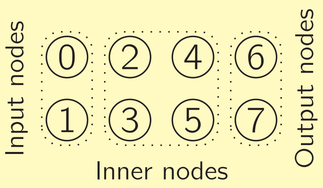
A Comparison between Geometric Semantic GP and Cartesian GP for Boolean Functions Learning

Andrea Mambrini[1] and Luca Manzoni[2]

[1] School of Computer Science, University of Birmingham a.mambrini@cs.bham.ac.uk
[2] Univ. Nice Sophia Antipolis, CNRS, I3S UMR 7271 luca.manzoni@i3s.unice.fr

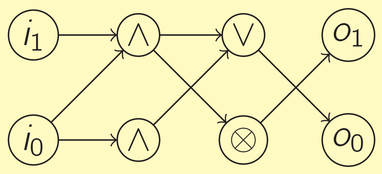
Cartesian GP

Cartesian GP (CGP) represents each individual as a grid. Each position on this grid is a node which computes a function according to its *function gene* and gets its inputs from some nodes on its left, according to its *connection genes*.



Function gene	Action
0	$\wedge$
1	$\vee$
2	$\otimes$

(0̲01 ; 0̲11 ; 1̲23 ; 2̲22 ; 54)



CGP natively support multiple output, so it is particularly suitable in solving multiple outputs problems.

Geometric Semantic GP

A Boolean expression $\{0,1\}^n \to \{0,1\}$ can be represented by its semantics, thus as a vector over $\{0,1\}^n$ defining the output behaviour for each possible input (the *output vector*).
Geometric Semantic GP (GSGP) uses geometric operators on the semantic space. Thus:

- a mutation produces an offspring whose output vector is similar to the one of the parent;
- a crossover produces an offspring whose output vector is a "combination" of the outputs of the parents.

Semantic Mutation. The offspring of the semantic mutation of T is, with equal probability $T \wedge \overline{M}$ or $T \vee M$, where M is a random minterm.
Effect: the output of the offspring differs from its parent by at most 1 bit.

0	1	1	1	$T = (x_1 \wedge x_2) \vee (x_1 \vee x_2)$	**Parent**
0	0	1	0	$M = x_1 \wedge \overline{x_2}$	**Random minterm**
0	1	**0**	1	$T \wedge \overline{M}$	**Offspring**

mutated bit

Block Mutation. Similar but a *block* on the output is changed.

0	1	1	1	$T = (x_1 \wedge x_2) \vee (x_1 \vee x_2)$	**Parent**
0	0	1	1	$M = x_1$	**Incomplete minterm**
0	1	**0**	**0**	$T \wedge \overline{M}$	**Offspring**

mutated block

Test Functions

***k*-even parity**. The output is 1 if and only if the k-bits input has an even number of bits set to 1.

***k*-multiplexer**. Input size is $k = h + \log_2 h$ bits. The first $\log_2 h$ bits represent an address on the remaining bitstring. The output is the value of the bit addressed by the first $\log_2 h$ bits.

Digital Adder (multiple outputs). The inputs are two n-bitstrings representing two n-bits integers plus one carry bit. The output is a carry bit and a n-bitstring representing the sum of the two inputs integers.

Digital Multiplier (multiple outputs). The inputs are two n-bitstrings representing two n-bits integers. The output is a $2n$-bitstring integer representing the product of the two inputs integers.

Black box scenario

The aim is to produce a boolean expression which *perfectly matches* the behaviour of the test functions above. Given that:

- The algorithm does not have access to the truth table of the test function
- The algorithm can give a candidate function $f(X_1, \cdots, X_n)$ to an oracle that will return the *fraction of matched outputs* for all the possible inputs (which are 2^n).

What we measure

Number of generations to produce an expression perfectly matching the behaviour of the test function

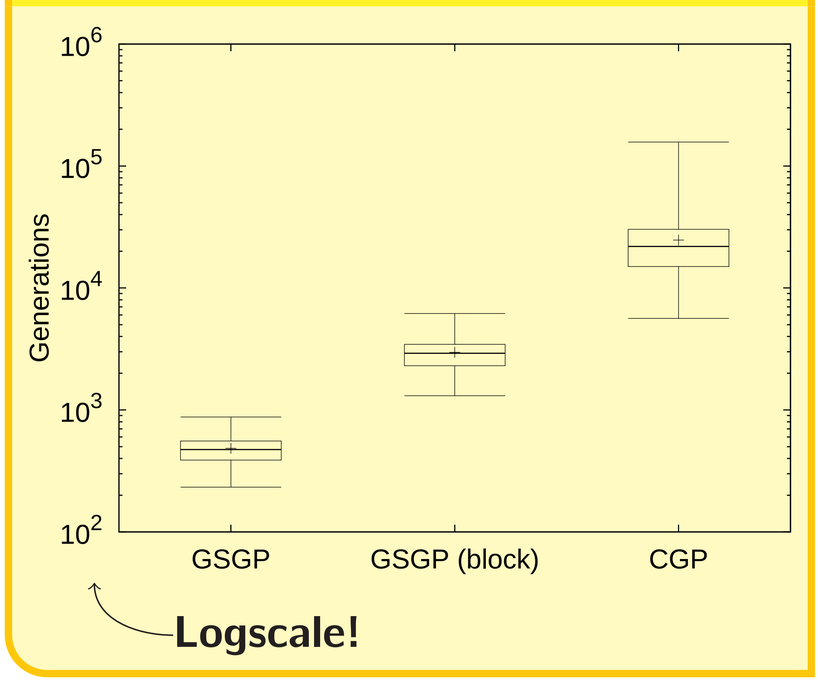
Experimental Settings

CGP. We use 4000 nodes with a mutation rate of 1%. The population size is 5 and each individual can use AND, OR, NOR, and NAND as function for the nodes.

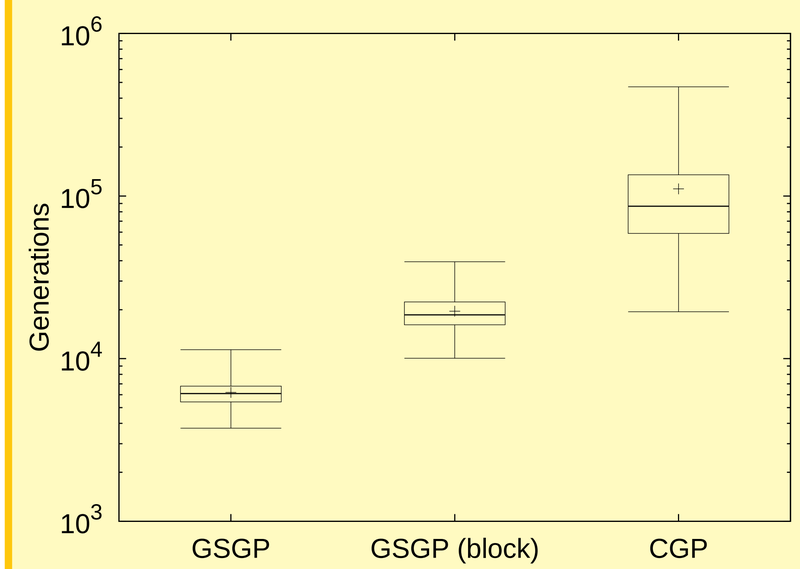
GSGP. We use semantic mutation and population size 1. To produce multiple outputs, every individuals is an array of single-output expressions, each one representing one output. At each generation one expression is randomly selected and semantic mutation is applied to it.

100 independent runs are performed for each test function. The fitness is the *fraction of correct output bits*.

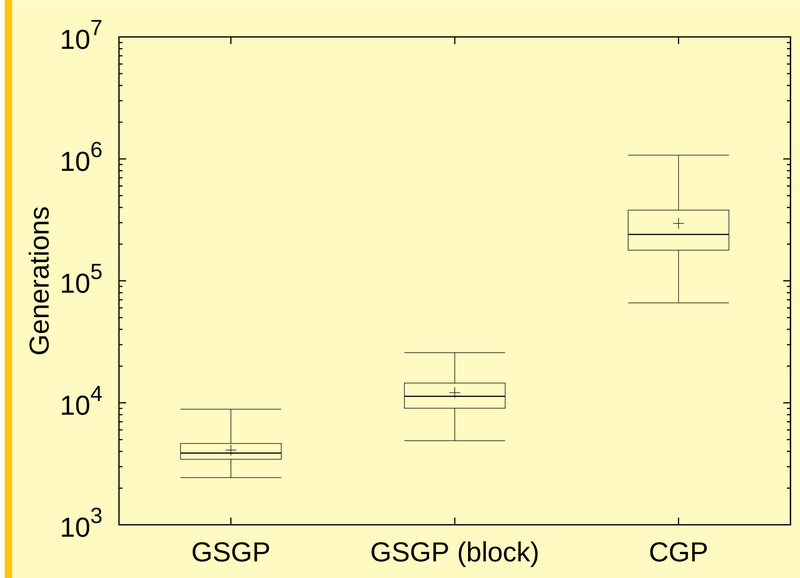
6-even parity



3-bits digital adder



3-bits digital multiplier



Results

In the setting proposed, with statistical significance, GSGP is the fastest method, followed by GSGP with block mutation. CGP is the slowest method.