Geometric Semantic Genetic Programming (GSGP): theory-laden design of variation operators

Andrea Mambrini

University of Birmingham, Birmingham UK

Science Coffee
ESTEC – Advanced Concept Team
12th July 2013

Research question

How can we do **provably good** design of search operators for genetic programming (GP)?

Outline

- Preliminaries (Runtime Analysis of RLS on Generalized OneMax)
- Traditional Genetic Programming, why is it **hard** to analyse?
- Geometric Semantic Genetic Programming (GSGP): why is it **easy** to analyse (and thus to design)?
- Analysis and design of Geometric Semantic Genetic Programming for Boolean Problems
- GSGP on other settings: Basis functions regression
- GSGP on other settings: Classification trees

OneMax Problem

- box-constrained binary problem
- fitness of a solution (to maximise): number of matching bits with a particular fixed bit-string (traditionally a string consisting of all 1s)

Example

- Size of the problem n=5
- Target string $\overline{X} = 11111$
- fitness of an individual X: $\sum_{i=1}^{5} 1 - HD(x_i, \overline{x}_i)$ (e.g. $X = 10110$ has fitness equal to 3).

RLS Algorithm

Algorithm - Random Local Search (RLS)

1. Inizialise P_0 with a bitstring $x \in \{0,1\}^n$
2. Create the offspring x' by flipping one bit at random
3. Select the fittest between x' and x for survival
4. Repeat from point 2 until stopping condition applies

Runtime of RLS on OneMax:

- An individual with fitness k is mutated to a better individual if one unmatched bit is flipped. This happens with probability $p = \frac{n-k}{n}$.
- The expected time to get to the optimal solution is thus: $E[T] = \sum_{k=1}^{n} \frac{n}{n-k} = O(n \log n)$.

Genetic Programming

Genetic programming is an evolutionary algorithm-based methodology to evolve functions (or computer programs).

- Individuals are functions represented for example as parse trees
- Mutation operators traditionally perform operation on the syntax of the tree
- The fitness function measures the input-output behaviour (semantic) of the function, usually comparing it with a target behaviour that is aimed to be reached.

Black-box boolean function learning

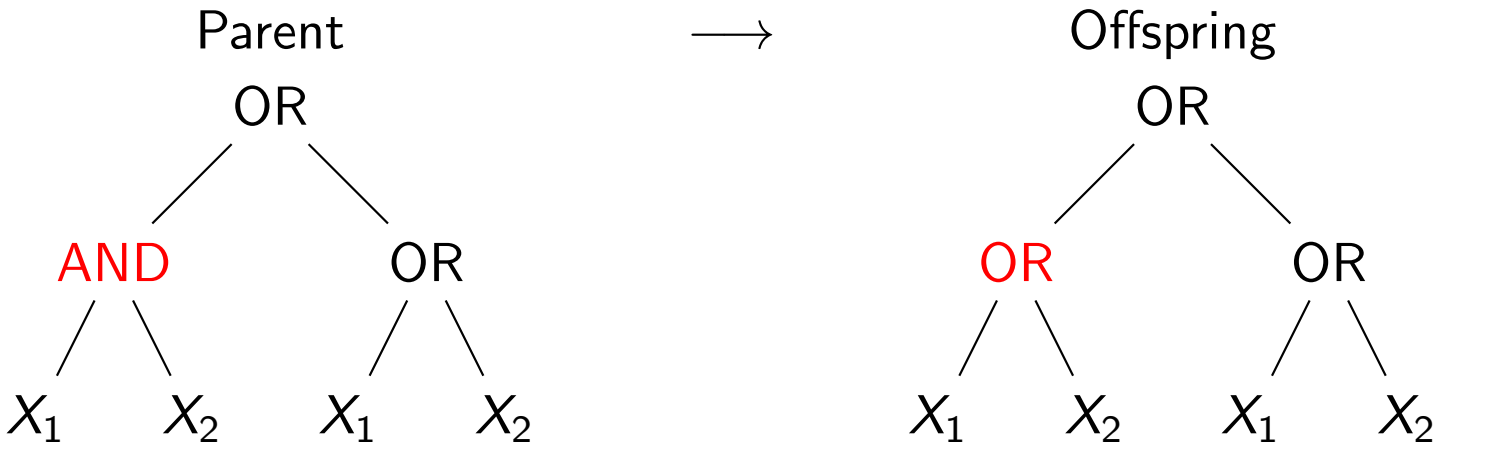
Which boolean expression produces the required output behaviour?

X_1	X_2	Output
1	1	1
1	0	0
0	1	1
0	0	0

Black-box setting:

- We don't have access to the truth table
- We can give our candidate function $f(X_1, X_2)$ to an oracle that will answer us the number of matched output rows.

Traditional Genetic Programming

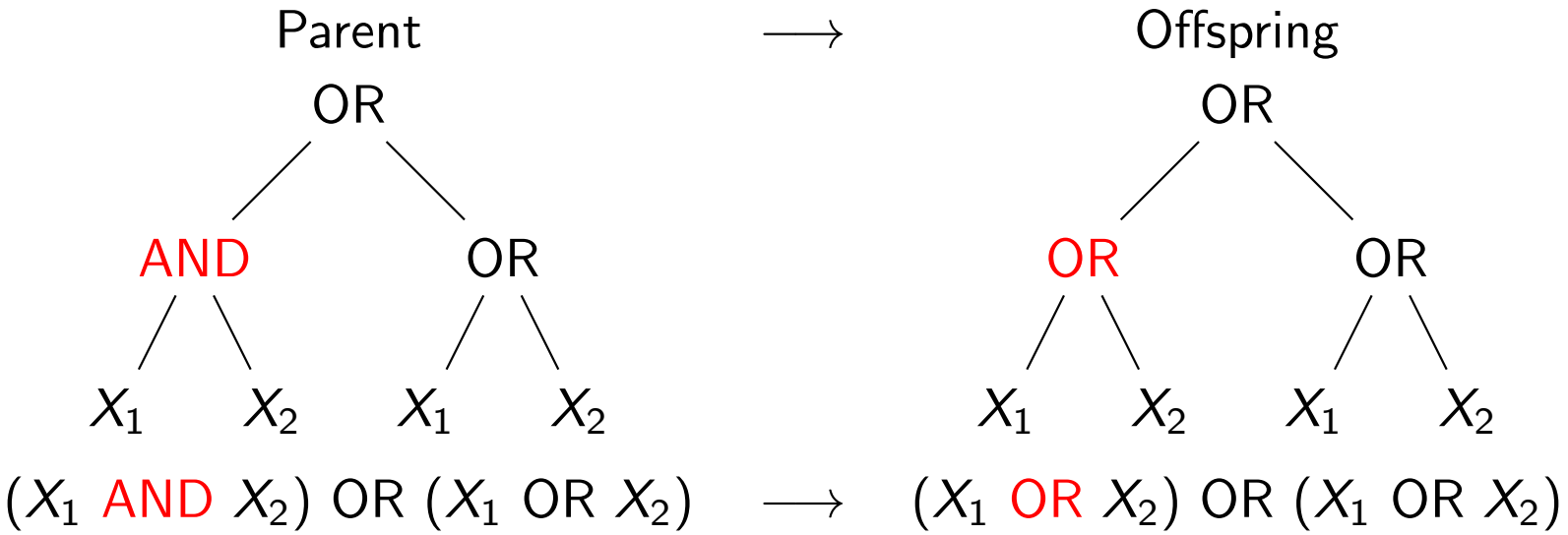


$(X_1 \text{ AND } X_2) \text{ OR } (X_1 \text{ OR } X_2) \longrightarrow (X_1 \text{ OR } X_2) \text{ OR } (X_1 \text{ OR } X_2)$

- Expressions are represented by trees.
- Variation operators produce offspring by syntactic manipulation of parent trees

X_1	X_2	Target Output	Parent's Output
1	1	**1**	1
1	0	**0**	1
0	1	**1**	1
0	0	**0**	0
Fitness			3

Traditional Genetic Programming

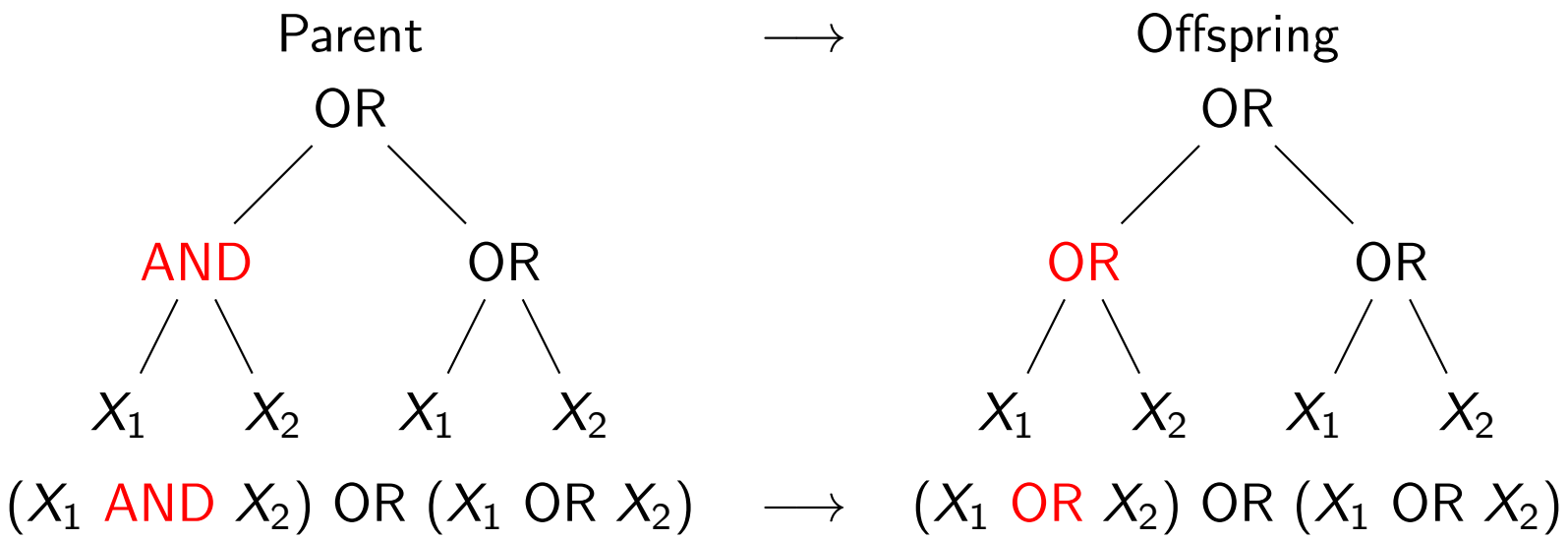


- Expressions are represented by trees.
- Variation operators produce offspring by syntactic manipulation of parent trees

Why GP is hard to analyse (and thus to efficiently design)?

- The fitness depends on the behaviour (semantic) of the function to evolve, while variation operators act on the syntax.
- How a syntactic variation operator affects the semantic of the expression?

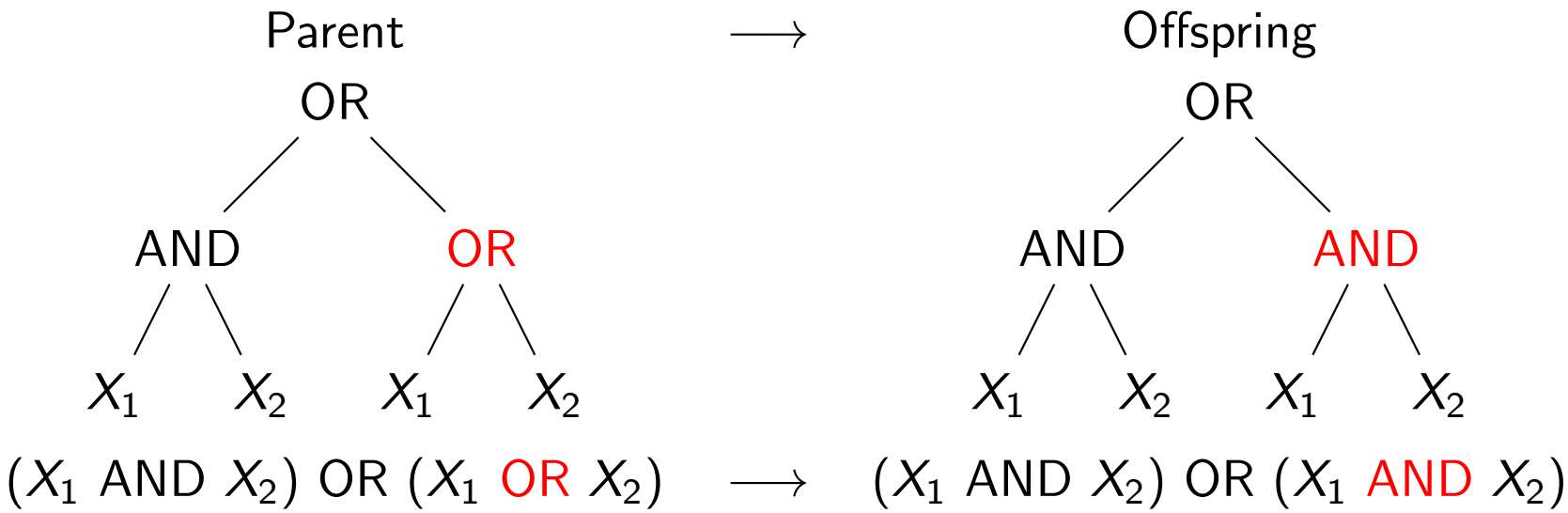
Traditional Genetic Programming



X_1	X_2	Output	Parent	Offspring
1	1	1	1	1
1	0	0	1	1
0	1	1	1	1
0	0	0	0	0

In this case the variation operator **hasn't produced any change** in the semantic of the function.

Traditional Genetic Programming



X_1	X_2	Output	Parent	Offspring
1	1	1	1	1
1	0	0	1	0
0	1	1	1	0
0	0	0	0	0

In this case it has produced a **big changes** on the truth table.

Can we improve the situation?

Can we redesign GP such that:

- it is easy to deal with it from a runtime point of view
- we can easily get runtime results for classes of problems
- we can use those results to design better operators
- those better operators actually outperform traditional GP on real problems

?

Let's find it out!

Semantic Genetic programming

Semantic variation operator

- acts on the syntax of an expression (given a parent expression produce as offspring another expression)
- **guarantee** some semantic features for the offspring (e.g. a semantic mutation might produce an offspring whose truth table always differs just by one row from the parent)

How to implement a semantic operator?

By trial & error use standard operators and reject offspring that do not conform to the semantic requirement $\rightarrow$ wasteful

By construction[1] : Operate on the syntax of the expression in a way that by construction the semantic criterion is satisfied

[1] A. Moraglio, K. Krawiec, C. Johnson – *Geometric Semantic Genetic Programming* – 5th Workshop on Theory of Randomized Search Heuristics, 2011

A semantic mutation operator[2]

Definition

Bit-flip point mutation: Given a parent function $P : \{0,1\}^n \to \{0,1\}$ the mutation returns the offspring boolean function $P' = \left(P \wedge \overline{M}\right) \vee \left(M \wedge \overline{P}\right)$, where M is a random minterm of all input variables.

Random minterm: $\mathrm{M} = X_1 \wedge \overline{X_2}$

Example: Parent: $\mathrm{P} = (X_1 \wedge X_2) \vee (X_1 \vee X_2)$

Offspring: $O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$

X_1	X_2	Output	M	Parent	Offspring
1	1	1	0	1	1
1	0	0	1	1	0
0	1	1	0	1	1
0	0	0	0	0	0

[2] A. Moraglio, A. Mambrini, and L. Manzoni – *Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming on Boolean Functions* – Foundations of Genetic Algorithms (FOGA 2013)

Semantic Bit-flip point mutation

X_1	X_2	Output	M	Parent	Offspring
1	1	1	0	1	1
1	0	0	1	1	0
0	1	1	0	1	1
0	0	0	0	0	0

Effect on the output vector (semantic):

- Select a row at random (randomly generating M)
- Flip the output of the function on that row (the row where M is true)

Basically at each generation we flip one bit from a bitstring (the output vector) aiming to reach a desired bit configuration...

that reminds RLS on OneMax!

Runtime

Search equivalence

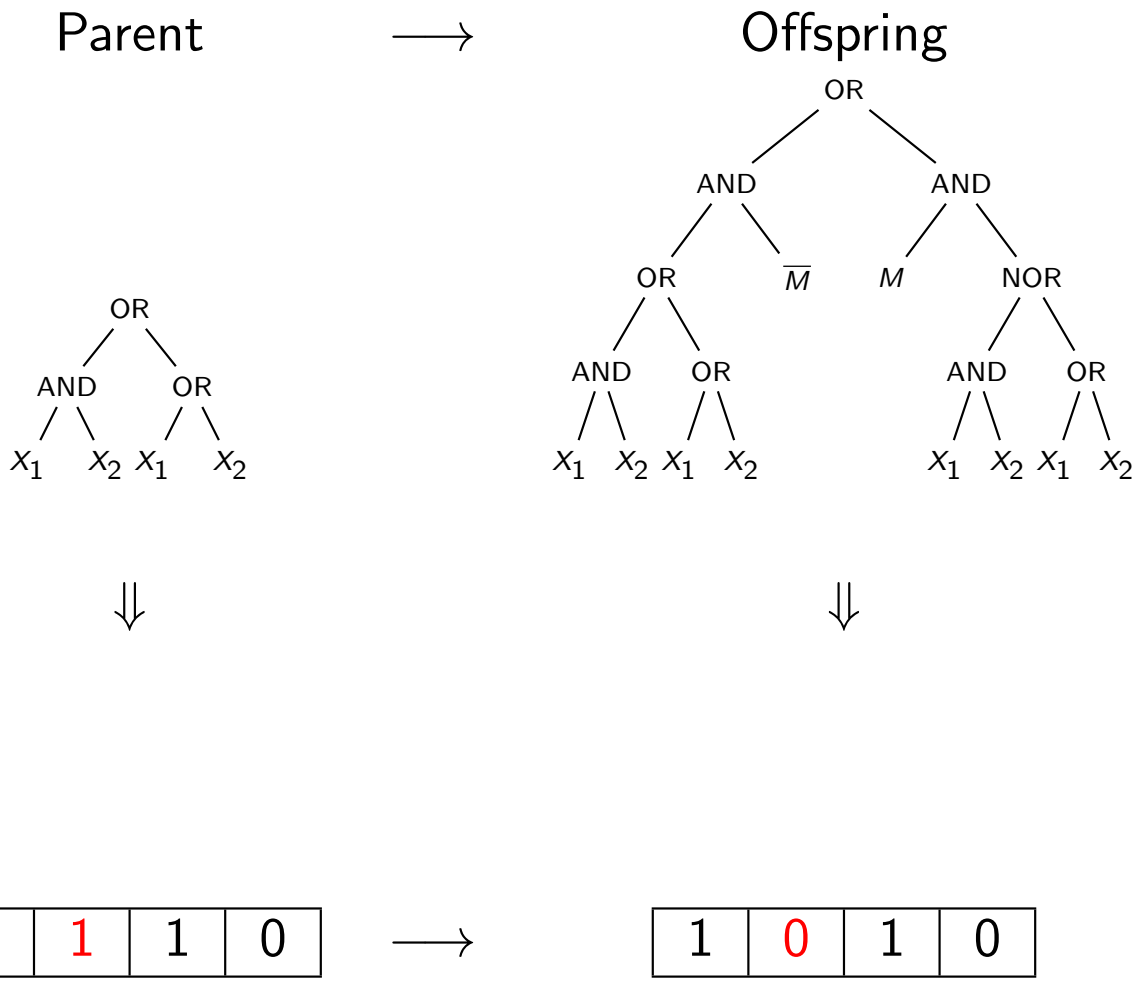
Genetic Programming using semantic operators solving **ANY** black-box boolean function learning problem

$\Updownarrow$ **is equivalent to** $\Updownarrow$

a Genetic Algorithm (evolving bitstrings) solving OneMax

The search outputs a tree (i.e., a boolean expression), but the runtime analysis can be done on the GA!

Equivalence



Runtime

Search equivalence

Genetic Programming using semantic operators solving **ANY** black-box boolean function learning problem

⇳ **is equivalent to** ⇳

a Genetic Algorithm (evolving bitstrings) solving OneMax

The search outputs a tree (i.e., a boolean expression), but the runtime analysis can be done on the GA!

Theorem

Geometric Semantic Genetic Programming solves the black box boolean function learning problem in time $T = O(N \log N)$, where N is the length of the output vector (i.e. the number of rows of the truth table).

Issues

The proposed operator still suffers from the following issues:

1. In the way we designed the semantic mutation operator ($O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$), the length of the offspring grows exponentially.
2. A fair choice of the problem size is the number of input variables n. Since $N = 2^n$, the runtime (which is $O(N \log N)$) is actually exponential in the problem size.

A shorter operator

Problem Since $O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$, the length of the offspring grows exponentially.
Solution:

Definition

Forcing point mutation: Given a parent function $P : \{0,1\}^n \to \{0,1\}$:

- generate a random minterm M
- return as offspring $P' = P \vee M$ with probability 0.5 and $P' = P \wedge \overline{M}$ with probability 0.5

Effect on the output vector: forcing a random bit (corresponding to the row that makes M true) to a random value (1 with probability 0.5, and 0 with probability 0.5).

Polynomial-sized training set

Problem The runtime $O(N \log N)$ is exponential in the problem size n (since $N = 2^n$).
Actually, in practice, the training set is *small*. Particularly we can assume it to have polynomial size w.r.t. the number of input variables.
$\tau = \text{poly}(n)$.

X_1	X_2	X_3	O
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	1

This transforms the problem seen by the EA on output vectors into a "sparse" version of OneMax. Still, using the bit-flip operator, the time to reach the optimum is exponential.

Incomplete minterms

Incomplete minterms force mutation on blocks of the output vector instead of single bits

X_1	X_2	X_3	$X_2 \wedge X_3$	$\overline{X_2} \wedge X_3$	$X_2 \wedge \overline{X_3}$	$\overline{X_2} \wedge \overline{X_3}$
1	1	1	**1**	0	0	0
1	1	0	0	0	**1**	0
1	0	1	0	**1**	0	0
1	0	0	0	0	0	**1**
0	1	1	**1**	0	0	0
0	1	0	0	0	**1**	0
0	0	1	0	**1**	0	0
0	0	0	0	0	0	**1**

We flip more bits at each generation keeping the length of the expression short.

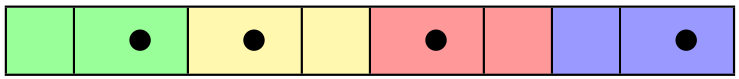
Probability of success

Definition

Fixed Block Mutation (FBM):

- Select randomly $v < n$ variables
- At each generation draw an incomplete minterm M comprising all the fixed v variables
- Use M for the forcing mutation

The output vector is partitioned into $b = 2^v$ blocks. At each generation a whole block is forced to the same value (0 or 1).



N = 8, b = 4

For $\tau = O(n^c)$

$$v = 2(c+1)\log_2(n) \Rightarrow \text{runtime:} O(n^{2c}\log n), p = O(1-2/n)$$

How fast is that?

We compared GSGP with the state of the art GP for boolean function, which is Cartesian Genetic Programming (CGP) on some boolean benchmarks (PARITY, MULTIPLEXER, ADDER, MULTIPLIER). GSGP appear to be much faster in all the benchmarks.

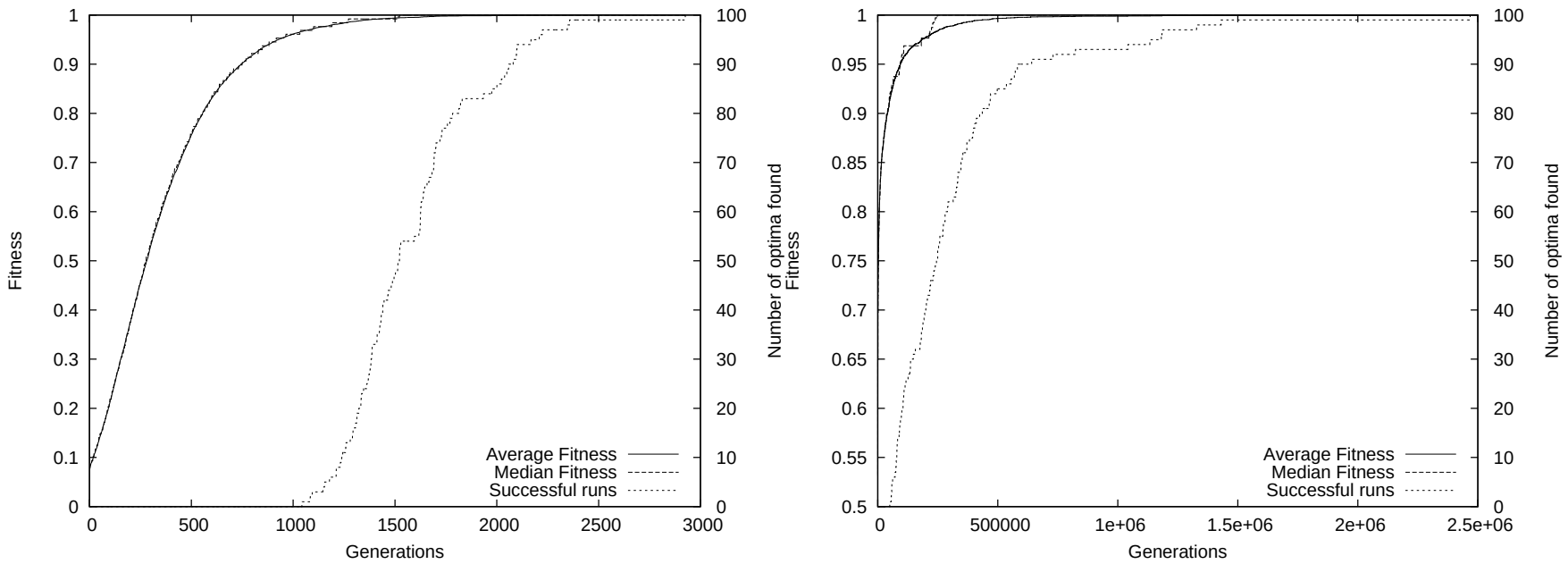


Figure: Comparison between GSGP (on the left) and CGP (on the right) on the 3-bits digital adder benchmark.

So far

- Unlike traditional GP, GSGP searches directly the semantic space of functions.
- GP with semantic operators on boolean expressions is equivalent to a GA with traditional operators on output vectors (bitstrings).
- For any boolean function to learn, the landscape seen by GSGP is OneMax. This allows us to analyse the runtime of GSGP and set some parameters using results from theory.
- We designed a block mutation (FBM) which reach the optimum in polynomial time with high probability on all black box boolean function learning problems.

Next:

- GSGP for other settings (real function and classifiers).

Other settings – Basis Functions Regression[3]

We want to learn a function $F : \mathbb{R}^n \to \mathbb{R}$ from the space of functions generated by the basis $(g_i(x), \ldots, g_m(x))$. We have k samples $\{(X_1, F(X_1)), \cdots (X_k, F(X_k))\}$.

Example $n = 3, k = 4$

	$x_{i,1}$	$x_{i,2}$	$x_{i,3}$	$f(X)$
X_1	0	0.7	4	0.9
X_2	2	3	1.3	1.4
X_3	4.5	-3	-2	8.5
X_4	8	7	1	-3.2

Blackbox setting:

- We don't have access to this table (neither the training points, nor the output of the function on the training points)
- We can give our candidate function $P(X)$ to an oracle that will answer us the distance of the output of the function on the training examples.

[3] A. Moraglio, A. Mambrini – *Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming for Basis Functions Regressions* – Genetic and Evolutionary Computation Conference (GECCO 2013)

What are we looking for?

We want an operator, that acting on a parent function $P(X)$ produce an offspring function $P'(X)$ with a predictable behaviour on the output vector $\{P'(X_1), \cdots, P'(X_n)\}$.

- Particularly since the output vector is a real vector we want a semantic operator behaving, on the output vector, as (1+1)-ES.

Algorithm - (1+1)-ES [Evolutionary Strategy]

1. Inizialise P_0 with a random real vector $X \in \mathbb{R}^n$
2. Generate a random gaussian vector $R \sim N(\mu, \Sigma)$
3. Create the offspring $X' = X + R$.
4. Select the fittest between X' and X for survival
5. Repeat from point 2 until stopping condition applies

Let's find an operator

We proved that, knowing the input points $\{X_1, \cdots, X_n\}$, we could induce the operator of (1+1)-ES using the following operator

$$P'(X) = P(X) + r(X)$$

where $r(X) = c_0 + c_1 g_1(X) + \cdots + c_n g_n(X)$ and $(c_0, \cdots, c_n) \sim G^{-1} * N(\mu, \Sigma)$, with

$$G = \begin{pmatrix} g_1(X_1) & g_2(X_1) & \cdots & g_n(X_1) \\ g_1(X_2) & g_2(X_2) & & \\ \vdots & \vdots & \ddots & \vdots \\ g_1(X_k) & g_2(X_k) & \cdots & g_n(X_k) \end{pmatrix}$$

Then $\{P'(X_1), \cdots, P'(X_n)\} = \{P(X_1), \cdots, P(X_n)\} + N(\mu, \Sigma)$

Re-using theory from (1+1)-ES

From Jägersküpper 2007[4], we can get suggestion on how to choose the variance matrix Σ and we can derive upper bound for the runtime. Particularly:

- Σ should be an isotropic diagonal matrix $\text{diag}(\sigma, \ldots, \sigma)$
- σ should be adapted throughout the run following the $1/5$ rule, which state to reset σ every k mutation to:
  - $\sigma' = \sigma / c$ if $f > 1/5$
  - $\sigma' = \sigma \cdot c$ if $f < 1/5$

  Where f is the fraction of successful mutation (the ones that actually improve the fitness) and $0.8 < c < 1$.

Following those guidelines the time to get to a solution within ε distance from the optimum is thus $\Theta(k \log \frac{k}{\varepsilon})$ (where k is the number of training examples).

[4] J. Jägersküpper. Analysis of a simple evolutionary algorithm for minimization in euclidean spaces. Theoretical Computer Science, 379(3):329–347, 2007.

How fast is that?

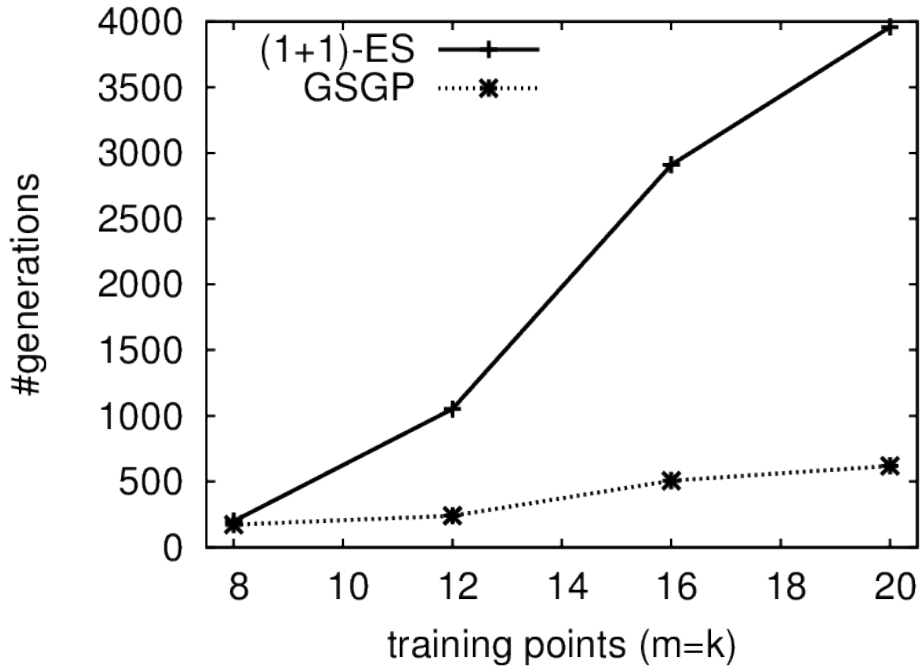


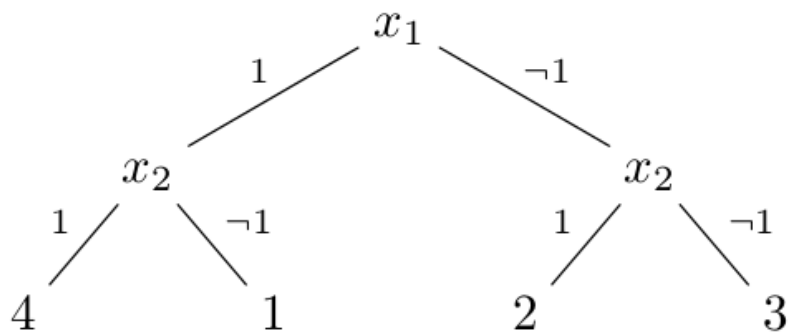
Figure: Comparison between (1+1)-ES on the space of coefficients with the proposed GSGP operator.

We see how the theoretical-designed GSGP operator performs much better than the naive approach of evolving the coefficients of the linear combination of basis functions.

Other settings – Classification trees[5]

From classification tables to classification trees

X_1	X_2	O
1	1	4
1	3	1
2	1	2
3	2	3



It is a generalization of the Boolean case in which:

- Each variable X_i has a value from a finite sized input alphabet IS_i
- We evolve decision trees instead of boolean expressions

[5] 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 (CEC 2013)

The operator

Definition

Fixed Block Mutation (FBM) Let us consider a fixed set of $v < n$ input variables (fixed in some arbitrary way at the initialisation of the algorithm). Given a parent classifier `T`, the mutation FBM returns the offspring classifier `T' = IF CONDR THEN OUTR ELSE T` where `CONDR` is an incomplete random condition comprising all v fixed variables, and `OUTR` is a random output symbol.

- Similarly to the boolean case this corresponds to forcing a block of rows to `OUTR`.
- Runtime results obtained reducing the problem to RLS solving OneMax on a string with symbols from a finite alphabet (similar to the bitstring case)
- Again it is possible to fix v in order to solve the problem in polynomial runtime with high probability (particularly if the training set is $O(n^c)$, then $v = \lceil (2c+1) \log_{r_m}(n) \rceil$, where r_m is the minimum size of the input alphabet across all variables).

Table: Example of FBM. The first four columns represent the truth table of the classifier to learn. The fifth and sixth columns are the output vector of the parent T and of the offspring T' after applying FBM with `CONDR` = ($X_1 = 2$ `AND` $X_2 = 1$) and `OUTR`=3. Horizontal lines separate blocks of the partition of the output vectors obtained by fixing variables X_1 and X_2.

X_1	X_2	X_3	Y	$P(T)$	$P(T')$
1	1	1	4	4	4
1	1	2	2	2	2
1	1	3	1	2	2
1	2	1	4	4	4
1	2	2	2	4	4
1	2	3	2	3	3
2	1	1	2	**1**	**3**
2	1	2	3	**4**	**3**
2	1	3	3	**2**	**3**
2	2	1	2	2	2
2	2	2	1	3	3
2	2	3	3	3	3

Conclusion

Take home messages:

- Geometric Semantic Genetic Programming makes the search performed by GP on the space of functions (trees) equivalent to that performed by a GA on the output space.
- It is thus possible to easily get runtime results for GP reusing runtime results for GA
- We can use those theoretical results to design good semantic GP operators

Future work:

- Design GSGP operators for other settings (i.e. Classifier with real inputs and regression trees)
- Study the generalization ability of GSGP as a machine learning tool
- Test GSGP on real-world problems

Thank you!