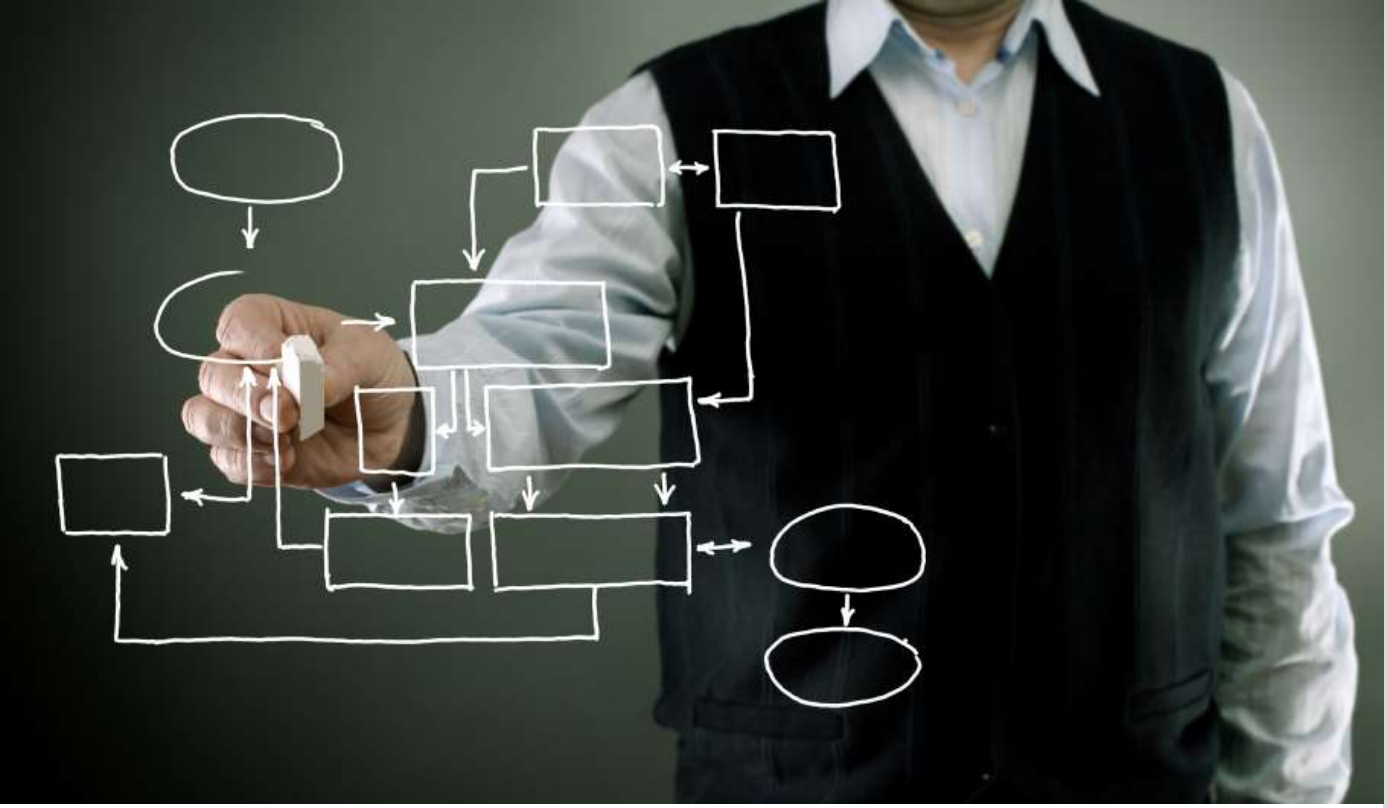


Hash-based Tree Similarity and Simplification in Genetic Programming for Symbolic Regression

Bogdan Burlacu, Lukas Kammerer, Michael Affenzeller and Gabriel Kronberger

Josef Ressel Centre for Symbolic Regression (JRZ)
University of Applied Sciences Upper Austria, Hagenberg (HEAL)

HEAL
HEURISTIC AND EVOLUTIONARY ALGORITHMS LABORATORY

SymReg
JOSEF RESSEL CENTER FOR SYMBOLIC REGRESSION

OBERÖSTERREICH

HeuristicLab

EuroCAST 2019

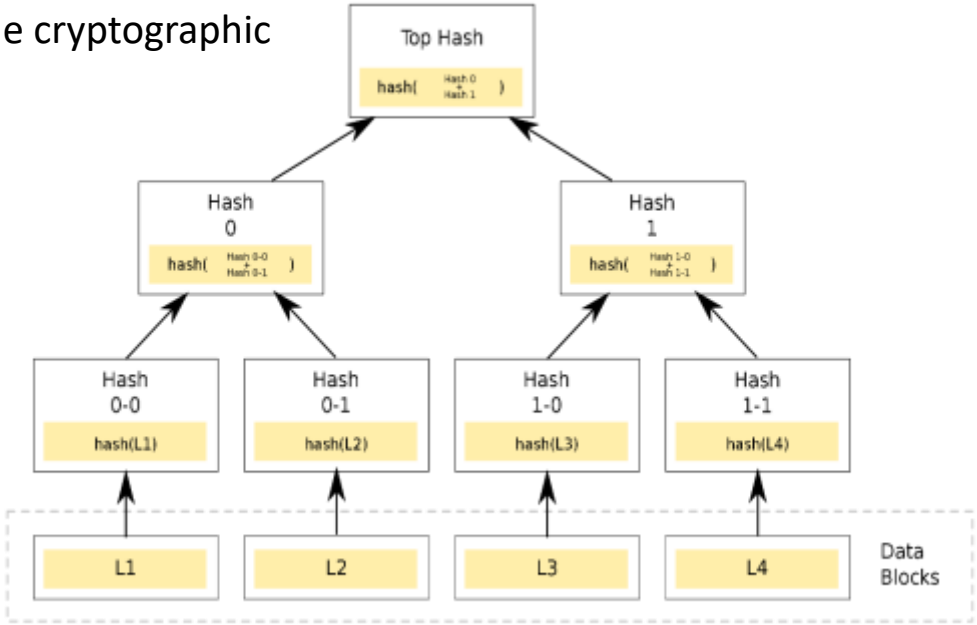
Contact:
Bogdan Burlacu
Heuristic and Evolutionary Algorithms Lab (HEAL)
Softwarepark 11
4232 Hagenberg, Austria

E-mail:
bogdan.burlacu@fh-hagenberg.at
Web:
http://heal.heuristiclab.com
http://dev.heuristiclab.com

Hashing of Symbolic Expressions

Similar in concept to Merkle trees

- Every leaf node is labelled with the hash of a data block
- Every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes



But different

- Initial node hash values
- Sorting of child nodes
- Non-cryptographic

Same hash → isomorphism

- Hash trees → postfix sequences
- Efficient comparison (=tree distance)

Hashing of Symbolic Expressions

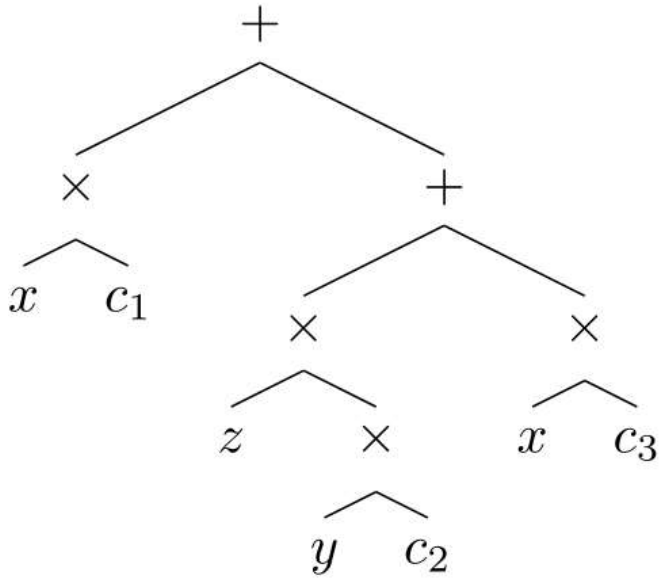
Algorithm 1: Tree hash algorithm

input : A symbolic expression tree T
output : A sequence of hash values for each node in T in postorder

```
1 nodes ← Postorder(T);
2 hashes ← empty hash value array;
3 foreach node n in nodes do
4     H(n) ← an initial hash value (for leaf nodes, it will remain unchanged);
5     if n is an internal node then
6         if n is commutative then
7             Sort the child nodes of n;
8         hashes ← Hash(Children(n), H(n));
9 return hashes;
```

Hashing Example (1)

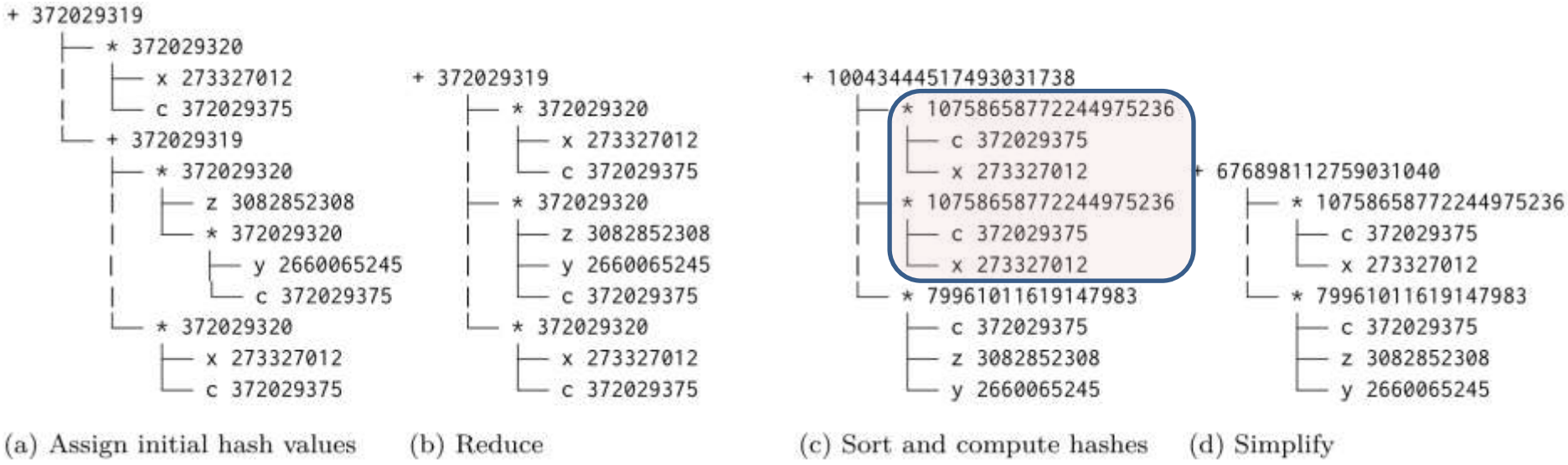
- $F(x, y, z) = c_1 x + c_2 yz + c_3 x$
  - Postfix representation: `c3 x * c2 y * z * + c1 x * +`
  - Tree representation




```
+ 372029319
    ├── * 372029320
    │   ├── x 273327012
    │   └── c 372029375
    └── + 372029319
        ├── * 372029320
        │   ├── z 3082852308
        │   └── * 372029320
        │       ├── y 2660065245
        │       └── c 372029375
        └── * 372029320
            ├── x 273327012
            └── c 372029375
```

Hashing Example (2)

- $F(x, y, z) = c_1 x + c_2 yz + c_3 x$
  - Steps (b) and (d) only required for simplification



- Tree hash value: `676898112759031040`
- Terms with the same hash value can be simplified according to arithmetic rules
- $F_{simplified}(x, y, z) = c_4 x + c_5 yz$ (original constants folded)

Fast Distance Matrix Calculation

Algorithm

1. Hash all trees into sequences of hash values
$$\{T_1, \dots, T_n\} \rightarrow \{H_1, \dots, H_n$$
2. Sort all hash sequences
3. Simple merge-count to calculate tree distance

Validation

- Identical results to bottom-up tree distance [Valiente, 2001]
- Improved performance
- Performance test: 5000 random trees

Algorithm 3: Merge-count hash values

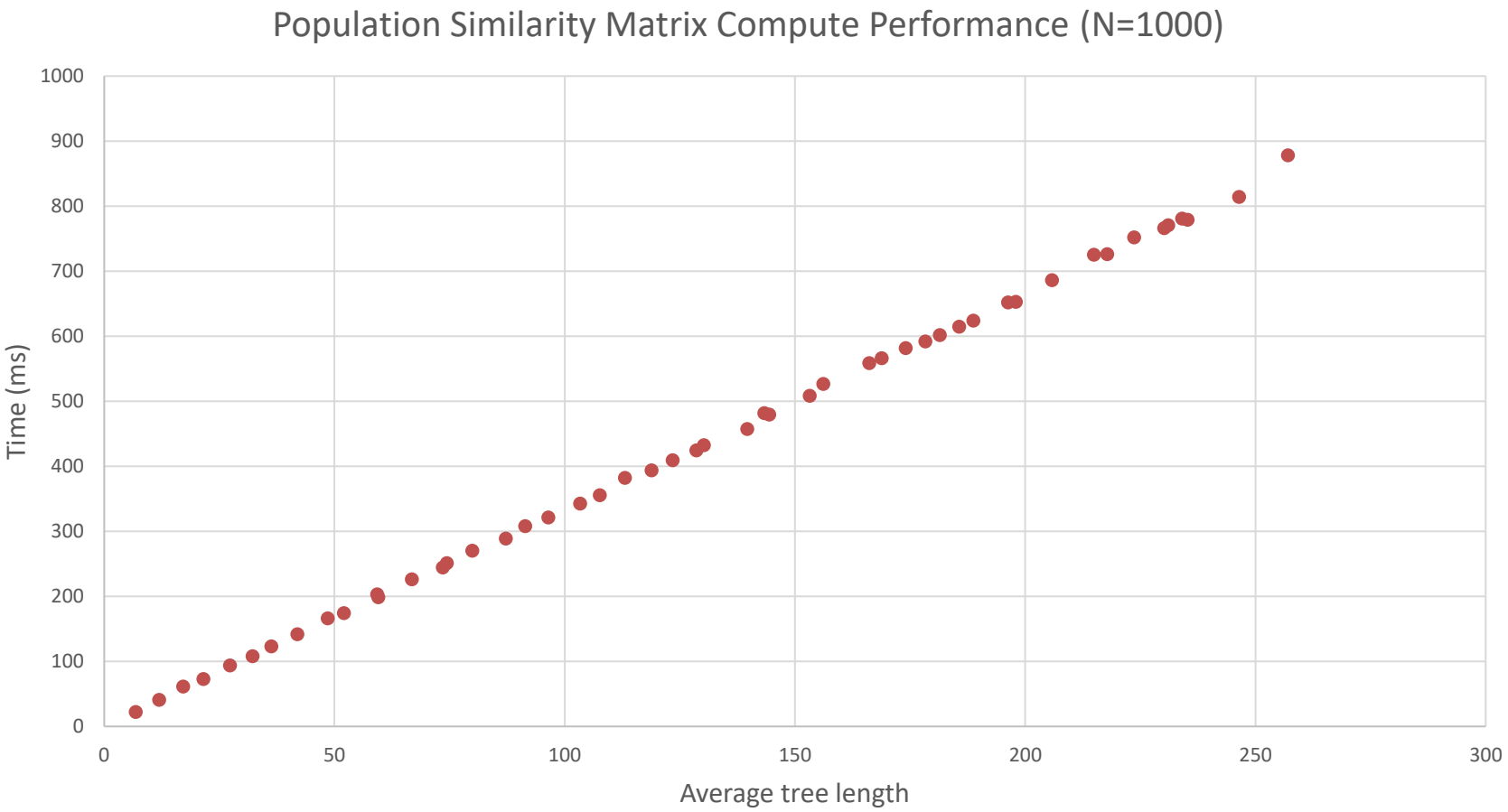
input : Two sorted hash arrays H_1 and H_2
output : The number of common hash values

```
1  i ← 0, j ← 0, count ← 0;
2  while i < |H1| and j < |H2| do
3      if H1[i] = H2[j] then
4          count ← count + 1;
5          i ← i + 1;
6          j ← j + 1;
7      else if H1[i] < H2[j] then
8          i ← i + 1;
9      else
10         j ← j + 1;
11 return 1 − 2 · count / (|H1| + |H2|)
```

TABLE I
RUNTIME PERFORMANCE OF HASH-BASED TREE DISTANCE VS THE BOTTOM-UP TREE DISTANCE

Tree distance method	Elapsed time (s)	Speed-up
Bottom-up	1225.751	1.0x
Hash-based (single-mode)	297.521	4.1x
Hash-based (batch-mode)	3.677	333.3x

Generational Overhead



Distance-based Diversity Control

Idea

- Compute distance matrix every generation
- For each individual, compute average distance to the rest of the population
- Promote diversity: focus on exploration, favour individuals that are farther from the rest

Structural and semantic diversity

- Tree distance inherently structural
- Tree semantics: include node coefficients in hash value calculation
- “strict” hashing → distinguish same structures with different semantics (coefficients)

Implementation

- Standard GA: weighted sum between fitness and average tree distance
- NSGA-II: average tree distance as secondary objective

Empirical Analysis

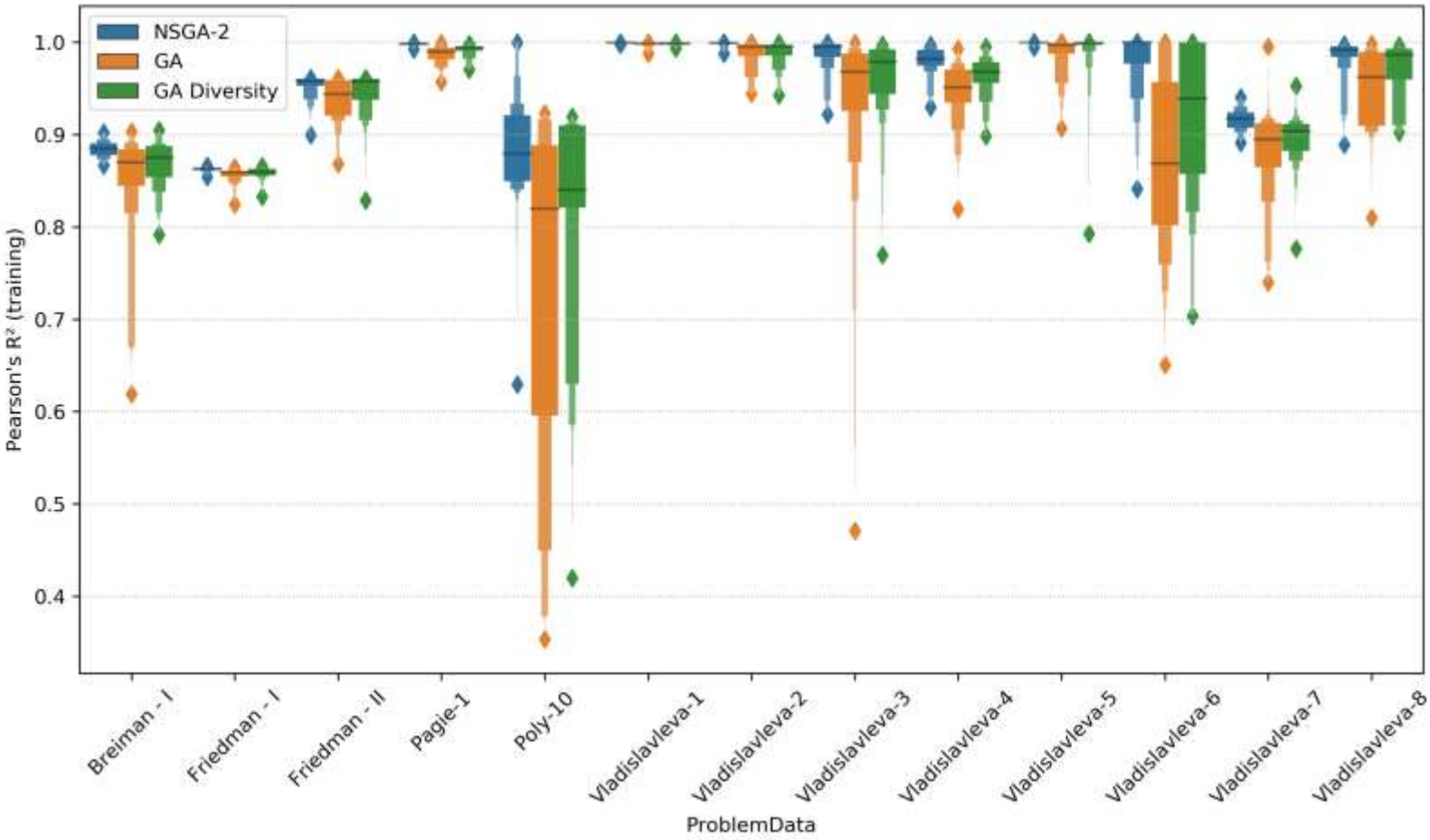
Algorithms: NSGA-2, GA, GA with diversity penalty term

Population size	1000
Generations	500
Crossover probability	100%
Crossover operator	Subtree crossover
Mutation probability	25%
Mutation operator	Remove branch, replace branch, change node type, one-point mutation
Selector	Tournament (GA) / Crowded tournament selector (NSGA-2), tournament size = 5
Tree initialization	Probabilistic tree creator (PTC2)
Maximum tree length	50
Maximum tree depth	12
Function set	$(+, -, \times, \div, exp, log, sin, cos, square)$

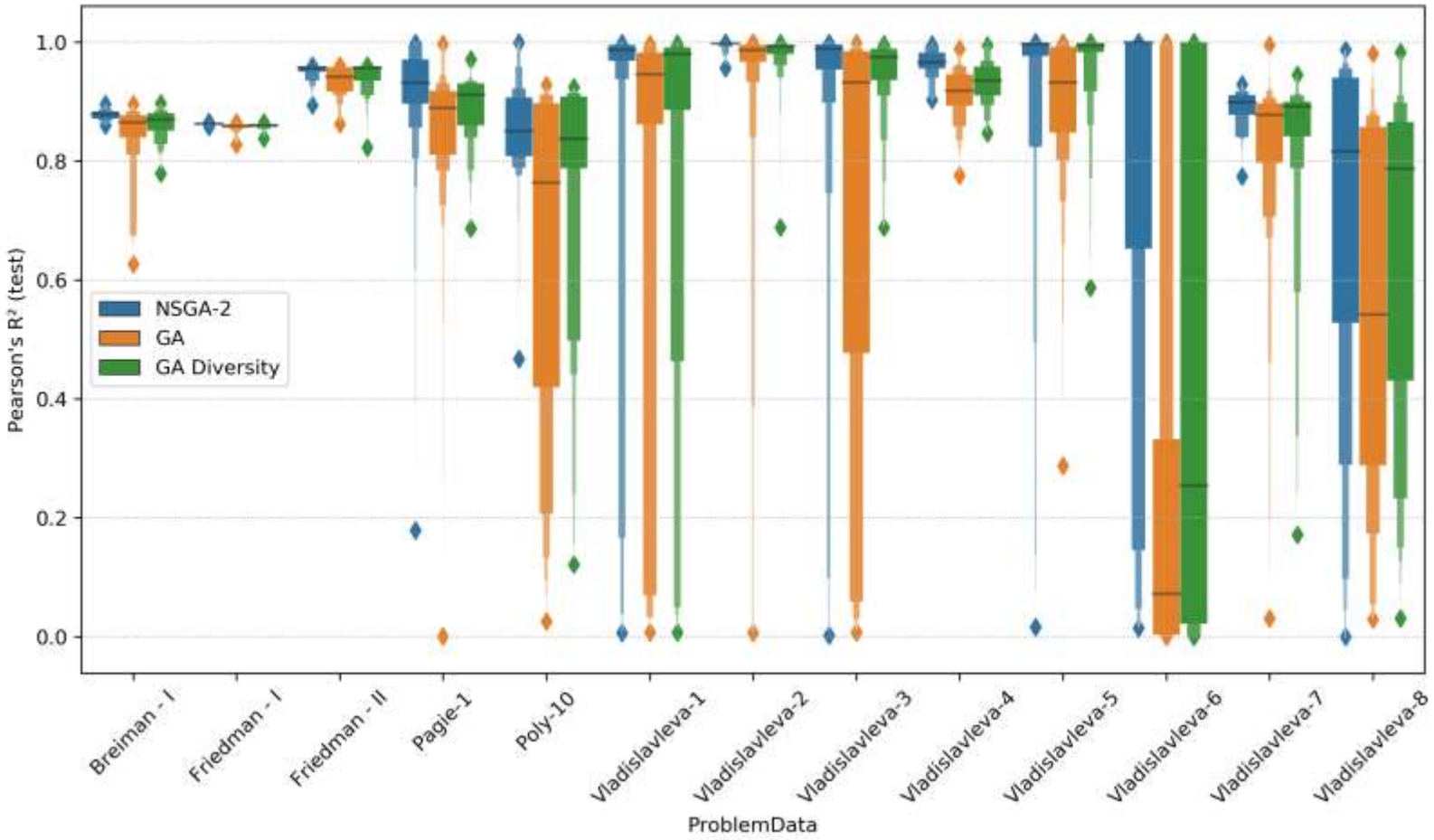
Benchmark problems:

- Vladislavleva $(F_1, \ldots, F_8)$
- Breiman-I, Friedman-I, Friedman-II
- Poly-10, Pagie-1

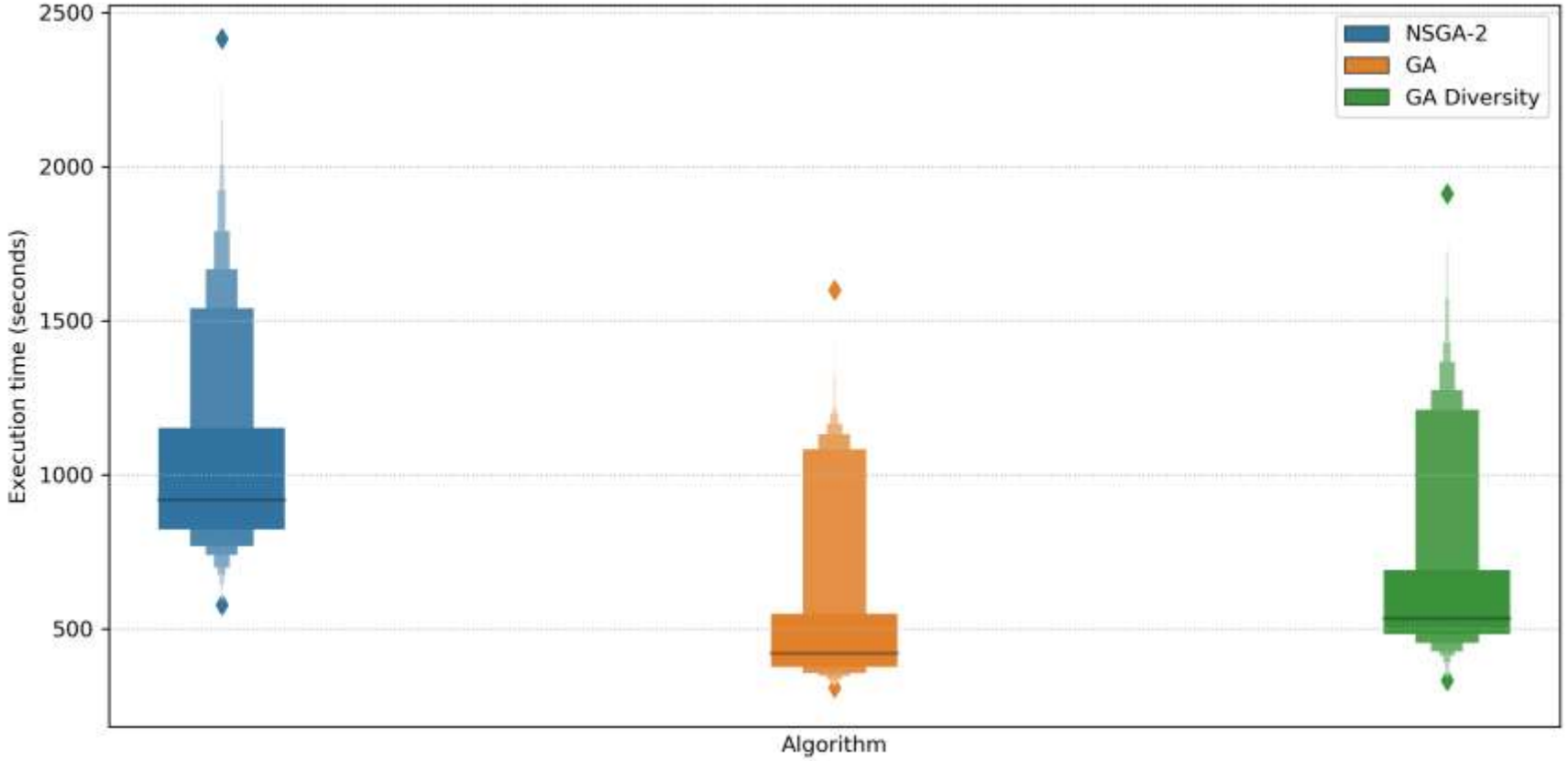
Training Solution Quality



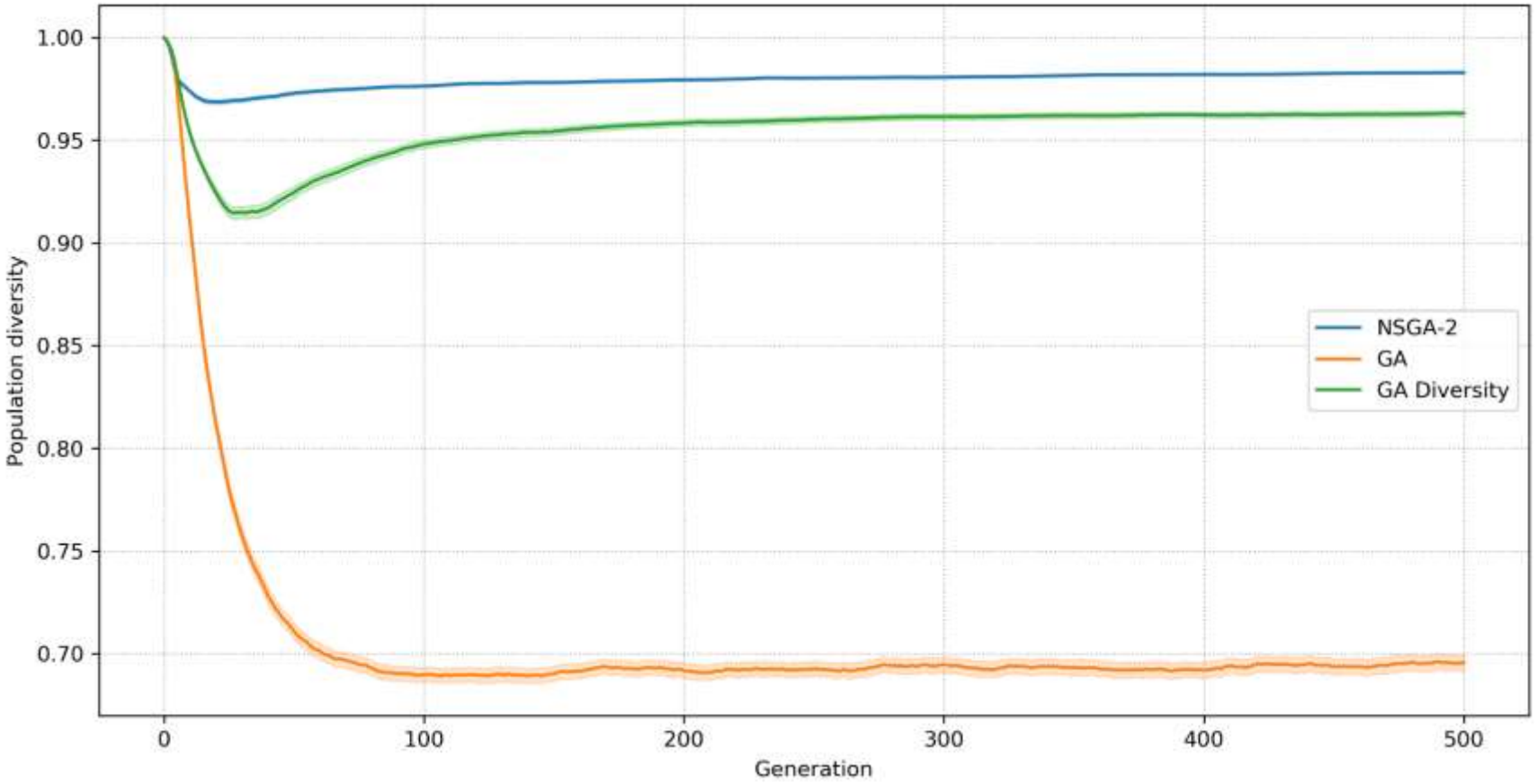
Test Solution Quality



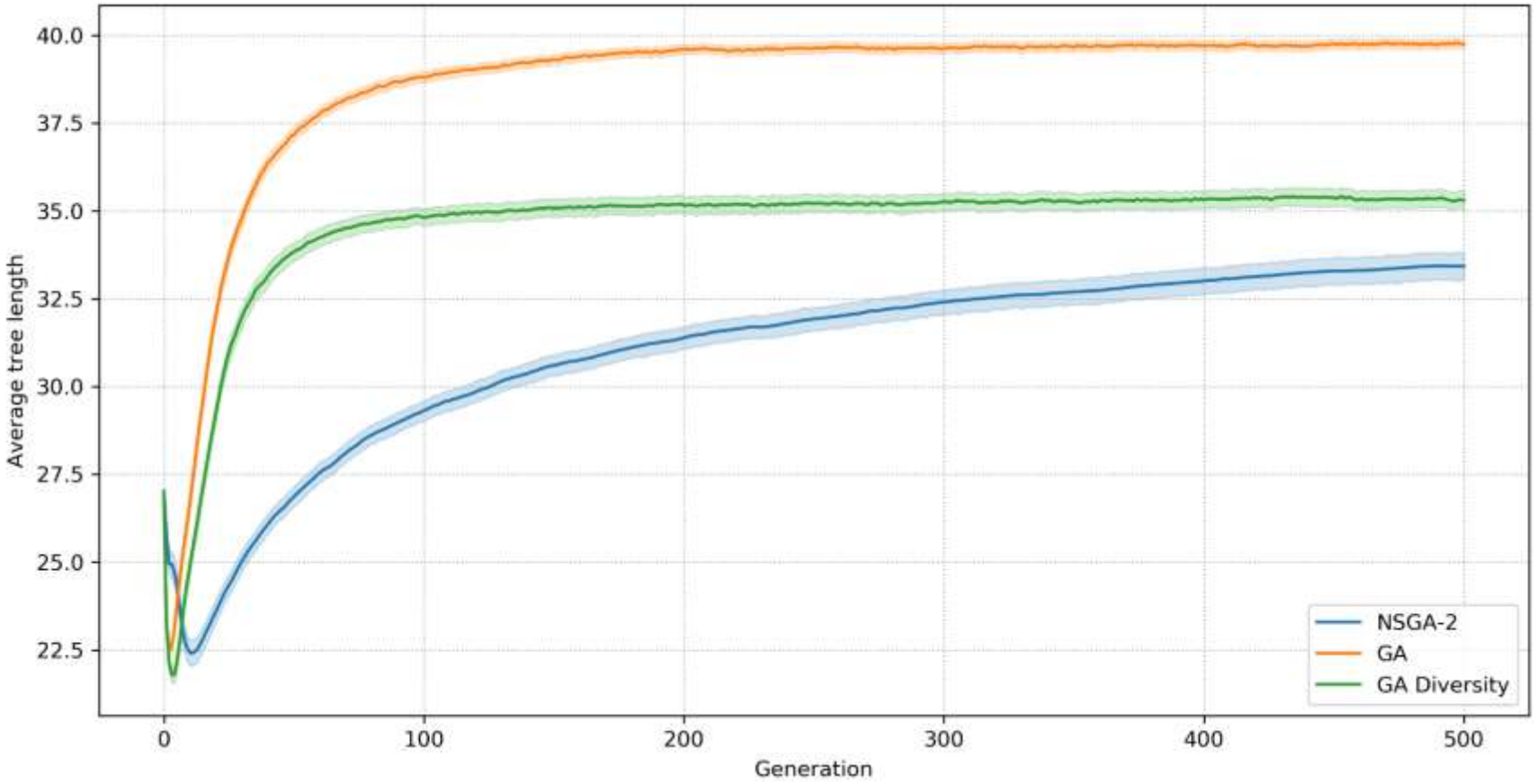
Execution Time



Population Diversity



Average Tree Length



Mining Common Subtrees

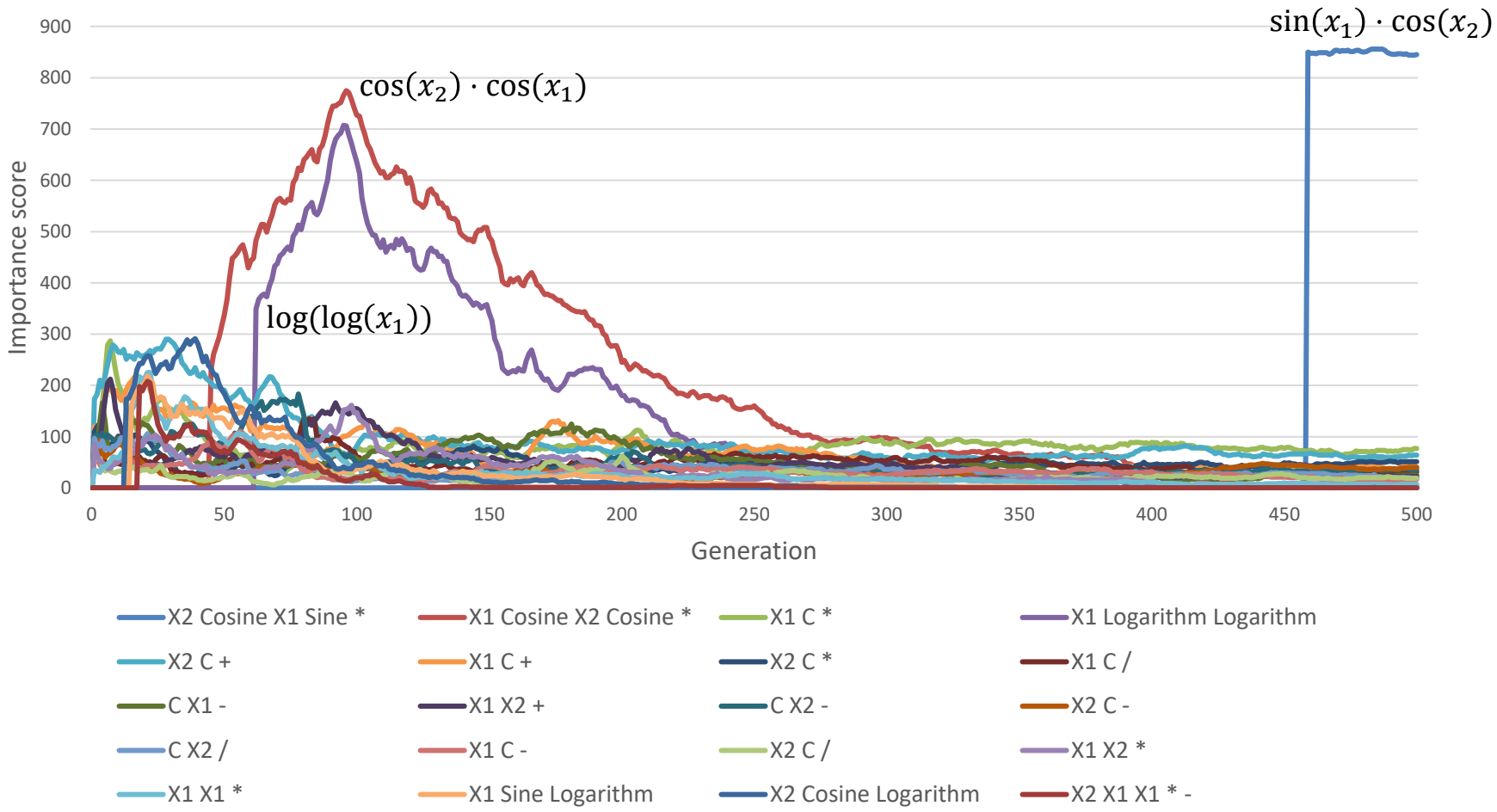
- **Hash entire population and count hash value frequencies**

- **Building blocks**
  - Building blocks discovered later in the run should count more
  - Use a weighted sum approach
  - Importance score calculated as

$$Score = \sum_{i=0}^{MAXGEN} i \cdot freq_i$$

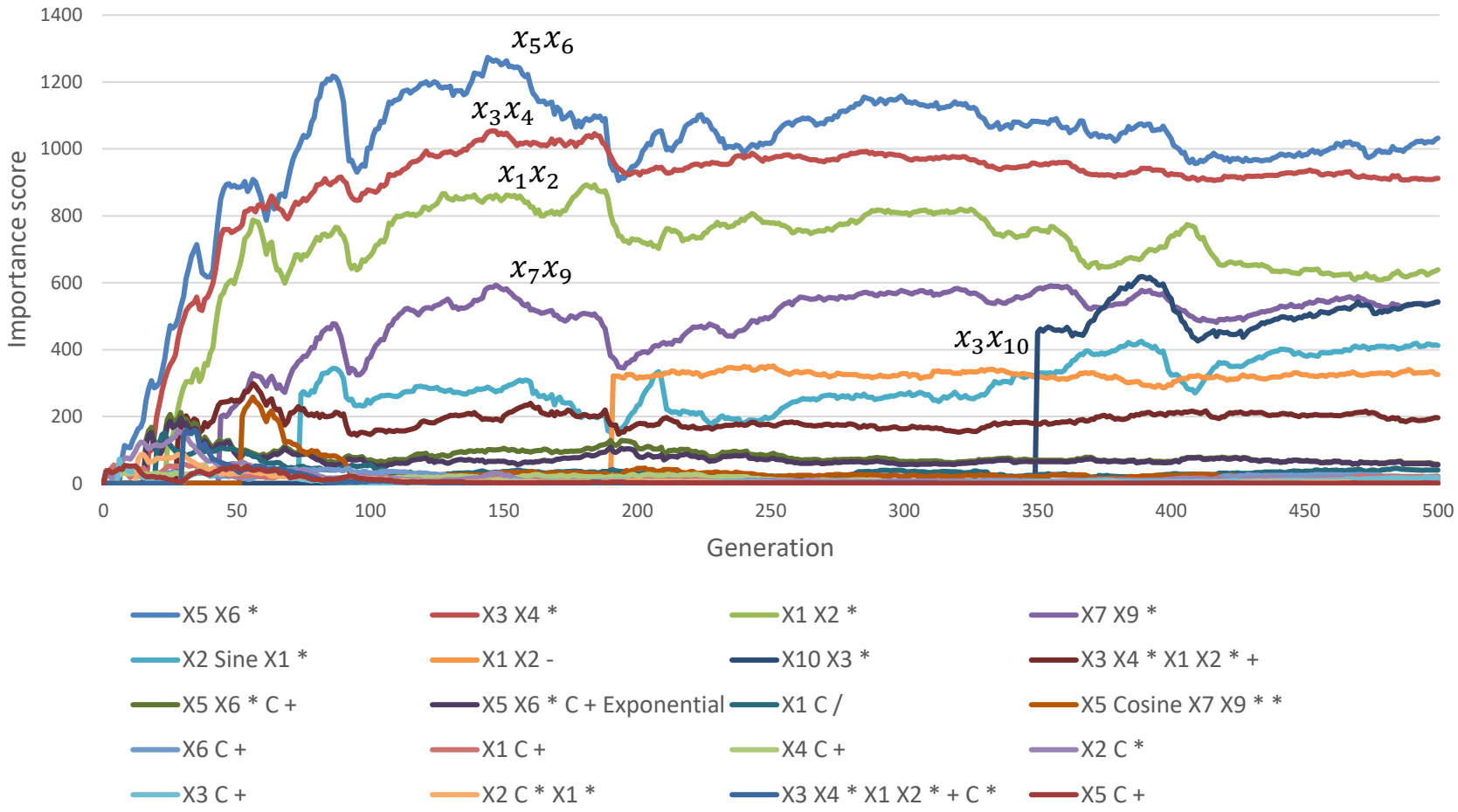
Most Common Subtrees – Example 1

Vladislavleva-6 $F(\mathbf{x}) = 6 \cdot \sin(x_1) \cdot \cos(x_2)$



Most Common Subtrees – Example 2

Poly-10 $F(\mathbf{x}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$



Conclusion

Efficient hashing method for expression trees

- Simple, linear data structures
- Sorting of node arguments for handling unordered trees and commutative symbols
- Each tree hashed only once
- Mining of common subtrees in the population

Used to identify equivalent expressions

- Simplification of same terms

Hash-based tree distance feasible to use online

- Incorporates tree semantics by hashing leaf node coefficients
- Diversity measure prevents premature convergence
- Helps create solutions with better generalization capabilities
- Suitable as secondary objective (NSGA-2)

Discussion

Hash-based Tree Similarity and Simplification in Genetic Programming for Symbolic Regression

Bogdan Burlacu, Lukas Kammerer, Michael Affenzeller and Gabriel Kronberger

Jossef Ressel Centre for Symbolic Regression (JRZ)
University of Applied Sciences Upper Austria, Hagenberg (HEAL)

HEAL
HEURISTIC AND EVOLUTIONARY ALGORITHMS LABORATORY

SymReg
JOSEF RESSEL CENTER FOR SYMBOLIC REGRESSION

OBERÖSTERREICH

HeuristicLab

EuroCAST 2019

Contact:
Bogdan Burlacu
Heuristic and Evolutionary Algorithms Lab (HEAL)
Softwarepark 11
4232 Hagenberg, Austria

E-mail:
Bogdan.burlacu@fh-hagenberg.at
Web:
http://heal.heuristiclab.com
http://dev.heuristiclab.com