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

Advanced Tutorial @ GECCO'16
http://www.sigevo.org/gecco-2016/

GECCO'16 Companion, July 20-24, 2016, Denver, CO, USA
ACM 978-1-4503-4323-7/16/07.
http://dx.doi.org/10.1145/2908961.2926990

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 Sematic Genetic Programming
5. Other Developments and Current Research Directions

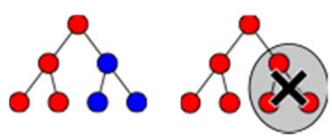
I. Introduction to Semantic Genetic Programming

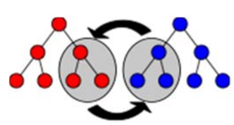
Genetic Programming

- 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 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 similarity of offspring to parents
  - A slight program modification can dramatically change its output
  - And conversely: high likelihood of no-effect (neutrality)
  - Low fitness-distance correlation





Questions

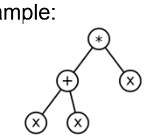
- Can we make GP more aware about the *effects* of program execution, i.e., program ‘behavior’?
- Can we design search operators that produce offspring program which behave similarly to parent(s)?
- Can we design search operators that are *guaranteed* to do so?

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:



x	result
-0.5	**0.5**
1.0	**2.0**
1.5	**4.5**
2.0	**8.0**

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

- 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

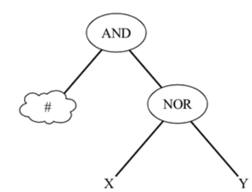
(McPhee, Ohs, Hutchison 2007/2008)

- 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.



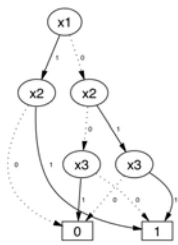
Parent semantics	Arg semantics (x)	(and x #)	(or x #)	(nand x #)	(nor x #)
0	0	0	0	0	0
0	1	0	0	0	0
1	0	1	1	1	1
1	1	1	1	1	1
+	0	0	+	1	-
+	1	+	1	-	0
-	0	1	-	0	+
-	1	-	0	+	1

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
- Repeat until *P*'s capacity:
  - 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*

Semantic-Aware Initialization

- Behavioral Initialization (Jackson 2010)
- Start: set $P \leftarrow \varnothing$
- Repeat until *P*'s capacity:
  - 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*
- 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

$$d(x,y) \geq 0$$

$$d(x,y) = 0 \Leftrightarrow x = y$$

$$d(x,y) = d(y,x)$$

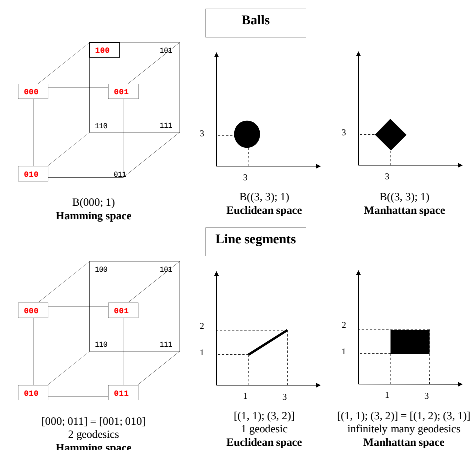
$$d(x,z) + d(z,y) \geq d(x,y)$$

Balls & Segments

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

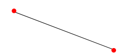
$$[x;y] = \{ z \in S \mid d(x,z) + d(z,y) = d(x,y) \}$$

Squared Balls & **Chunky Segments**

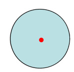


Geometric Crossover & Mutation

- **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.

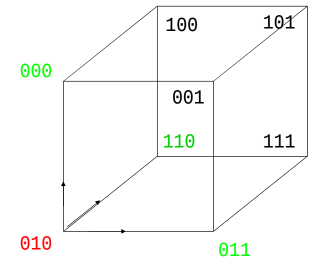


- **Geometric mutation**: a mutation operator is a r-geometric mutation under the metric d if all its offspring are in the d-ball of radius r centred in the parent.



Example of Geometric Mutation

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

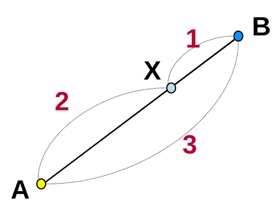


Neighbourhood structure naturally associated with the shortest path distance.

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.

A	0	1	1	0	1
B	1	1	0	1	1
X	0	1	0	1	1



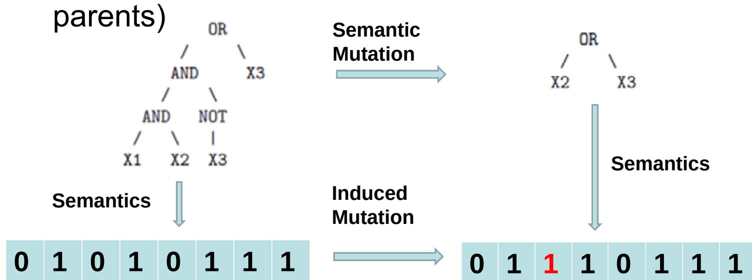
H(A,X) + H(X,B) = H(A,B)

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

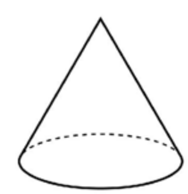
- Aim: we want to find a function that scores perfectly on a given set of input-output examples (test cases)
- Error of a program: number of mismatches on the test cases
- Fitness as distance: the error of a program can be interpreted as the distance of the output vector of the program to the target 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)

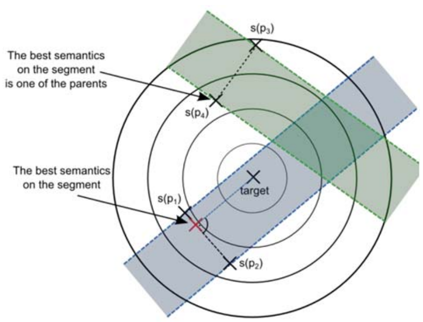
- Goal: Minimize offspring's total semantic distance from the parents under some assumed metric || ||.
- Technical realization: Mate the parents (x,y) repetitively using a 'regular' crossover operator CX
- Calculate parent semantics $s(p_1)$, $s(p_2)$
- Repeat:
  - Apply CX to (p_1,p_2) n times, creating 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*:

$$\text{argmin } ||s(z) - s(p_1)|| + ||s(z) - s(p_2)||$$

- A form of *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

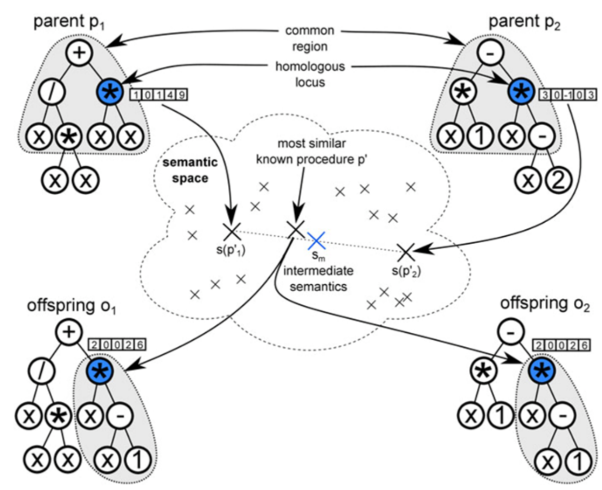
- All candidate offspring on the segment [s(p_1);s(p_2)] minimize total distance equally well, no matter how different from the parents they are.
  - An offspring z that is a 'semantic clone' of p_1 ($s(z) = s(p_1)$) also minimises the total distance.
  - The likelihood of crossover producing a semantic 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.



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

```
              OR
            /    \
         AND       AND
T3 =    /   \     /    \
      T1     TR  NOT    T2
                  |
                  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(X1,X2,X3) that returns 1 when an odd number of input variables is 1, 0 otherwise.

```
X1 X2 X3 | Y
0  0  0  | 0
0  0  1  | 1
0  1  0  | 1
0  1  1  | 0
1  0  0  | 1
1  0  1  | 0
1  1  0  | 0
1  1  1  | 1
```

```
            OR
           /   \
        AND     X3
       /   \
    AND    NOT
   /   \    |
 X1    X2   X3
```

Error = HD(Y,O) = 5

O= 0 1 0 1 0 1 1 1

Example: tree crossover

```
T1 =    AND
       /   \
     X1     X2

TR =    NOT
         |
         X3

T2 =    OR
       /  \
     X2    X3
```

```
            OR                                        OR
           /  \                                      /  \
T3 =    AND    AND        substitution &          AND    X3
       /  \    /  \       simplification         /   \
      T1  TR NOT  T2      ========>            AND    NOT
              |                               /   \    |
              TR                             X1   X2   X3
```

Example: output vector crossover

X1	X2	X3	Y	T1	T2	TR	T3
0	0	0	0	0	0	1	0
0	0	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	1
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	0	1

- 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

```
              +
             / \
            *   *
T3 =       / \ / \
          T1 CR -  T2
               / \
              1   CR
```

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

```
       IF_THEN_ELSE
T3 =   /    |    \
      RC    T1    T2
```

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

Problem	Hits %								Length			
	GP		GPt		SSHC		SGP					
	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
Comparator6	80.2	3.8	90.9	3.5	99.8	0.5	99.5	0.7	1.0	2.0	2.9	2.8
Comparator8	80.3	2.8	94.9	2.4	100.0	0.0	99.9	0.2	1.0	2.3	2.9	3.0
Comparator10	82.3	4.3	95.3	0.9	100.0	0.0	100.0	0.1	1.6	2.4	2.7	3.0
Multiplexer6	70.8	3.3	94.7	5.8	99.8	0.5	99.5	0.8	1.1	2.2	2.7	2.9
Multiplexer11	76.4	7.9	88.8	3.4	100.0	0.0	99.9	0.1	2.2	2.4	2.9	2.6
Parity5	52.9	2.4	56.3	4.9	99.7	0.9	98.1	2.1	1.4	1.7	2.9	2.9
Parity6	50.5	0.7	55.4	5.1	99.7	0.6	98.8	1.7	1.0	1.9	3.0	3.0
Parity7	50.1	0.2	51.7	2.8	99.9	0.2	99.5	0.6	1.0	1.7	3.0	3.1
Parity8	50.1	0.2	50.6	0.9	100.0	0.0	99.7	0.3	1.0	1.6	3.4	3.4
Parity9	50.0	0.0	50.2	0.1	100.0	0.0	99.5	0.3	1.0	1.3	3.8	3.8
Parity10	50.0	0.0	50.0	0.0	100.0	0.0	99.4	0.2	0.9	1.2	4.1	4.1
Random5	82.2	6.6	90.9	6.0	99.5	1.2	98.8	2.1	0.9	1.6	2.7	2.8
Random6	83.6	6.6	93.0	4.1	99.9	0.4	99.2	1.3	1.2	1.9	2.9	2.8
Random7	85.1	5.3	92.9	3.8	99.9	0.2	99.8	0.4	1.1	2.0	2.8	2.9
Random8	89.6	5.3	93.7	2.4	100.0	0.1	99.9	0.2	1.4	2.0	3.0	2.9
Random9	93.1	3.7	95.4	2.3	100.0	0.1	100.0	0.1	1.5	1.8	2.9	2.9
Random10	95.3	2.3	96.2	2.0	100.0	0.0	100.0	0.0	1.5	1.8	2.8	3.0
Random11	96.6	1.6	97.3	1.5	100.0	0.0	100.0	0.0	1.6	1.7	2.7	3.1
True5	100.0	0.0	100.0	0.0	99.9	0.6	100.0	0.0	1.1	1.3	2.0	2.4
True6	100.0	0.0	100.0	0.0	99.8	0.6	100.0	0.0	1.2	1.2	2.6	2.5
True7	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0	1.2	1.2	2.9	2.6
True8	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.1	1.2	1.4	3.3	2.9

Polynomial Regression Problems

Problem	Hits %					
	GP		SSHC		SGP	
	avg	sd	avg	sd	avg	sd
Polynomial3	79.9	23.1	100.0	0.0	99.5	1.5
Polynomial4	60.5	27.6	99.9	0.9	99.9	0.9
Polynomial5	40.7	21.6	100.0	0.0	99.5	2.0
Polynomial6	37.5	23.4	100.0	0.0	98.9	3.1
Polynomial7	30.7	18.5	100.0	0.0	99.9	0.9
Polynomial8	34.7	16.0	99.5	2.0	99.7	1.3
Polynomial9	20.7	13.2	100.0	0.0	98.5	4.9
Polynomial10	25.7	16.7	99.4	1.7	99.9	0.9

Classification Problems

Problem			Hits %								Length			
			GP		GPt		SSHC		SGP					
n_v	n_c	n_{cl}	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
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)

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

T1, T2: parent trees
TR: random tree

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)

```
                       OR
Prob 0.5 --> TM =     /  \
                     T    M

                       AND
Prob 0.5 --> TM =     /   \
                     T    NOT
                            \
                             M
```

T: parent tree
M: random minterm tree
TM: mutant tree

size(TM) = 2 + size(M) + 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. Compression of offspring (Vanneschi et al, 2013)
   - Linear growth even with crossover
   - Applicable to any domain
   - Complicated Implementation (pointers structure)
   - Final solution is black box

Compression 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.

These are memory references!

Id	Individual
T_1	$x_1 + x_2 x_3$
T_2	$x_3 - x_2 x_4$
T_3	$x_3 + x_4 - 2x_1$
T_4	$x_1 x_3$
T_5	$x_1 - x_3$

Id	Individual
R_1	$x_1 + x_2 - 2x_4$
R_2	$x_2 - x_1$
R_3	$x_1 + x_4 - 3x_3$
R_4	$x_2 - x_3 - x_4$
R_5	$2x_1$

Id	Operator	Entry
T_6	crossover	$\langle \&(T_1), \&(T_4), \&(R_1) \rangle$
T_7	crossover	$\langle \&(T_4), \&(T_5), \&(R_2) \rangle$
T_8	crossover	$\langle \&(T_3), \&(T_5), \&(R_3) \rangle$
T_9	crossover	$\langle \&(T_1), \&(T_5), \&(R_4) \rangle$
T_{10}	crossover	$\langle \&(T_3), \&(T_4), \&(R_5) \rangle$

Compression 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.

Id	Individual				
T_1	$x_1 + x_2x_3$	7.34	8.62	9.51	4.07
T_2	$x_3 - x_2x_4$	9.73	4.29	5.26	1.45
T_3	$x_3 + x_4 - 2x_1$	2.92	3.76	5.23	6.24
T_4	x_1x_3	7.28	1.78	3.26	5.74
T_5	$x_1 - x_3$	2.57	4.67	3.22	6.91

Id	Individual				
R_1	$x_1 + x_2 - 2x_4$	2.64	3.28	5.93	4.29
R_2	$x_2 - x_1$	6.94	7.53	8.53	2.65
R_3	$x_1 + x_4 - 3x_3$	4.84	3.56	2.76	9.76
R_4	$x_2 - x_3 - x_4$	4.37	5.94	2.59	1.85
R_5	$2x_1$	4.67	3.27	2.57	7.47

Id	Operator	Entry				
T_6	crossover	$\langle \&(T_1), \&(T_4), \&(R_1) \rangle$	......	......	......	......
T_7	crossover	$\langle \&(T_4), \&(T_5), \&(R_2) \rangle$	......	......	......	......
T_8	crossover	$\langle \&(T_3), \&(T_5), \&(R_3) \rangle$	......	......	......	......
T_9	crossover	$\langle \&(T_1), \&(T_5), \&(R_4) \rangle$	......	......	......	......
T_{10}	crossover	$\langle \&(T_3), \&(T_4), \&(R_5) \rangle$	......	......	......	......

We also store semantics

Storing also the semantics of each individual allows us to calculate the fitness without evaluating the whole expresson!!

Obtainable directly from here

Semantics in the next population

Compression 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 Compression (Moraglio, 2014)

- Individuals are represented directly as anonymous **Python functions**:

```
P1 = lambda x1, x2, x3: x1 or (x2 and not x3)
P2 = lambda x1, x2, x3: x1 and x2
RF = lambda x1, x2, x3: not (x2 and x3)
```


Functional Compression

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

```
OX = lambda x1, x2, x3:
        ((P1() and RF()) or (P2() and not RF())
```

- The size of offspring is **constant** in the number of generations

- The function calls structure keeps **implicitly trace of all ancestry** of an individual

Functional Compression

- All individuals are **momoized functions**: fitness of the offspring can be computed directly from stored output vectors of parents.
- **Garbage collection** of unreferenced past functions done automatically by the Python compiler.
- **Final solution is a Python compiled function**. The extracted solution is exponentially long.
- **But** the compiled final solution can be queried on non-training inputs to make predictions in **linear time**.

GSGP Implementations

- Original Mathematica implementation with algebraic simplification: https://github.com/amoraglio/GSGP
- Compression method (>2000 lines in C++): http://gsgp.sourceforge.net/
- Functional compression (<100 lines in Python): https://github.com/amoraglio/GSGP
- Scala implementation using the ScaPS library: http://www.cs.put.poznan.pl/kkrawiec/wiki/?n=Site.Scaps

RUNTIME ANALYSIS OF MUTATION-BASED GSGP

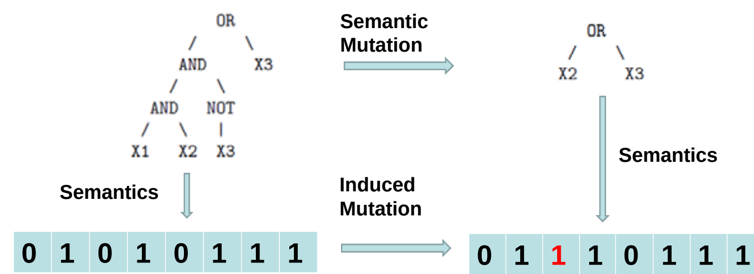
Runtime Analysis

- 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

Semantics

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 $\mathcal{X} : \{0,1\}^n \to \{0,1\}$, the mutation returns the offspring boolean function $\mathcal{X}' = \mathcal{X} \vee M$ with probability 0.5, and $\mathcal{X}' = \mathcal{X} \wedge \overline{M}$ with probability 0.5, where M is a random minterm of all input variables.*

```
         OR
        /   \
      AND    X3
     /   \
   AND   NOT
  /   \    |
X1    X2  X3
```

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

X1	X2	X3	Output
0	0	0	0
0	0	1	1
0	1	0	0 → 1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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.

```
        OR
       /  \
    AND    X3
    /  \
  AND   NOT
  / \    |
X1   X2  X3
```

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

X1	X2	X3	Output
0	0	0	0 → 1
0	0	1	1 → 1
0	1	0	0 → 1
0	1	1	1 → 1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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) v X3
M = !X1 ^ X2
X' = X ^ !M

X1	X2	X3	Output
0	0	0	0
0	0	1	1
0	1	0	0 → 0
0	1	1	1 → 0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Polynomial Runtime with High Probability of Success on All Boolean Problems!

THEOREM 4. *Let us assume that the size of the training set τ 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

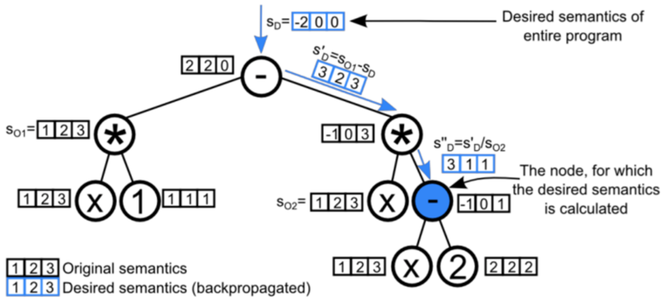
- Rigorous runtime analysis of GSGP on general classes of non-toy problems is possible as the landscape is always a cone
- There are issues with GSGP which require careful design of semantic mutations to obtain efficient search. Theory can guide the design of provably good semantic operators in terms 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

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: $y = x^2 \Rightarrow x = \pm\text{sqrt}(x)$
- The desired output *t* of a program (target) is known.
- Given a program and *t*, this allows deriving *desired semantics* at any point in a program tree.

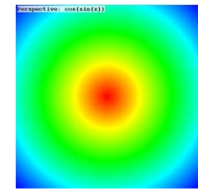
Semantic Backpropagation



SBP can be used to back propagate any semantics.

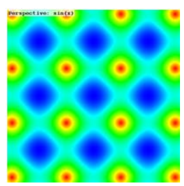
Propagation of Desired Semantics: Example 1

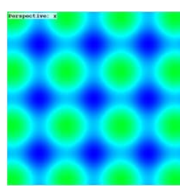
- Two fitness cases, 2D semantic space
- Desired outputs: (0,0)
- Program: cos(sin(*x*))
- Visualization:
  - semantic distance as a function of inputs (x_1, x_2)
  - red = smaller semantic distance (greater fitness)



Propagation of Desired Semantics: Example 2

- Top: desired semantics of cos(#)
  - target achieved for $x_1, x_2 = \pi + k\pi$, $k \in Z$
- Bottom: desired semantics of cos(sin(#))
  - Target cannot be achieved, because sin $\in$ [-1,1], and thus no *x* causes cos(sin(*x*)) = 0





Semantic Backpropagation

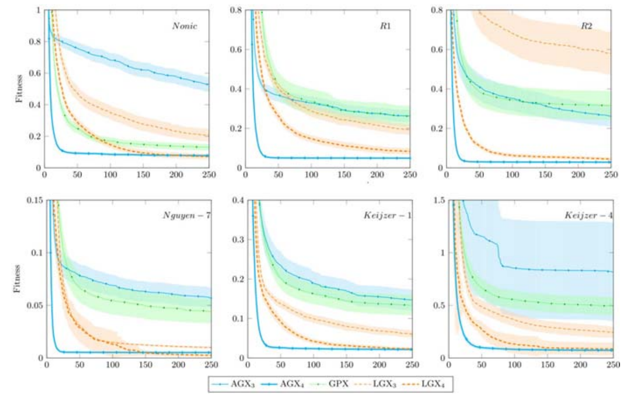
- Desired semantics is a *n*-tuple of *sets of* desired outputs, because not all instructions are bijective (*n* = number of tests). Examples:
  - *D* = ({2}, {3}, {2,-4}, {0, 1})
  - *D* = ({T}, {F}, {T,F})
- Captures exponentially many GP semantics (with respect to *n*).
- Special case: *non-realizable desired semantics*, e.g., *D* = ({T}, ∅, {T,F})
  - Or: non-realizable under assumed constraints (e.g., size of subprogram).
- Algorithms have to account for that.

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*
- 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

(*) Subprogram libraries:
- Static: Generated prior to run
- Dynamic: Other programs in population

AGX: Some Results



(Pawlak, Wieloch, Krawiec, 2014)

SGP and Neutrality

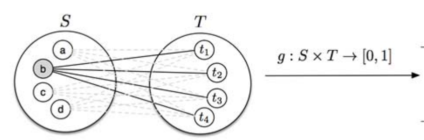
- Similarly to non-semantic operators, SGP operators can be ineffective (in the semantic sense).
  - The offspring is a semantic clone of a parent.
  - Slows down the search process.
- Percentage of neutral mutations:

Operator	Symbolic regression	Boolean function synthesis
SGX (Moraglio et al.)	***0.679***	***0.719***
AGX (Pawlak et al.)	***0.131***	***0.935***
LGX (Krawiec et al.)	***0.067***	***0.724***
KLX (Krawiec et al.)	***0.866***	***0.895***
SAC (Uy et al.)	***0.067***	***0.649***
GPX (Koza et al.)	***0.103***	***0.518***

- Can be tackled by testing potential offspring for semantic neutrality.

GP as a Test-Based Problem

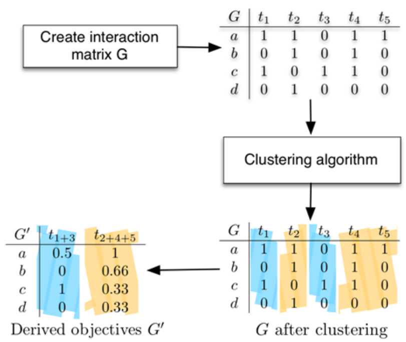
- Test based problem (S, T, G, Q) (Popovici et al. 2012):
  - S – set of candidate solutions (in GP: programs)
  - T – set of tests (in GP: tests, fitness cases)
  - G – interaction matrix
  - Q – quality measure
- Examples: Games (strategies vs. opponents), control problems (controllers vs. initial conditions), machine learning from examples (hypotheses vs. examples)
  - Generally: co-optimization and co-search



G	t_1	t_2	t_3	t_4
a	1	1	0	0
b	0	1	0	1
c	1	1	1	0
d	0	1	1	0

Discovery of Underlying Objectives via Clustering

(Krawiec & Liskowski 2013)



Discovery of Underlying Objectives (and Surrogate Fitness) via Matrix Factorization

(Krawiec & Liskowski 2016)

$$G = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & & 1 & 2 & \\ & 2 & 1 & & 1 \\ 1 & & & 2 & 2 \\ 2 & 1 & & & 1 \end{pmatrix} \end{array}$$

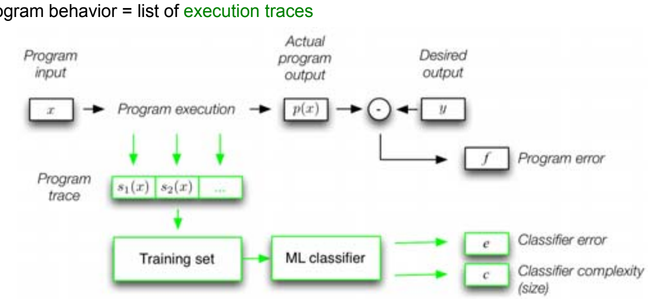
Missing outcomes due to $\alpha < 1$

$$W = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{ccc} f_1 & f_2 & f_3 \end{array} \\ \begin{pmatrix} 0.46 & 1.96 & 0.6 \\ 1.27 & 0.1 & 0.95 \\ 1.37 & 0.02 & 2.83 \\ 0.4 & 1.86 & 1.60 \end{pmatrix} \end{array}, \quad H = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 0.48 & 1.50 & 0.01 & 0.41 & 0.41 \\ 0.87 & 0.14 & 0.19 & 0.77 & 0.01 \\ 0.11 & 0.09 & 1.02 & 0.50 & 0.51 \end{pmatrix} \end{array}$$

$$\hat{G} = WH = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 2 & 1.02 & 1 & 2 & 0.52 \\ 0.8 & 2 & 1 & 1.07 & 1 \\ 1 & 2.31 & 2.1 & 2 & 2 \\ 2 & 1 & 2.01 & 2.4 & 1 \end{pmatrix} \end{array} \xrightarrow{f(p_i) = \sum_{j=1}^{n} g_{ij}} \begin{array}{l} f(p_1) = 6.54 \\ f(p_2) = 5.87 \\ f(p_3) = 9.41 \\ f(p_4) = 8.41 \end{array}$$

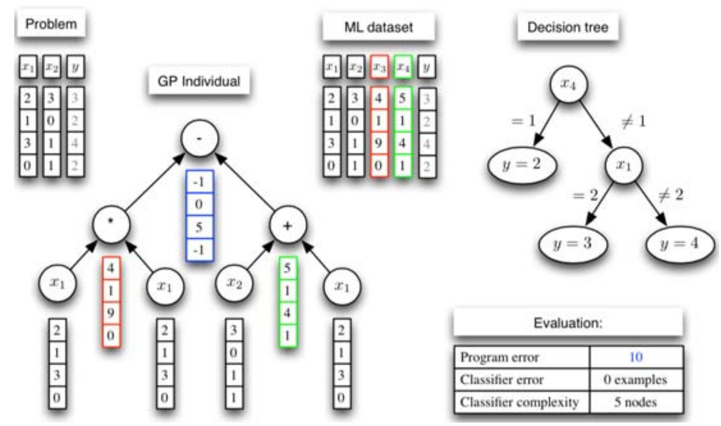
Behavioral GP

- 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



Recent Developments

- New approaches based on semantic back propagation (Ffrancon & Schoenauer, 2015)
- Lexicase selection (Helmuth et al. 2012)
  - Epsilon-lexicase selection (La Cava et al., GECCO'16)
- Relationship to novelty search (program semantics = behavioral descriptor)

Competent Initialization

- Competent Geometric Semantic Genetic Programming (Pawlak & Krawiec 2016)
- Start: $P \leftarrow$ all terminal instructions
- Repeat until *P*'s capacity:
  - Create a candidate program *p* by picking a random nonterminal and appending it with randomly selected programs from *P*
  - Add *p* to *P* if its semantics
    - is sufficiently distant from semantics of all programs in *P, and*
    - expands the convex hull of semantics in *P*
- Observation: Probability of enclosing target semantics in population's convex hull decreases with target distance to the origin of coordinate system

Bounds on offspring's fitness in GSGP
(Pawlak 2015)

- Let:
  - L_F be a Minkowski metric of order *F* used by fitness function
  - L_D be a Minkowski metric or order *D* used by geometric operators (operator's metric)
- *r*-geometric mutation applied to a program *p* produces an offspring *p'* such that
  - $f(p') = f(p) \pm r \cdot n^{1/F - 1/D}$ if $F \leq D$
  - $f(p') = f(p) \pm r$ otherwise
- The fitness of an offspring resulting from geometric crossover relative to the worse parent's fitness is bounded from above by:

		Fitness function		
		L_1	L_2	L_∞
Operator's metric	L_1	n	√n	1
	L_2	1	1	1
	L_∞	1	√2	2

- Practical upshot: use L_2!

Other Lines of Investigation in GSGP

- Application to other types of GP
  - Geometric Sematic 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

- A. Moraglio, K. Krawiec, C. Johnson, Geometric Semantic Genetic Programming, PPSN XII, 2012.
- K. Krawiec, P. Lichocki, Approximating Geometric Crossover in Semantic Space, GECCO 2009,
- 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,
- T. Pawlak, B. Wieloch, K. Krawiec, Semantic Backpropagation for Designing Genetic Operators in Genetic Programming, IEEE Transactions on Evolutionary Computation, 2014.
- 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,
- N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, D.N. Phong, On the roles of semantic locality in genetic programming, Information Sciences, 2013,
- N.Q. Uy, N.X. Hoai, Michael O'Neill, Semantics based mutation in genetic programming: The case for real-valued symbolic regression, MENDEL 2009.
- L. Beadle, C. Johnson, Semantic analysis of program initialisation in genetic programming, Genetic Programming and Evolvable Machines, 2009,
- D. Jackson, Promoting Phenotypic Diversity in Genetic Programming, PPSN XI, 2010.
- Semantic selection:
- E. Galvan-Lopez, B. Cody-Kenny, L. Trujillo, A. Kattan, Using Semantics in the Selection Mechanism in Genetic Programming: a Simple Method for Promoting Semantic Diversity, CEC 2013.
- R.E. Smith, S. Forrest, and A.S. Perelson. "Searching for diverse, coop- erative populations with genetic algorithms". In: Evolutionary Compu- tation 1.2 (1993).
- Lasarczyk, C. W. G. & and Wolfgang Banzhaf, P. D. Dynamic Subset Selection Based on a Fitness Case Topology Evolutionary Computation, 2004, 12, 223-242
- Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, R. I. McKay, and Dao Ngoc Phong. On the roles of semantic locality of crossover in genetic programming. Information Sciences, 235:195–213, 20 June 2013.
- Mauro Castelli, Leonardo Vanneschi, and Sara Silva. Semantic search-based genetic programming and the effect of intron deletion. IEEE Transactions on Cybernetics, 44(1):103–113, January 2014.
- Langdon, W. B. & Poli, R. Foundations of Genetic Programming Springer-Verlag, 2002
- McPhee, N. F., Ohs, B. & Hutchison, T., Semantic Building Blocks in Genetic Programming, in O'Neill, M et al. (eds.) Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008, Springer, 2008, 4971, 134-145

References

- A. Moraglio, *Towards a Geometric Unification of Evolutionary Algorithms*, PhD Thesis, University of Essex, UK, 2007.
- A. Moraglio, R. Poli, Topological Interpretation of Crossover, Genetic and Evolutionary Computation Conference, pages 1377-1388, 2004.
- A. Moraglio, A. Mambrini, L. Manzoni, Runtime Analysis of Mutation-Based Geometric Semantic Geometric Programming on Boolean Functions, Foundations of Genetic Algorithms, 2013.
- A. Moraglio, A. Mambrini, Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming for Basis Functions Regression, Genetic and Evolutionary Computation Conference, 2013.
- A. Mambrini, L. Manzoni, A. Moraglio, Theory-Laden Design of Mutation-Based Geometric Semantic Genetic Programming for Learning Classification Trees, *IEEE Congress on Evolutionary Computation* 2013.
- A. Moraglio, J. McDermott, M. O'Neill, Geometric Semantic Grammatical Evolution, SMGP workshop at PPSN, 2014.
- A. Moraglio, An Efficient Implementation of GSGP using Higher-Order Functions and Memoization, SMGP workshop at PPSN, 2014.
- J. Fieldsend, A. Moraglio. Strength through diversity: Disaggregation and multi-objectivisation approaches for genetic programming, GECCO, 2015 (to appear).
- 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
- R. Ffrancon, M. Schoenauer, Greedy Semantic Local Search for Small Solutions, Semantic Methods in Genetic Programming Workshop, GECCO'15, 2015.
- T.P. Pawlak, *Competent Algorithms for Geometric Semantic Genetic Programming*, PhD Thesis, Poznan University of Technology, 2015.
- T.P. Pawlak, K. Krawiec, Progress properties and fitness bounds for geometric semantic search operators, *Genetic Programming and Evolvable Machines*, Vol. 17, pp. 5-23, March 2016.