Semantic Genetic Programming

Alberto Moraglio
University of Exeter
Exeter, UK
A.Moraglio@exeter.ac.uk

Krzysztof Krawiec
Poznan University of Technology
Poznan, Poland
krawiec@cs.put.poznan.pl
Instructors

• Alberto Moraglio
  – Position: Lecturer in Computer Science at the University of Exeter, UK
  – Research Area: founder of the Geometric Theory of Evolutionary Algorithms, which unifies Evolutionary Algorithms across representations and has been used for the principled design of new successful search algorithms, including a new form of Genetic Programming based on semantics, and for their rigorous theoretical analysis.

• Krzysztof Krawiec
  – Position: Associate Professor at Poznan University of Technology, Poland
  – Research Area: genetic programming and coevolutionary algorithms, with applications in program synthesis, modeling, image analysis, and games. Within GP: design of effective search operators (particularly crossovers), discovery of semantic modularity of programs, and exploitation of program execution traces for improving performance of program synthesis.
Aims

• Give a comprehensive overview of semantic methods in genetic programming

• Illustrate in an accessible way a formal geometric framework for program semantics

• Analyze rigorously their performance (runtime analysis)

• Present current challenges and trends in semantic GP

• Outline new emerging approaches
Agenda

1. Introduction to Semantic Genetic Programming
2. Geometric Operators on Semantic Space
3. Approximating Geometric Semantic Genetic Programming
4. Geometric Semantic Genetic Programming
5. Other Developments and Current Research Directions
I. Introduction to Semantic Genetic Programming
Genetic Programming

- **Generate-and test** approach to program synthesis
- Programs represented as **symbolic structures** (usually abstract syntax trees, ASTs)
- Population-based
- **Iterative**: start with a population of programs drawn at random, and repeat:
  - select the most promising individuals,
  - perturb using mutation and crossover
- ... until solution found
- This tutorial: focus on tree-based GP (but usually easily generalizable to other genres).
Motivations for Semantic GP (SGP)

• Traditional GP search operates directly on syntax, largely disregarding program semantics.

• Consequences:
  – Complex, rugged genotype-phenotype mapping
  – Low relatedness of offspring to parents
  – Slight change can dramatically change the output of the program
  – And conversely: high likelihood of no-effect (neutrality)
  – Low fitness-distance correlation
Questions

• Can we make GP more aware about the *effects* of program execution, i.e., *program ‘behavior’*?
• Can we *design search operators* that produce offspring program which behave similarly to parent(s)?
• Can we design search operators that are *guaranteed* to do so?
Program Semantics

- **Program semantics** = a formal method of capturing program behavior in abstraction from syntax.
- Common formalisms: denotational semantics, operational semantics.
  - Rarely applicable in GP, where program correctness typically expressed w.r.t. to fitness cases (tests).
- Note: semantics (noun) vs. semantic (adj.)
GP Semantics

- Problems in GP are typically posed using a set of *fitness cases* (*tests*).
- Observation: Program behavior is reflected in the *effects* of computation, i.e., program output.
- Program semantics in GP: the *tuple* (vector) of outputs for the training fitness cases. Example:

![Diagram of a tree structure with operations and values]

<table>
<thead>
<tr>
<th>x</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1.5</td>
<td>4.5</td>
</tr>
<tr>
<td>2.0</td>
<td>8.0</td>
</tr>
</tbody>
</table>

`semantics = [0.5, 2.0, 4.5, 8.0]`

- Important consequence: semantic $s(p)$ is a point in an *n*-dimensional space.
- A distance between $s(p_1)$ and $s(p_2)$ reflects *semantic similarity* of $p_1$ and $p_2$. 
Semantic Building Blocks

(McPhee, Ohs, Hutchison 2007/2008)

- Studied the impact of subtree crossover in terms of semantic building blocks.
- Describe the semantic action of crossover.
- Provide insight into what does (or doesn’t) make crossover effective.
- Define semantics of subtrees and semantics of contexts, where context = a tree with one branch missing.
- Definition of program semantics inspired by Poli's and Page's work on sub-machine code GP
Semantic Building Blocks

(McPhee, Ohs, Hutchison 2007/2008)

- Distribution of context semantics are key in the success (or failure) of runs.
- A very high proportion (typically over 75%) of crossover events are guaranteed to perform no useful search in the semantic space.

<table>
<thead>
<tr>
<th>Parent semantics</th>
<th>Arg semantics (x)</th>
<th>(and x #)</th>
<th>(or x #)</th>
<th>(nand x #)</th>
<th>(nor x #)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>+</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>+</td>
<td>1</td>
</tr>
</tbody>
</table>
Semantically-Driven Crossover (SDC)

(Beadle and Johnson 2008)

- Program semantics = reduced ordered binary decision diagram (ROBDDs)

- Trial-and error wrapper of tree-swapping crossover:
  - Pick a pair of parents and generate from them a potential offspring (candidate offspring)
  - Calculate ROBDD semantics of parents and offspring
  - Repeat if semantics the same as of any of the parents

Analogously: Semantically-driven mutation (SDM) (Beadle & Johnson 2009)
Semantic-Aware Crossovers

• Motivation: swap semantically similar subprograms in the parent programs, to ‘smoothen’ the semantic effect of crossover.

• **Semantic-aware crossover (SAX)** (Quang et al. 2011)
  
  – Select a pair of subprograms such that their semantics are sufficiently similar (upper limit on distance)

• **Semantic Similarity-based Crossover (SSX)** (Quang et al. 2011)
  
  – As SAX, but imposes also lower limit on distance between the subprograms, to prevent producing semantically neutral offspring (see efficiency later in this tutorial).

• (Quang et al. 2013): Picks the closest semantically different subprogram in the other parent.

• Analogous mutations defined too.
Semantic-Aware Initialization

Semantically-driven Initialization (Beadle and Johnson 2009)

• Constructs a population of semantically distinct programs of gradually increasing complexity.

• Start with population $P$ filled with all single-instruction programs

• To generate a new program:
  – Repeat:
    • Create a random program $p$ by combining a randomly selected non-terminal instruction $r$ (of arity $k$) with $k$ randomly selected programs in $P$
  – Until $p$ has a non-constant semantics that is sufficiently distant from semantics of all programs in $P$
  – Add $p$ to $P$ and return $p$
Semantic-Aware Initialization

• Behavioral Initialization (Jackson 2010)
  – Set $P \leftarrow \emptyset$

• To generate a new program:
  – Repeat:
    • Create a random program $p$ using conventional methods (e.g., Grow or Full)
  – Until the semantic of $p$ is sufficiently distant from semantics of all programs in $P$
  – Add $p$ to $P$ and return $p$

• Observation: Semantic diversity decreases rapidly with run progress (as opposed to syntactic/structural which increases and then levels-off)
II. Geometric Operators on Semantic Space
Metric Space

\[ d(x, y) \geq 0 \]
\[ d(x, y) = 0 \iff x = y \]
\[ d(x, y) = d(y, x) \]
\[ d(x, z) + d(z, y) \geq d(x, y) \]
Balls & Segments

\[ B(x; r) = \{ y \in S \mid d(x, y) \leq r \} \]

\[ [x; y] = \{ z \in S \mid d(x, z) + d(z, y) = d(x, y) \} \]
Squared Balls & Chunky Segments

**Balls**

- **B(000; 1)**
  - Hamming space

- **B((3, 3); 1)**
  - Euclidean space

- **B((3, 3); 1)**
  - Manhattan space

**Line segments**

- **[000; 011] = [001; 010]**
  - 2 geodesics
  - Hamming space

- **[(1, 1); (3, 2)]**
  - 1 geodesic
  - Euclidean space

- **[(1, 1); (3, 2)] = [(1, 2); (3, 1)]**
  - infinitely many geodesics
  - Manhattan space
Geometric Crossover & Mutation

• **Geometric crossover**: a recombination operator is a geometric crossover under the metric $d$ if all its offspring are in the $d$-metric segment between its parents.

• **Geometric mutation**: a mutation operator is a r-geometric mutation under the metric $d$ if all its offspring are in the $d$-ball of radius $r$ centred in the parent.
Example of Geometric Mutation

Traditional one-point mutation is 1-geometric under Hamming distance.

Neighbourhood structure naturally associated with the shortest path distance.
Example of Geometric Crossover

• Geometric crossover: offspring are in a segment between parents for some distance.
• The traditional crossover is geometric under the Hamming distance.

\[ H(A,X) + H(X,B) = H(A,B) \]
Significance of Geometric View

- Unification Across Representations
- Simple Landscape for Crossover
- Crossover Principled Design
- Principled Generalisation of Search Algorithms
- General Theory Across Representations
Semantic Operators

- Semantic search operators: operators that act on the syntax of the programs but that guarantee that some semantic criterion holds (e.g., semantic mutation: offspring are semantically similar to parents)
Fitness as Distance

- **Aim**: we want to find a function that scores perfectly on a given set of input-output examples (test cases)
- **Error of a program**: number of mismatches on the test cases
- **Fitness as distance**: the error of a program can be interpreted as the distance of the output vector of the program to the target output vector
- **Distance functions**: Hamming distance for Boolean outputs, Euclidean distance for continuous outputs
Semantic Distance & Operators

- The **semantic distance** between two functions is the distance of their output vectors measured with the distance function used in the definition of the fitness function.
- **Semantic geometric operators** are geometric operators defined on the metric space of functions endowed with the semantic distance.
Semantic Fitness Landscape

- The fitness landscape seen by GP with semantic geometric operators is always a cone landscape by definition (unimodal with a linear gradient) which GP can easily optimise!
III. Approximating Geometric Semantic GP
Trial-and-Error Geometric Crossover (KLX)

Krawiec and Lichocki Crossover, KLX (Krawiec and Lichocki 2009)

• Goal: Minimize offspring’s total semantic distance from the parents under some assumed metric \( || \ || \).

• Technical realization: Mate the parents \((x,y)\) repetitively using a ‘regular’ crossover operator \( CX \)

• Calculate parent semantics \( s(p_1), s(p_2) \)

• Repeat:
  – Apply \( CX \) to \((p_1,p_2)\) \( n \) times, creating a pool of candidates \( C \)
  – Calculate the semantics \( s(z) \) of each candidate \( z \in C \)

• Return the candidate \( z \) that minimises the total distance:
  \[
  \arg\min \ ||s(z) - s(p_1)|| + ||s(z) - s(p_2)||
  \]

• A form of \textit{brood selection}
Trial-and-Error Geometric Crossover (KLX)

Motivation: Given a globally convex fitness landscape (one global optimum), solutions on a segment connecting solutions $x$ and $y$ cannot be worse than the worse of them.
Promotion of Equidistance

- All candidate offspring on the segment \([s(p_1), s(p_2)]\) minimize total distance equally well, no matter how different from the parents they are.

  - An offspring \(z\) that is a ‘semantic clone’ of \(p_1\) \((s(z) = s(p_1))\) also minimises the total distance.

  - The likelihood of crossover producing a semantic clone of one of the parents is high in GP (see remarks on neutrality later)

- KLX promotes similarity to parents. This may hamper exploration.

- Idea: Extend total distance by a term that promotes balanced distance from both parents (KLX+)

\[
\text{argmin } ||s(z) - s(p_1)|| + ||s(z) - s(p_2)|| + ||s(z) - s(p_1)|| - ||s(z) - s(p_2)||
\]
Locally Geometric Crossover

(Krawiec & Pawlak 2012)

• Motivations: Finding an ‘almost geometric’ offspring can be difficult for entire parent programs,
  – … but should be easier for subprograms.
  – This may make sense if ‘geometricity’ can propagate through a tree.

• The algorithm:
  – Find the syntactic common region of the parents (where the trees overlap)
  – Select two homogenous nodes (subprograms) $p_1$ and $p_2$ in the common regions
  – Calculate the midpoint $s_m$ between $s(p_1)$ and $s(p_2)$
  – Find two programs $p'_1$ and $p'_2$ in a library that have the closest semantic distance from $s_m$
  – Replace $p_1$ and $p_2$ with $p'_1$ and $p'_2$, respectively.
Semantic Backpropagation

• Motivation: many instructions used in GP are invertible or partially invertible.

• Example: symbolic regression:
  – Fully invertible: e.g., addition: \( y = x + c \Rightarrow x = y - c \)
  – Partially invertible: e.g., square: \( y = x^2 \Rightarrow x = \pm \sqrt{x} \)

• The desired output \( t \) of a program (target) is known.

• Given a program and \( t \), this allows deriving \textit{desired semantics} at any point in a program tree.
Semantic Backpropagation

SBP can be used to back propagate any semantics.
Semantic Backpropagation

• Note: desired semantics is not a vector of scalar values.

• Desired semantics is a tuple of sets of desired outputs, because not all instructions are bijective. Examples:
  – \( D = (\{2\}, \{3\}, \{2,-4\}, \{0, 1\}) \)
  – \( D = (\{T\}, \{F\}, \{T,F\}) \)

• Special case: non-realizable desired semantics, e.g., \( D = (\{T\}, \emptyset, \{T,F\}) \)
  – Or: non-realizable under assumed constraints (e.g., size of subprogram).

• Algorithms have to account for that.
Propagation of Desired Semantics

- Two fitness cases, 2D semantic space
- Desired outputs: (0,0)
- Program: \( \cos(\sin(x)) \)
- Visualization:
  - semantic distance as a function of inputs \((x_1, x_2)\)
  - red = smaller semantic distance (greater fitness)
Propagation of Desired Semantics

• Top: desired semantics of \( \cos(#) \)
  
  – target achieved for \( x_1, x_2 = \pi + k\pi, \ k \in \mathbb{Z} \)

• Bottom: desired semantics of \( \cos(\sin(#)) \)
  
  – Target cannot be achieved, because \( \sin \in [-1,1] \), and thus no \( x \) causes \( \cos(\sin(x)) = 0 \)
Operators Based on SBP

- **Approximately Geometric Crossover, AGX** (Krawiec & Pawlak 2013)
  - A *crossover* operator
  - Uses SBP to match the midpoint on the segment connecting the parents’ semantics
  - Starting point of SBP: the midpoint on the segment

- **Random Desired Operator, RDO** (Wieloch & Krawiec 2013)
  - A *mutation* operator
  - Uses SBP to match the target of the search process
  - Starting point of SBP: the target semantics of the
Operators Based on SBP

• Common part of workflow:
  – Pick a node $p'$ in a parent $p$
  – Perform semantic backpropagation of desired semantics from the root of $p$ to $p'$, obtaining desired semantics $D$
  – Replace $p'$ with a (sub)program from a library that best matches $D$

• Other differences:
  – RDO is agnostic about geometric considerations
  – RDO and AGX may use various libraries
AGX: Some Results

(Pawlak, Wieloch, Krawiec, 2014)
Library of Subprograms

- The source of subprograms for SBP
  - **Static**: Generated prior to run
  - **Dynamic**: Other programs in the current population
- Example of static library: All programs built upon given set of instructions.
  - Instructions \{+, −, ×, /, sin, cos, exp, log, x\}, max tree height \( h \)
  - Semantic duplicates eliminated
- Total number of programs: 212 (for \( h = 3 \)), 108520 (for \( h = 4 \))
  - Depends on the instruction set and tests (in general the fewer tests, the fewer unique semantics)
  - Impact of floating-point precision
Semantic Diversity of Libraries

Exemplary library:

- All programs composed of \{+,−,x,/,sin,exp,x\}, max tree depth: 4.
- Semantics: 20 points distributed equidistantly in \([-5, 5]\) ⇒ 20-dimensional semantic space
- Semantic duplicates removed.

Visualization:

- Reduction to 2D by PCA,
- Red: the smallest (i.e. single node) programs,
- Blue: the longest (i.e. 15 nodes) programs.

Observation: strongly non-uniform distribution of semantics.

- Expected: see (Langdon & Poli 2002)
Technical Challenges of SBP

- **Limited semantic diversity**
  - Using a mutation operator in parallel recommended (to provide constant influx of new code)

- **Computational overhead of library search**
  - Can be tackled with appropriate algorithms (nearest-neighbor search, e.g., kd-trees)

<table>
<thead>
<tr>
<th></th>
<th>Static library</th>
<th>Population-based library</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time of build</strong></td>
<td>Once, before run</td>
<td>Every generation</td>
</tr>
<tr>
<td><strong>No. of unique procedures</strong></td>
<td>Constant</td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Semantic diversity</strong></td>
<td>Guaranteed</td>
<td>May converge to local optimum</td>
</tr>
<tr>
<td><strong>Can produce new semantics</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
SBP: Remarks and Extensions

• Requirements of SBP-based operators
  – AGX requires a means of *constructing* a midpoint on a segment.
    • Possible in vector spaces, but in general not in metric spaces
  – RDO can work with any metric (vector space not required)

• The node/subtree $p$ to be replaced can be selected deterministically:
  – E.g., the node where the divergence of the actual semantics $s(p)$ and the desired semantics $D$ is the greatest (Wieloch 2012)
IV. Geometric Semantic GP (GSGP)
Geometric Semantic Operators Construction

• By approximation:
  – Trial & Error is wasteful
  – Offspring do not conform exactly to the semantic requirement

• By direct construction: Is it possible to find search operators that operate on syntax but that are guaranteed to respect geometric semantic criteria by direct construction?

• Due to the complexity of genotype-phenotype map in GP (Krawiec & Lichocki 2009) hypothesized that designing a crossover operator with such a guarantee is in general impossible. A pessimist? No, the established view until then...
Geometric Semantic Crossover for Boolean Expressions

T3 = OR
  /   \ AND  AND
 /     \ /     \ /     \
T1      TR  NOT  T2
     / \   /   /
    TR  TR TR

T1, T2: parent trees
TR: random tree
Theorem

The output vector of the offspring T3 is in the Hamming segment between the output vectors of its parent trees T1 and T2 for any tree TR.
Example: parity problem

• 3-parity problem: we want to find a function $P(X_1, X_2, X_3)$ that returns 1 when an odd number of input variables is 1, 0 otherwise.

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ O = \begin{array}{cccccc}
0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1
\end{array} \]

Error $= HD(Y, O) = 5$
Example: tree crossover

\[
T_1 = \begin{array}{c}
\text{AND} \\
/ \\
X_1 & X_2
\end{array}
\]

\[
T_{R} = \begin{array}{c}
\text{NOT} \\
/ \\
X_3
\end{array}
\]

\[
T_2 = \begin{array}{c}
\text{OR} \\
/ \\
X_2 & X_3
\end{array}
\]

\[
T_3 = \begin{array}{c}
\text{AND} \\
/ \\
T_1 & T_R
\end{array} \quad \begin{array}{c}
\text{AND} \\
/ \\
T_{R} & T_2
\end{array}
\]

substitution & simplification

\[
\begin{array}{c}
\text{OR} \\
/ \\
T_1 & T_3
\end{array} \quad \begin{array}{c}
\text{AND} \\
/ \\
T_3 & \text{NOT}
\end{array}
\]

\[
\begin{array}{c}
/ \\
X_1 & X_2 & X_3
\end{array}
\]
Example: output vector crossover

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
<th>T1</th>
<th>T2</th>
<th>TR</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

• The output vector of TR acts as a crossover mask to recombine the output vectors of T1 and T2 to produce the output vector T3.

• This is a geometric crossover on the semantic distance: output vector of T3 is in the Hamming segment between the output vectors of T1 and T2.
Geometric Semantic Crossover for Arithmetic Expressions

Function co-domain: real
Output vectors: real vectors

Semantic distance = Manhattan
CR = random function with co-domain [0,1]

Semantic distance = Euclidean
CR = random real in [0,1]
Geometric Semantic Crossover for Classifiers

Function co-domain: symbol
Output vectors: symbol string

\[
T_3 = \begin{array}{c}
\text{IF\_THEN\_ELSE} \\
/ \quad | \quad \backslash \\
\text{RC} \quad T_1 \quad T_2
\end{array}
\]

Semantic distance = Hamming
RC = random function with boolean co-domain
(i.e., random condition function of the inputs)
Remark 1: Domain-Specific

• Unlike traditional syntactic operators which are of general applicability, semantic operators are domain-specific.

• But there is a systematic way to derive them for any domain.
Remark 2: Quick Growth

- Offspring grows in size very quickly, as the size of the offspring is larger than the sum of the sizes of its parents!
- To keep the size manageable we need to simplify the offspring without changing the computed function:
  - Boolean expressions: Boolean simplification
  - Math Formulas: algebraic simplification
  - Programs: simplification by formal methods
Remark 3: Syntax Does Not Matter!

• The offspring is defined purely functionally, independently from how the parent functions and itself are actually represented (e.g., trees)
• The genotype representation does not matter: solution can be represented using any genotype structure (trees, graphs, sequences)/language (Java, Lisp, Prolog) as long as the semantic operators can be described in that language
Semantic Mutations

• It is possible to derive geometric semantic mutation operators.

• They also have very simple forms for Boolean, Arithmetic and Program domains.
EXPERIMENTS
Boolean Problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Hits %</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP avg</td>
<td>GP std</td>
</tr>
<tr>
<td>Comparator6</td>
<td>80.2</td>
<td>3.8</td>
</tr>
<tr>
<td>Comparator8</td>
<td>80.3</td>
<td>2.8</td>
</tr>
<tr>
<td>Comparator10</td>
<td>82.3</td>
<td>4.3</td>
</tr>
<tr>
<td>Multiplexer6</td>
<td>70.8</td>
<td>3.3</td>
</tr>
<tr>
<td>Multiplexer11</td>
<td>76.4</td>
<td>7.9</td>
</tr>
<tr>
<td>Parity5</td>
<td>52.9</td>
<td>2.4</td>
</tr>
<tr>
<td>Parity6</td>
<td>50.5</td>
<td>0.7</td>
</tr>
<tr>
<td>Parity7</td>
<td>50.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Parity8</td>
<td>50.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Parity9</td>
<td>50.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Parity10</td>
<td>50.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Random5</td>
<td>82.2</td>
<td>6.6</td>
</tr>
<tr>
<td>Random6</td>
<td>83.6</td>
<td>6.6</td>
</tr>
<tr>
<td>Random7</td>
<td>85.1</td>
<td>5.3</td>
</tr>
<tr>
<td>Random8</td>
<td>89.6</td>
<td>5.3</td>
</tr>
<tr>
<td>Random9</td>
<td>93.1</td>
<td>3.7</td>
</tr>
<tr>
<td>Random10</td>
<td>95.3</td>
<td>2.3</td>
</tr>
<tr>
<td>Random11</td>
<td>96.6</td>
<td>1.6</td>
</tr>
<tr>
<td>True5</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>True6</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>True7</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>True8</td>
<td>100.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
### Polynomial Regression Problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>GP</th>
<th></th>
<th>SSHC</th>
<th></th>
<th>SGP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg</td>
<td>sd</td>
<td>avg</td>
<td>sd</td>
<td>avg</td>
<td>sd</td>
</tr>
<tr>
<td>Polynomial3</td>
<td>79.9</td>
<td>23.1</td>
<td>100.0</td>
<td>0.0</td>
<td>99.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Polynomial4</td>
<td>60.5</td>
<td>27.6</td>
<td>99.9</td>
<td>0.9</td>
<td>99.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Polynomial5</td>
<td>40.7</td>
<td>21.6</td>
<td>100.0</td>
<td>0.0</td>
<td>99.5</td>
<td>2.0</td>
</tr>
<tr>
<td>Polynomial6</td>
<td>37.5</td>
<td>23.4</td>
<td>100.0</td>
<td>0.0</td>
<td>98.9</td>
<td>3.1</td>
</tr>
<tr>
<td>Polynomial7</td>
<td>30.7</td>
<td>18.5</td>
<td>100.0</td>
<td>0.0</td>
<td>99.9</td>
<td>0.9</td>
</tr>
<tr>
<td>Polynomial8</td>
<td>34.7</td>
<td>16.0</td>
<td>99.5</td>
<td>2.0</td>
<td>99.7</td>
<td>1.3</td>
</tr>
<tr>
<td>Polynomial9</td>
<td>20.7</td>
<td>13.2</td>
<td>100.0</td>
<td>0.0</td>
<td>98.5</td>
<td>4.9</td>
</tr>
<tr>
<td>Polynomial10</td>
<td>25.7</td>
<td>16.7</td>
<td>99.4</td>
<td>1.7</td>
<td>99.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Classification Problems

| Problem | Hits % | | | | | | Length |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | GP | avg | sd | GPt | avg | sd | SSHC | avg | sd | SGP | avg | sd | Length |
| n_v | n_c | n_cl | | | | | | | | | | | |
| 3 | 3 | 2 | 80.00 | 8.41 | 97.30 | 4.78 | 99.74 | 0.93 | 99.89 | 0.67 | 1.6 | 1.9 | 2.3 | 2.3 |
| 3 | 3 | 4 | 49.15 | 9.96 | 78.89 | 8.93 | 99.89 | 0.67 | 99.00 | 1.63 | 1.6 | 2.1 | 2.3 | 2.3 |
| 3 | 3 | 8 | 37.04 | 5.07 | 59.52 | 14.26 | 99.74 | 0.93 | 96.04 | 2.85 | 1.2 | 1.9 | 2.3 | 2.3 |
| 3 | 4 | 2 | 67.92 | 7.05 | 93.80 | 5.41 | 99.95 | 0.28 | 99.58 | 0.80 | 1.8 | 2.3 | 2.7 | 2.7 |
| 3 | 4 | 4 | 39.11 | 7.02 | 68.48 | 8.66 | 99.84 | 0.47 | 98.08 | 1.64 | 1.7 | 2.3 | 2.7 | 2.7 |
| 3 | 4 | 8 | 28.02 | 3.73 | 46.98 | 14.48 | 99.73 | 0.58 | 94.22 | 1.72 | 1.1 | 2.0 | 2.7 | 2.7 |
| 4 | 3 | 2 | 88.31 | 6.98 | 98.89 | 2.89 | 99.96 | 0.22 | 100.00 | 0.00 | 1.6 | 1.9 | 2.9 | 2.9 |
| 4 | 3 | 4 | 48.85 | 6.54 | 88.15 | 10.10 | 100.00 | 0.00 | 99.54 | 0.68 | 1.4 | 2.2 | 2.9 | 2.9 |
| 4 | 3 | 8 | 36.54 | 9.01 | 60.37 | 17.14 | 100.00 | 0.00 | 96.63 | 1.23 | 1.0 | 1.9 | 2.9 | 2.9 |
| 4 | 4 | 2 | 82.75 | 8.21 | 99.79 | 1.12 | 100.00 | 0.00 | 99.86 | 0.23 | 2.2 | 2.3 | 3.3 | 3.3 |
| 4 | 4 | 4 | 44.13 | 8.75 | 77.55 | 6.30 | 100.00 | 0.00 | 99.68 | 0.29 | 2.0 | 2.4 | 3.3 | 3.3 |
| 4 | 4 | 8 | 30.63 | 5.33 | 50.21 | 15.08 | 99.96 | 0.12 | 98.84 | 0.58 | 1.4 | 2.1 | 3.3 | 3.3 |
DEALING WITH GROWTH
Geometric Semantic Crossover for Boolean Expressions (Growth)

T1, T2: parent trees
TR: random tree

\[
T_3 = \begin{array}{c}
\text{OR} \\
/ \ \\
\text{AND} \\
/ \\
T_1 \\
/ \\
T_2 \\
/ \\
\text{TR} \\
\text{TR}
\end{array}
\]

size(T3) = 4 + 2 * size(TR) + size(T1) + size(T2)
average size at generation n + 1 > 2 * average size at generation n

PROBLEM: size grows exponentially in the number of generation!
Geometric Semantic Mutation for Boolean Expressions (Growth)

\[ \text{size(TM)} = 2 + \text{size(M)} + \text{size(T)} \]

average size at generation \( n + 1 \) = constant + average size at generation \( n \)

**NO PROBLEM:** size grows linearly in the number of generation
Three Solutions

1. Algebraic simplification of offspring
   - Can be computationally expensive
   - Not all domains can be simplified algebraically
   - Understandable final solutions

2. Not using crossover
   - Semantic Hill-Climber finds optimum efficiently
   - Linear growth is acceptable

3. Compactification of offspring (Vanneschi et al, 2013)
   - Linear growth even with crossover
   - Applicable to any domain
   - Complicated Implementation (pointers structure)
   - Final solution is black box
Compactification Method
(Vanneschi et al, 2013)

- Individuals are represented as **explicit shared linked data structure** to their parents, and recursively to all their ancestry.

- At each generation, each new offspring of crossover requires only a new triplet of references ➞ Linear growth in the number of generations.
Compactification Method

- Output vector of offspring can be computed using the explicitly stored output vectors of the parent and mask trees. This turns fitness computation from exponential in the number of generations to constant time.

<table>
<thead>
<tr>
<th>Id</th>
<th>Individual</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x1 + x2x3</td>
<td>7.34</td>
<td>8.62</td>
<td>9.51</td>
<td>4.07</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>x3 - x2x4</td>
<td>9.73</td>
<td>4.29</td>
<td>5.26</td>
<td>1.45</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>x3 + x4 - 2x1</td>
<td>2.92</td>
<td>3.76</td>
<td>5.23</td>
<td>6.24</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>x1x3</td>
<td>7.28</td>
<td>1.78</td>
<td>3.26</td>
<td>5.74</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>x1 - x3</td>
<td>2.57</td>
<td>4.67</td>
<td>3.22</td>
<td>6.91</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Individual</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x1 + x2 - 2x4</td>
<td>2.64</td>
<td>3.28</td>
<td>5.93</td>
<td>4.29</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>x2 - x1</td>
<td>6.94</td>
<td>7.53</td>
<td>8.53</td>
<td>2.65</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>x1 + x4 - 3x3</td>
<td>4.84</td>
<td>3.56</td>
<td>2.76</td>
<td>9.76</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>x2 - x3 - x4</td>
<td>4.37</td>
<td>5.94</td>
<td>2.59</td>
<td>1.85</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>2x1</td>
<td>4.07</td>
<td>3.27</td>
<td>2.57</td>
<td>7.47</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Operator</th>
<th>Entry</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
<th>T10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>crossover</td>
<td>(&amp;(T1), &amp;(T4), &amp;(R1))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>crossover</td>
<td>(&amp;(T4), &amp;(T5), &amp;(R2))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>crossover</td>
<td>(&amp;(T3), &amp;(T5), &amp;(R3))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>crossover</td>
<td>(&amp;(T1), &amp;(T5), &amp;(R4))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>crossover</td>
<td>(&amp;(T3), &amp;(T4), &amp;(R5))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compactification Method

- Explicit garbage collection of unreferenced past individuals in the data structure.

- Final solution is extracted from data structure but this takes exponentially long in the number of generation.

- Extracted solution is queried on non-training inputs to make predictions. This takes exponential time since done on extracted solution.

Good idea, but can be improved and beautified!
Functional Compactification (Moraglio, 2014)

• Individuals are represented directly as anonymous Python functions:

\[
P1 = \text{lambda } x1, x2, x3: x1 \text{ or (x2 and not x3)}
\]
\[
P2 = \text{lambda } x1, x2, x3: x1 \text{ and x2}
\]
\[
RF = \text{lambda } x1, x2, x3: \text{not (x2 and x3)}
\]
Functional Compactification

• Offspring **call** parents rather than pointing to them:

\[
OX = \text{lambda } x1, x2, x3:\n((P1() \text{ and } RF()) \text{ or } (P2() \text{ and } \neg RF()))
\]

• The size of offspring is **constant** in the number of generations
Functional Compactification

• Mutation and Crossover are higher order functions that take functions in inputs (parents) and return functions as output (offspring):

  Crossover: \((B^3 \rightarrow B) \times (B^3 \rightarrow B) \rightarrow (B^3 \rightarrow B)\)

• The function calls structure keeps implicitly trace of all ancestry of an individual
Functional Compactification

• All individuals are *momoized functions*:
  - The output of previously seen inputs is retrieved from an implicit storage, not recalculated
  - The first time the fitness of an individual is calculated, its output vector is implicitly stored
  - As the output vectors of parents are stored, the fitness of the offspring takes constant time in num generations
Functional Compactification

- **Garbage collection** of unreferenced past functions done automatically by the Python compiler.

- **Final solution is a Python compiled function** (but can be extracted by keeping track of its source code). The extracted solution would be exponentially long.

- The compiled final solution can be queried on non-training inputs to make predictions. Thanks to the memoization obtaining the output takes only **linear time**.
Functional Compactification

• The functional interpretation of the compactification method delegates implicitly all book-keeping of the original compactification method to the Python compiler.

• The resulting code is elegant, much shorter and clear as it has only minimal clutter (< 100 lines including extensive comments vs original compactification > 2000 lines of C++).
GSGP Implementations

• Original Mathematica implementation with algebraic simplification (see https://github.com/amoraglio/GSGP)

• Compactification method in C++ (see http://gsgp.sourceforge.net/)

• Functional compactification aka Tiny GSGP in Python (see https://github.com/amoraglio/GSGP)

• Scala implementation using the ScaPS library (see http://www.cs.put.poznan.pl/kkrawiec/wiki/?n=Site.Scaps)
RUNTIME ANALYSIS OF MUTATION-BASED GSGP
Runtime Analysis

• Rigorous analytical formula of the expected optimisation time of the search algorithm A on the problem class P (on the worst instance) for increasing size n of the problem
Runtime Analysis (example)

- Algorithm: stochastic hill-climber i.e., flip a bit of the current solution and accept new solution if it is better than current

- Problem class: one-max i.e., sum of ones in the bit string to maximise; the problem size is the string size

- Expected optimisation time: $O(n \log n)$ by coupon collector argument

- This result generalises to onemax with an unknown target string, i.e., to any cone landscape on binary strings
Semantic Mutation
(syntactic search & semantic effect)
Search Equivalence

Semantic GP search at a syntax level on any problem

Traditional GA search on output vectors on onemax

The search outputs a tree (i.e., a function), but the runtime analysis can be done on the GA!
**Forcing Point Mutation (not Bit Flip)**

**Definition 3.** Forcing point mutation: Given a parent function \( \mathcal{X} : \{0, 1\}^n \rightarrow \{0, 1\} \), the mutation returns the offspring boolean function \( \mathcal{X}' = \mathcal{X} \lor M \) with probability 0.5, and \( \mathcal{X}' = \mathcal{X} \land \overline{M} \) with probability 0.5, where \( M \) is a random minterm of all input variables.

\[
\begin{align*}
X &= ((X_1 \land X_2) \lor \lnot X_3) \lor X_3 \\
M &= \lnot X_1 \land X_2 \land \lnot X_3 \\
X' &= X \lor M
\end{align*}
\]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>X2</td>
<td>X3</td>
<td>Output</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 \rightarrow 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[X = ((X_1 \land X_2) \lor \lnot X_3) \lor X_3\]
\[M = \lnot X_1 \land X_2 \land \lnot X_3\]
\[X' = X \lor M\]
Issue 1: Exponential Chromosome Size

- Problem size $n$: number of input variables

- Output vector size $N$: $2^n$ (exponentially long in the number of variables!)

- $(1+1)$-EA on OneMax has runtime $N \log N = n \ 2^n$ (exponential!)
Issue 2: Exponential Amount of Neutrality

• Training set size $t$: must be polynomial in $n$ for the fitness to be computable in poly time
• The output vectors of size $2^n$ have only poly$(n)$ active bits, all other bits are inactive: sparse OneMax with very rare active bits
• Black-box model: we do not know which bits are active and which are inactive
• $(1+1)$-EA takes exponential time to optimise sparse OneMax
Solution: Block Mutation

• Use incomplete minterm as a basis for forcing mutation. This has the effect of forcing at once blocks of entries to the same random value.

X = ((X1 ^ X2) ^ !X3) v X3
M = !X1
X' = X v M
Fixed Block Mutation

**Definition 6.** Fixed Block Mutation (FBM): Let us consider a fixed set of \( v < n \) variables (fixed in some arbitrary way at the initialisation of the algorithm). FBM draws uniformly at random an incomplete minterm \( M \) comprising all fixed variables as a base for the forcing mutation.

Fix Variables = \{X1,X2\}
Possible M = \{!X1 ^ !X2, !X1 ^ X2, X1 ^ !X2, X1 ^ X2\}

\[
X = ((X1 ^ X2) ^ !X3) \lor X3
\]

\[
M = !X1 ^ X2
\]

\[
X' = X ^ !M
\]

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 \to 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1 \to 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Polynomial Runtime with High Probability of Success on All Boolean Problems!

**Theorem 4.** Let us assume that the size of the training set $\tau$ is a polynomial $n^c$ in the number of input variables $n$, with $c$ a positive constant. Let us choose the number of fixed variables $v$ logarithmic in $n$ such that $v > 2c \log_2(n)$. Then, semantic GP with FBM finds a function satisfying the training set in polynomial time with high probability of success, on any problem $P$, and training set $T$ uniformly sampled from $P$.

**Proof idea:** choose $v$ such that the number of partitions of the output vector is polynomial in $n$ (so that the runtime is polynomial), and larger enough than the training set, so that each training example is in a single block w.h.p. (which guarantees that the optimum can be reached).
Lesson from Theory

- Rigorous runtime analysis of GSGP on general classes of non-toy problems is possible as the landscape is always a cone.
- There are issues with GSGP which require careful design of semantic mutations to obtain efficient search. Theory can guide the design of provably good semantic operators in terms of runtime.
- Runtime analysis of GSGP with several other mutation operators for Boolean, arithmetic and classification domains have been done producing refined provably good semantic search operators.
V. Other developments & current research directions
SGP and Neutrality

• Similarly to non-semantic operators, SGP operators can be ineffective (in the semantic sense).
  – The offspring is a semantic clone of a parent.
  – Slows down the search process.

• Percentage of neutral mutations:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbolic regression</th>
<th>Boolean function synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGX (Moraglio et al.)</td>
<td>0.679</td>
<td>0.719</td>
</tr>
<tr>
<td>AGX (Pawlak et al.)</td>
<td>0.131</td>
<td>0.935</td>
</tr>
<tr>
<td>LGX (Krawiec et al.)</td>
<td>0.067</td>
<td>0.724</td>
</tr>
<tr>
<td>KLX (Krawiec et al.)</td>
<td>0.866</td>
<td>0.895</td>
</tr>
<tr>
<td>SAC (Uy et al.)</td>
<td>0.067</td>
<td>0.649</td>
</tr>
<tr>
<td>GPX (Koza et al.)</td>
<td>0.103</td>
<td>0.518</td>
</tr>
</tbody>
</table>

• Can be tackled by testing potential offspring for semantic neutrality.
GP as a Test-Based Problem

- **Test based problem** \((S, T, G, Q)\) (Popovici et al. 2012):
  - \(S\) – set of candidate solutions (in GP: programs)
  - \(T\) – set of tests (in GP: tests, fitness cases)
  - \(G\) – interaction matrix
  - \(Q\) – quality measure

- **Examples:** Games (strategies vs. opponents), control problems (controllers vs. initial conditions), machine learning from examples (hypotheses vs. examples)
  - Generally: co-optimization and co-search

\[
g : S \times T \rightarrow [0, 1]
\]

<table>
<thead>
<tr>
<th></th>
<th>(t_1)</th>
<th>(t_2)</th>
<th>(t_3)</th>
<th>(t_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(b)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(c)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(d)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Discovery of Underlying Objectives via Clustering

(Krawiec & Liskowski 2013)

Create interaction matrix $G$

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Clustering algorithm

Derived objectives $G'$

<table>
<thead>
<tr>
<th></th>
<th>$t_{1+3}$</th>
<th>$t_{2+4+5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0.66</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>0.33</td>
</tr>
</tbody>
</table>

$G'$ after clustering
Behavioral GP

- Generalizes program behavior to the **entire course of program execution**, not only program output
- Program behavior = list of execution traces

(Krawiec & Swan 2013, Krawiec & O'Reilly 2014)
Behavioral GP: Example

Problem

GP Individual

ML dataset

Decision tree

Evaluation:

- Program error: 10
- Classifier error: 0 examples
- Classifier complexity: 5 nodes
Recent Developments

• New approaches based on semantic back propagation (Ffrancon & Schoenauer, 2015)
• Lexicase selection (Helmuth et al. 2012)
• Relationship to novelty search (program semantics = behavioral descriptor)
Other Lines of Investigation in GSGP

• Application to other types of GP
  – Geometric Semantic Grammatical Evolution

• Many Real-World Applications (Vanneschi et al, 2013)

• Generalisation Studies
  – PAC learning for provably good generalisation of GSGP

• Derivation of semantic operators for more complex domain (e.g., recursive programs) on more complex data structures (e.g., lists)
Thank you!

Questions?

Credits: The authors thank Bartosz Wieloch and Tomasz Pawlak for their feedback on the slides of the tutorial. Other credits: Wikipedia
References

• K. Krawiec, T. Pawlak, Locally Geometric Semantic Crossover: A Study on the Roles of Semantic and Homology in Recombination Operators, Genetic Programming and Evolvable Machines, 2013,
• L. Beadle, C. Johnson, Semantically Driven Crossover in Genetic Programming, CEC 2008,
• L. Beadle, C. Johnson, Semantically Driven Mutation in Genetic Programming, CEC 2009,
• N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, E. Galvan-Lopez, Semantically-based crossover in genetic programming: application to real-valued symbolic regression, Genetic Programming and Evolvable Machines, 2011,
• Semantic selection:
  • Lasarczyk, C. W. G. & and Wolfgang Banzhaf, P. D. Dynamic Subset Selection Based on a Fitness Case Topology Evolutionary Computation, 2004, 12, 223-242
• Langdon, W. B. & Poli, R. Foundations of Genetic Programming Springer-Verlag, 2002
• A. Moraglio, A. Mambrini, Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming for Basis Functions Regression, Genetic and Evolutionary Computation Conference, 2013.
• L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics, EuroGP 2013
• L. Vanneschi, S. Silva, M. Castelli, L. Manzoni, Geometric semantic genetic programming for real life applications, in Genetic Programming Theory and Practice XI, 2013