOT, IF.`
Fitness cases:	The $2^{11} = 2{,}048$ combinations of the 11 Boolean arguments.
Raw fitness:	Number of fitness cases for which the S-expression matches correct output.
Standardized fitness:	Sum, taken over the $2^{11} = 2{,}048$ fitness cases, of the Hamming distances (i.e., number of mismatches). Standardized fitness equals 2,048 minus raw fitness for this problem.
Hits:	Equivalent to raw fitness for this problem.
Wrapper:	None.
Parameters:	$M = 4{,}000$ (with over-selection). $G = 51$.
Success predicate:	An S-expression scores 2,048 hits.

population size

maximum number of generations

Figure 4: Table 7.6 of [7]: Tableau for the Boolean 11-multiplexer problem.

Hits Histograms of Fitness Values

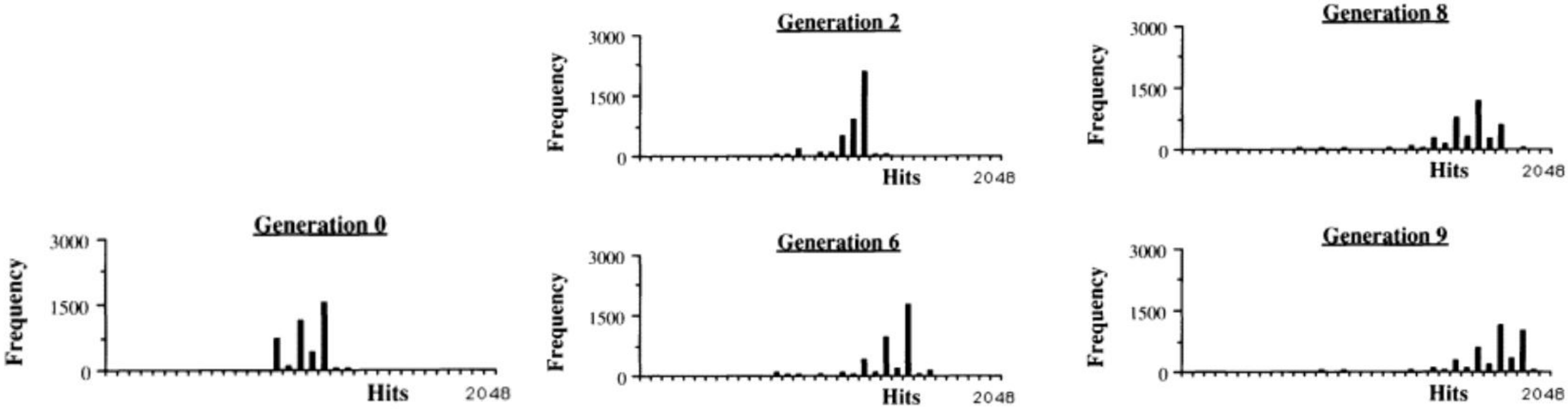
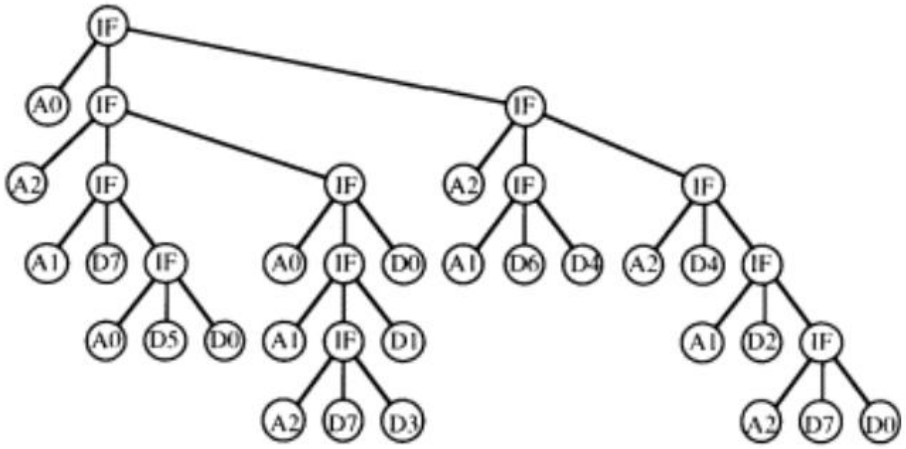


Figure 5: Figures 7.34 and 7.33 of [7]: Hits histograms for generations 0, 2, 6, 8, and 9 for the 11-multiplexer problem, and the best-of-run individual from generation 9 which solves the problem.



(IF A0 (IF A2 (IF A1 D7 (IF A0 D5 DO))(IF A0 (IF A1 (IF A2 D7 D3) D1)DO))(IF A2 (IF A1 D6 D4)(IF A2 D4(IF A1 D2 (IF A2 D7 DO)))))

Exercise

You will work on evolving functions for N-Parity problems in today's lab.

Outline of This Lecture

- Introduction
- Main Steps of GP and Its Operators
- Evolving Boolean N-multiplexer Functions using GP
- **Multi-objective Genetic Programming (MOGP) Approaches**
- Evaluation of MOGP Fitness Schemes

Motivation: Class Imbalance I

- Given a classification task, most machine learning (ML) methods assume that:
  - ✓Every class has the same misclassification cost.
  - ✓The distribution of data among different classes is balanced.
  - ✓The aim is to maximise the classification accuracy: $Acc = \frac{TP+TN}{TP+TN+FP+FN}$.

		Actual	
		Positive	Negative
Prediction	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

- Question: Is the following prediction good (Acc = 98.92%)?

	Positive	Negative
Positive	10,000	10
Negative	99	1

Motivation: Class Imbalance II

- Many real-world applications have very unbalanced distributions amon classes, such as medical diagnostics, fraud detection, fault detection.

  Sometimes, $TNR = \frac{TN}{FN+TN}$ is more important than $TPR = \frac{TP}{FP+TP}$.
- Minority class: rare cases, high misclassification cost.
- Quantifying cost is hard in practice.

Core Ingredient: GP to Evolve Classifiers

- A classifier is a mapping from an instance to a class ID.
  → A mapping can be a function and if-else blocks.
  → A function and if-else blocks can be represented as a tree.
  → A tree can be evolved by GP.
  → GP can be used to evolve classifiers.
- However, GP can also evolve classifiers biased toward the majority class when data are unbalanced.

Let's Put All Together

- GP
- Ensemble learning
- Multi-objective optimisation

⇒ Urvesh Bhowan et al. “Evolving diverse ensembles using genetic programming for classification with unbalanced data”. In: IEEE Transactions on Evolutionary Computation 17.3 (2013), pp. 368–386

Objectives and Fitness Design

- Baseline: Single-objective GP (SOGP).
  - ✓Fitness: *Ave = ωTPR + (1 − ω)TNR*.
    - ➢ Assume positive samples are majority.
    - ➢ Accuracy for majority: $TPR = \frac{TP}{TP+FP}$.
    - ➢ Accuracy for minority: $TNR = \frac{TN}{TN+FN}$.
  - ✓Issue: ω must be specified prior to the evolutionary search.
- Our approach: Multi-objective GP (MOGP) using Pareto-based fitness schemes.
  - ✓Two fitness functions: TPR and TNR.
  - ✓Dominance measures (use one of them):
    - ➢ SPEA2 (dominance rank and dominance count) and
    - ➢ NSGAII (dominance rank only).
  - ✓Trick: Crowding distance measure to encourage diverse solutions on frontier.

Outline of This Lecture

- Introduction
- Main Steps of GP and Its Operators
- Evolving Boolean N-multiplexer Functions using GP
- Multi-objective Genetic Programming (MOGP) Approaches
- **Evaluation of MOGP Fitness Schemes**

Research Question I

Question I: Do the designed MOGP approach with Pareto-based fitness schemes outperform the SOGP?

⇒ Evaluation of MOGP fitness schemes and comparison between MOGP and SOGP on evolving diverse-classifier.

Research Question II

Question II: Are our MOGP methods domain specific?

- Six selected benchmark data sets:
  - ✓ From different problem domains.
  - ✓ Varying levels of class imbalance.
  - ✓ Different complexities where some tasks are easily separable (e.g., Yst_2).
  - ✓ Well- vs. sparsely represented.
  - ✓ High vs. low dimensionality.
  - ✓ Binary vs. real-valued feature types.

Name	Classes (Minority/Majority)	Number of Examples			Imb. Ratio	Features	
		Total	Minority	Majority		No.	Type
Ion	Good/bad (ionosphere radar signal)	351	126 (35.8%)	225 (64.2%)	1:3	34	Real
Spt	Abnormal/normal (cardiac tomography scan)	267	55 (20.6%)	212 (79.4%)	1:4	22	Binary
Ped	Pedestrian/background (image cut-out)	24 800	4800 (19.4%)	20 000 (80.6%)	1:4	22	Real
Yst_1	*mit*/nontarget (protein sequence)	1482	244 (16.5%)	1238 (83.5%)	1:6	8	Real
Yst_2	*me3*/nontarget (protein sequence)	1482	163 (10.9%)	1319 (89.1%)	1:9	8	Real
Bal	Balanced/unbalanced (balance scale)	625	49 (7.8%)	576 (92.2%)	1:12	4	Integer

Figure 6: Unbalanced classification tasks used in the experiments. Table II of [1]. For each task, 50% of the examples in each class were randomly chosen for the training and the test sets. → Preserve the same class imbalance ratio.

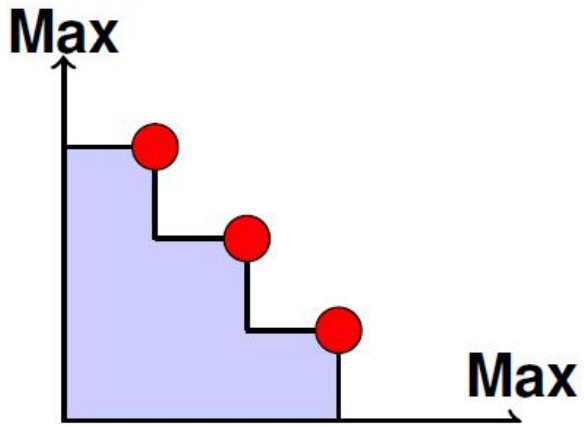
Research Question III

Question III: What quality indicators should be used?

Quality Indicator: Hyperarea (Hypervolume, HV)

- Hyperarea, (also known as the hypervolume) of the evolved Pareto-approximated fronts.
  - ✓Classic performance indicator !
  - ✓Area dominated by the Pareto-approximated solutions in the objective space.

    = sum of the areas of individual trapezoids fitted under each front solution in the objective space.



Take Home Message

- SPEA2's average hyperarea is statistically better than NSGAII on the three tasks, and not statistically different to NSGAII on the remaining three tasks.
- The hyperarea of the Pareto-optimal (PO) front is also better in SPEA2 for all tasks except Bal.
- The two MOGP approaches show similar average training times.

← Quality and time should both be considered!

Task	NSGAII Fitness			SPEA2 Fitness		
	Hyperarea		Training Time	Hyperarea		Training Time
	Average	PO Front		Average	PO Front	
Ion	0.793 ± 0.041	0.952	8.3 s ± 1.3	**0.848 ± 0.041**	**0.992**	9.3 s ± 2.4
Spt	0.733 ± 0.026	0.938	16.9 s ± 2.1	0.732 ± 0.032	**0.971**	9.7 s ± 2.5
Ped	0.881 ± 0.013	0.903	3.5 m ± 52.6	**0.902 ± 0.019**	**0.922**	3.9 m ± 1.1
Yst_1	0.793 ± 0.008	0.917	23.5 s ± 4.5	0.793 ± 0.009	**0.931**	20.8 s ± 7.1
Yst_2	0.942 ± 0.008	0.986	23.5 s ± 4.4	**0.949 ± 0.011**	**0.991**	20.1 s ± 8.1
Bal	0.749 ± 0.049	**0.993**	20.1 s ± 2.6	0.757 ± 0.063	0.985	15.2 s ± 3.9

Figure 7: Average Hyperarea of evolved Pareto-approximated fronts, Pareto-optimal (PO) fronts and training times for the MOGP over 50 independent trials. Table III of [1]. PO front is the set of nondominated solutions from the union of all Pareto-approximated fronts evolved from the 50 independent runs.

Observation: “SPEA2's average hyperarea is not statistically different to NSGAII on the three tasks: Spt, Yst1 and Bal.”

Task	NSGAII Fitness			SPEA2 Fitness		
	Hyperarea		Training Time	Hyperarea		Training Time
	Average	PO Front		Average	PO Front	
Ion	0.793 ± 0.041	0.952	8.3 s ± 1.3	**0.848 ± 0.041**	**0.992**	9.3 s ± 2.4
Spt	0.733 ± 0.026	0.938	16.9 s ± 2.1	0.732 ± 0.032	**0.971**	9.7 s ± 2.5
Ped	0.881 ± 0.013	0.903	3.5 m ± 52.6	**0.902 ± 0.019**	**0.922**	3.9 m ± 1.1
Yst_1	0.793 ± 0.008	0.917	23.5 s ± 4.5	0.793 ± 0.009	**0.931**	20.8 s ± 7.1
Yst_2	0.942 ± 0.008	0.986	23.5 s ± 4.4	**0.949 ± 0.011**	**0.991**	20.1 s ± 8.1
Bal	0.749 ± 0.049	**0.993**	20.1 s ± 2.6	0.757 ± 0.063	0.985	15.2 s ± 3.9

Question IV: What do their Pareto fronts look like? Can we approximate the fronts from independent runs?

Summary

- GP is good for evolving the syntax of arithmetic expressions, formulas in first-order predicate logic, programs or learners.
- MOGP is helpful for evolving classifiers that handle class imbalance.
- Local search can enhance GP /MOGP.
- Runtime is a problem!
  - ✓GP-based algorithms need much more time than traditional ML algorithms.
  - ✓MOGP methods with local search consumes more time than their counterparts without local search.

References for This Lecture I

1. Urvesh Bhowan et al. “Evolving diverse ensembles using genetic programming for classification with unbalanced data”. In: IEEE Transactions on Evolutionary Computation 17.3 (2013), pp. 368–386.
2. Urvesh Bhowan et al. “Reusing genetic programming for ensemble selection in classification of unbalanced data”. In: IEEE Transactions on Evolutionary Computation 18.6 (2014), pp. 893–908.
3. Hugo De Garis. “Genetic programming: Building artificial nervous systems using genetically programmed neural network modules”. In: Machine Learning Proceedings 1990. Elsevier, 1990, pp. 132–139.
4. Agoston E Eiben and James E Smith. Introduction to evolutionary computing. Vol. 53. Springer, 2003.
5. Christian Gagn´e et al. “Ensemble learning for free with evolutionary algorithms?” In: Proceedings of the 9th annual conference on Genetic and evolutionary computation. ACM. 2007, pp. 1782–1789.

References for This Lecture II

6. John R Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Vol. 34. Stanford University, Department of Computer Science Stanford, CA, 1990.
7. John R Koza. Genetic programming: on the programming of computers by means of natural selection. Vol. 1. MIT press, 1992.
8. John R Koza, Forrest H Bennett III, and David Andre. Method and apparatus for automated design of complex structures using genetic programming. US Patent 5,867,397. 1999.
9. Pu Wang et al. “Multiobjective genetic programming for maximizing ROC performance”. In: Neurocomputing 125 (2014), pp. 102–118.
10. M-J Willis et al. “Genetic programming: An introduction and survey of applications”. In: Second international conference on genetic algorithms in engineering systems: innovations and applications. IET. 1997, pp. 314–319.