Theory-Laden Design of Mutation-Based Geometric Semantic Genetic Programming for Learning Classification Trees

Andrea Mambrini
School of Computer Science
University of Birmingham
B15 2TT, Birmingham, UK
Email: a.mambrini@cs.bham.ac.uk

Luca Manzoni
Dipartimento di Informatica, Sistemistica e Comunicazione. Università degli Studi di Milano-Bicocca
Viale Sarca 336, 20126 Milano, Italy
Email: luca.manzoni@disco.unimib.it

Alberto Moraglio
School of Computer Science
University of Birmingham
B15 2TT, Birmingham, UK
Email: a.moraglio@cs.bham.ac.uk

***Abstract*—Geometric Semantic Genetic Programming (GSGP) is a recently introduced form of Genetic Programming (GP), rooted in a geometric theory of representations, that searches the semantic space of functions. The fitness landscape seen by GSGP is *always* – for any domain and for any problem – unimodal with a linear slope by construction. This makes the search for the optimum much easier than for traditional GP, and it opens the way to analyse theoretically in a easy manner the optimisation time of GSGP in a *general setting*. Very recent work proposed a runtime analysis of mutation-based GSGP on the class of *all* Boolean function learning problems. We present a runtime analysis of mutation-based GSGP on the class of *all* classification tree learning problems, which is a classical application domain of standard GP.**

I. Introduction

Traditional Genetic Programming searches the space of functions (programs) by using search operators that manipulate their syntactic representation, regardless of their actual semantics. Recently, a number of approaches have appeared that use the semantics of programs in various ways to improve the search of GP [1], [2], [3], [4], [5], [6]. Whereas these semantically aware methods are promising, their implementations are very wasteful as heavily based on trial-and-error: search operators are implemented via acting on the syntax of the parents to produce offspring, which are accepted only if some semantic criterion is satisfied. More importantly from a theoretical perspective, these implementations do not provide insights on how syntactic and semantic searches relate to each other. Is a *direct* implementation of semantic operators possible? That is, can we act on the syntax of the parent programs and produce offspring that are *guaranteed* to respect some semantic criterion or specification by construction?

Geometric Semantic Genetic Programming [7], [8] is a novel form of genetic programming that answers this question in the affirmative. GSGP uses geometric crossover and geometric mutation [9], [10] to search *directly* the semantic space of functions/programs. Informally, semantic geometric crossover and semantic geometric mutation generate offspring that are guaranteed to be, respectively, "semantically intermediate" and "semantically near" their parents. These search operators can be directly implemented for different domains following a *simple formal recipe* [7], [8], which was used to derive specific forms of GSGP for a number of classic GP domains (i.e, Boolean functions, arithmetic functions and classifiers).

The fitness landscape seen by the semantic geometric operators is *always* unimodal with a linear slope (cone landscape) by construction, as the fitness of an individual is by definition its *semantic distance* to the optimum individual. This suggests that GSGP performs better than standard GP. GSGP was compared with standard GP on several well-known problems across domains (finding Boolean functions, polynomial regressions, and classification tasks) and it consistently found much better solutions with the same budget of fitness evaluations [7], [8]. Furthermore, GSGP has been found more efficient and generalising better than standard GP on some initial studies on real-world problems [11].

Genetic programming has been hard to analyse theoretically. The literature comprises several approaches: schema theory [12], Markov models of GP search [13], theory-laden methods to combat bloat [14], analysis of search space of GP programs and experimental hardness studies [12]. There are also few theoretical works on GP from a semantic perspective [15], [16]. Due to the difficulty of analysing GP, there is only very initial work on its runtime analysis. Durrett et al. [17] present the runtime analysis of a mutation-based GP on very simplified problems (i.e., ORDER and MAJORITY).

The cone landscape seen by GSGP makes it very attractive from a theoretical point of view. When GSGP is applied to search the space of Boolean functions, the GSGP search is equivalent to a GA search on OneMax-like problems. Based on this, very recently [18], a runtime analysis of GSGP with various types of mutations on the class of *all* Boolean functions has been possible by simply extending known runtime results for GAs. Remarkably, it was found that GSGP finds the optimum Boolean function in polynomial time on most of the Boolean problems.

We continue this line of investigation and present a runtime analysis of GSGP on the class of *all* classification tree learning problems. This is a large class of problems whose functions to learn have categorical input variables, i.e., they take values from a limited fixed set of discrete or symbolic values, and categorical outputs. Boolean learning problems can be seen as a specific subclass of classification problems with two categorical values for the input variables and the outputs.

However, classification problems and Boolean problems differ in the way solutions are normally represented in GP, which are classification trees for the former, and parse trees of Boolean expressions using AND, OR, and NOT operators for the latter. The different representations give rise to different issues in the design of efficient semantic operators.

II. Geometric Semantic Genetic Programming

A. General Definitions

A search operator $CX : S \times S \rightarrow S$ is a *geometric crossover* w. r. t. the metric d on S if for any choice of parents p_1 and p_2, any offspring $o = CX(p_1, p_2)$ is in the segment $[p_1, p_2]$ between parents, i.e., it holds that $d(p_1, o)+d(o, p_2) = d(p_1, p_2)$. A search operator $M : S \rightarrow S$ is a *geometric ϵ-mutation* w. r. t. the metric d if for any parent p, any of its offspring $o = M(p)$ is in the ball of radius ϵ centered in the parent, i.e., $d(o, p) \leq \epsilon$. Given a fitness function $f : S \rightarrow \mathbb{R}$, the geometric search operators induce the fitness landscape identified by the triple (f, S, d). Many well-known recombination operators across representations are geometric crossovers [10].

For most applications, genetic programming can be seen as a supervised learning method. Given a training set made of fixed input-output pairs $T = \{(x_1, y_1), ..., (x_N, y_N)\}$ (i.e., fitness cases), a function $h : X \rightarrow Y$ within a certain fixed class H of functions (i.e., the search space specified by the chosen terminal and function sets) is sought that interpolates the known input-output pairs. I.e., for an optimal solution h^* it holds that $\forall (x_i, y_i) \in T : h^*(x_i) = y_i$. The fitness function $F_T : \mathrm{H} \rightarrow \mathbb{R}$ measures the error of a candidate solution h on the training set T. Compared to other learning methods, two distinctive features of GP are (i) it can be applied to learn virtually any type of functions, and (ii) it is a *black-box method*, as it does not need explicit knowledge of the training set, but only of the errors on the training set.

We define the *genotype-phenotype mapping* as the function $P : \mathrm{H} \rightarrow Y^{|X|}$ that maps a representation of a function h (i.e., its genotype) to the vector of the outcomes of the application of the function h to all possible input values in X (i.e., its phenotype), i.e., $P(h) = (h(x_1), ..., h(x_{|X|}))$. We can define a *partial* genotype-phenotype mapping by restricting the set of input values X to a given subset X' as follows: $P_{X'} : \mathrm{H} \rightarrow Y^{|X'|}$ with $P_{X'}(h) = (h(x_1), ..., h(x_{|X'|}))$ with $x_i \in X'$. Let $I = (x_1, ..., x_N)$ and $O = (y_1, ..., y_N)$ be the vectors obtained by splitting inputs and outputs of the pairs in the training set T. The output vector of a function h on the training inputs I is therefore given by its partial genotype-phenotype mapping $P_I(h)$ with input domain restricted to the training inputs I, i.e., $P_I(h) = (h(x_1), ..., h(x_N))$. The training set T identifies the partial genotype-phenotype mapping of the optimal solution h^* restricted to the training inputs I, i.e., $P_I(h^*) = O$.

Traditional measures of error of a function h on the training set T can be *interpreted as distance* between the target output vector O and the output vector $P_I(h)$ measured using some suitable metric D, i.e., $F_T(h) = D(O, P_I(h))$ (to minimise). For example, when the space H of functions considered is the class of classification functions, the input and output spaces are $X = \{0, 1, \ldots, di - 1\}^n$ and $Y = \{0, \ldots, do - 1\}$, where di and do are the cardinalities of the input and output alphabets, and the output vector restricted to the training inputs is a vector of integers (from the output domain) of size N (i.e., size of training set). A suitable metric D to measure the error as a distance between integer vectors is e.g. the Hamming distance (HD), when the outputs are interpreted as categorial values, or also the Manhattan distance (MD), when they are interpreted as quantitative values. We focus on categorical outputs.

We define *semantic distance* SD between two functions $h_1, h_2 \in \mathrm{H}$ as the distance between their corresponding output vectors measured with the metric D used in the definition of the fitness function F_T, i.e., $SD(h_1, h_2) = D(P(h_1), P(h_2))$. The semantic distance SD is a genotypic distance induced from a phenotypic metric D, via the genotype-phenotype mapping P [1].

We define *semantic geometric crossover and mutation* as the instantiations of geometric crossover and geometric mutation to the space of functions H endowed with the distance SD. E.g., semantic geometric crossover SGX on classification functions represented e.g. as classification trees returns offspring classification functions (i.e., trees) such that the output vectors of the offspring are in the Hamming segment between the output vectors of the parents (w. r .t. all $x_i \in X$).

When the training set covers all possible inputs, the *semantic fitness landscape* seen by an evolutionary algorithm with semantic geometric operators is, from the definition of semantic distance, a unimodal landscape in which the fitness of a solution is its distance in the search space to the optimum. This holds for any domain of application of GP (e.g., Boolean, Classification, Arithmetic, Program), any specific problem within a domain and for any choice of metric for the error function. Naturally, in practice, the training set covers only a fraction of all possible input-output pairs of a function. This has the effect of *adding a particular form of neutrality* to the cone landscape, as only the part of the output vector of a function corresponding to the training set affects its fitness, the remaining large part is "inactive".

GP search with geometric operators w. r .t. the semantic distance SD on the space of functions H (represented e.g. as trees) is formally equivalent to EA search with geometric operators w. r .t. the distance D on the space of output vectors. This is because: (i) semantic classes of functions are in bijective correspondence with output vectors, as "functions with the same output vector" is the defining property of a semantic class of function; (ii) semantic geometric operators on functions (i.e., trees) are isomorphic to geometric operators on output vectors, as SD is induced from D via the genotype-phenotype mapping P.

B. Representations of Classification Functions

A classification function can be represented in various way. It can be represented by explicit enumeration of its output values for any combination of the input values, analogously to a truth table representation for Boolean functions. Similarly, it can be represented by its output vector (i.e., the output column

[1] P is generally non-injective (i.e., different genotypes may have the same phenotype), but it can be made bijective by interpreting a genotype as a representation of a semantic class (i.e., of all functions with the same semantics) without affecting the subsequent analysis (see [18]).

of its truth table) when the input combinations are considered in some arbitrary but fixed order.

It can also be represented as a nested structure of IF−THEN−ELSE statements, whose conditions can be arbitrary Boolean expressions combining atomic conditions on input variables and input symbols, and whose THEN and ELSE clauses are nested IF−THEN−ELSE statements or output symbols. As we are focusing on categorical variables, relevant atomic conditions are equality/non-equality tests between two input variables or between an input variable and an input symbol. Classifiers with conditions using Boolean expressions can always be expanded and expressed in a simpler way as nested IF−THEN−ELSE statements using only atomic conditions testing the equality of a single input variable to an input symbol. Note, however, the expansion may increase of a multiplicative factor the size of the resulting simpler classifier.

Perhaps, the most used and most natural representation of a classification function is a classification tree. In this representation, non-terminal nodes correspond to input variables and branching edges from a node represent the input values that variable takes. Terminal nodes correspond to output symbols. The output for a given input combination is determined by starting from the root node and following a path in the tree dictated by the input combination to reach a terminal node containing the output value. Classification trees are closely related to the IF−THEN−ELSE representation. The nested IF−THEN−ELSE representation with atomic conditions correspond graphically to binary classification trees (i.e., trees whose all non-terminal nodes have two sub-trees). Classification trees correspond to nested SWITCH statements that can be expanded into nested IF−THEN−ELSE statements. Furthermore, classification trees can be represented more compactly using a directed acyclic graph representation analogous to Boolean Decision Diagrams in which subtrees that occur more than once in the original decision tree are not duplicated but referred using an edge pointing to the first occurrence of the subtree when re-occurring.

Finally, we note that Boolean expressions can be seen as classifiers whose sets of input symbols and output symbols have two values (i.e., Boolean classifiers). However, classification problems and Boolean problems differ in the way solutions are normally represented in GP, which are classification trees for the former, and parse trees of Boolean expressions using AND, OR, and NOT operators for the latter.

In the following, we design semantic operators to evolve efficiently, both in time and space, optimal classification functions *represented as standard classification trees* as this is the most natural and used representation.

C. Construction of Semantic Operators

The commutative diagram below illustrates the relationship between the semantic geometric crossover GX_{SD} on genotypes (e.g., trees) on the top, and the geometric crossover (GX_D) operating on the phenotypes (i.e., output vectors) induced by the genotype-phenotype mapping P, at the bottom. It holds that for any $T1, T2$ and $T3 = GX_{SD}(T1, T2)$ then $P(T3) = GX_D(P(T1), P(T2))$.

TABLE I. TRAINING SET FOR THE CLASSIFIER PROBLEM (FIRST 3 COLUMNS). OUTPUT VECTORS OF PARENT CLASSIFIER T AND OFFSPRING T' AFTER MUTATION WITH $CONDR = (X_1 == 2$ AND $X_2 == 2)$ AND $OUTR = 4$ (COLUMN 4 AND 5)

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

$$\begin{array}{ccccc} T1 & \times & T2 & \xrightarrow{GX_{SD}} & T3 \\ \downarrow P & & \downarrow P & & \downarrow P \\ O1 & \times & O2 & \xrightarrow{GX_D} & O3 \end{array} \quad (1)$$

The problem of finding an algorithmic characterization of semantic geometric crossover can be stated as follows: given a family of functions H, find a recombination operator GX_{SD} (unknown) acting on elements of H that induces via the genotype-phenotype mapping P a geometric crossover GX_D (known) on output vectors. E.g., for the case of classification functions with fitness measure based on Hamming distance, output vectors are integer vectors and GX_D is the discrete recombination that uses a random mask to recombine integer vectors, which is known to return offspring vectors on the Hamming segment between parent vectors [10]. We want to derive a recombination operator acting on classification functions that corresponds to the discrete recombination on their output vectors.

Definition 1: Given two parent classifiers $\texttt{T1}, \texttt{T2} : IS_1 \times \ldots \times IS_n \rightarrow OS$, the recombination SGXP returns the offspring classifier `T3 = IF CONDR THEN T1 ELSE T2` where `CONDR` is a random boolean condition on the input variables. Given a parent classifier `T`, the mutation SGMP returns the offspring classifier `T' = IF CONDR THEN OUTR ELSE T` where `CONDR` is a random condition which is true only for a single input combination of all input variables (complete condition), and `OUTR` is a random output symbol.

Theorem 1: SGXP and SGMP are, respectively, a semantic geometric crossover and a semantic 1-geometric mutation for the space of classifiers with fitness function based on Hamming distance, for any training set and any problem.

The proof of the previous theorem can be found in [7]. In the following, we give an example to illustrate the theorem for the SGMP mutation. The input/output pairs describing the target classifier are in the first three columns of Tab. I. The classifier has $n = 2$ input variables, with input sets $IS_1 = \{1, 2, 3\}$ and $IS_2 = \{1, 2\}$, and output set $OS = \{1, 2, 3, 4\}$. Let us consider a parent classifier `T = IF` $(X_1 ==$ `1) THEN 2 ELSE 3`. Its fitness is $f(P) = 3$, which is the Hamming distance between the output vector $P(T)$ (column 4 in Tab. I) and the target output vector O (column 3 in Tab. I). The output vector $P(T)$ is obtained by querying `T` for all input combinations of X_1 and X_2. Say we apply to `T` the mutation SGMP with random condition parameter `CONDR =` $(X_1$ `== 2 AND` X_2 `== 2)` and random output parameter `OUTR=4`. The offspring

T′ is obtained by substituting CONDR, OUTR and T in the mutation scheme T′ = IF CONDR THEN OUTR ELSE T obtaining T′ = IF (X_1 == 2 AND X_2 == 2) THEN 4 ELSE (IF (X_1 == 1) THEN 2 ELSE 3). The output vector $P(T')$ is in column 5 in Tab. I. Note that the semantic distance between parent and offspring is $SD(T,T') = HD(P(T),P(T')) = 1$ as their output vectors differ only at one position. This shows that in this particular case SGMP is a semantic 1-geometric mutation. This holds generally for each choice of the parent T, complete condition CONDR and output value OUTR.

As the offspring classifier of SGMP is obtained by the application of a functional form (i.e., mutation scheme) to its parent classifier, the semantics (i.e., underlying function) of the offspring does not depend on the actual representation of its parent (i.e., its syntax) but only on its semantics. This applies to SGXP as well, and it is in fact a characteristic of all geometric semantic operators [7]. Consequently, the search of *semantic GP is not affected by how functions are actually represented* [2]. However, the representation plays an important role in terms of space efficiency of semantic GP, as we illustrate next. Continuing the example above, Fig. 1(left) shows the parent T represented as a classification tree. Fig. 1(centre) shows the offspring T′ represented as a classification tree after expanding the condition CONDR, i.e., T′ = IF (X_1 == 2 AND X_2 == 2) THEN 4 ELSE T which expanded becomes T′ = IF (X_1 == 2) THEN (IF (X_2 == 2) THEN 4 ELSE T) ELSE T. Fig. 1(right) shows the completely expanded offspring. Using the classification tree representation the size of the offspring T′ is more than double the size of its parent T. So, the size of the *offspring grows exponentially in the number of mutation applications*, which is a serious drawback of this representation. There are a number of remedies: (i) use a representation with more complex conditions so that the expansion of the condition is not required; (ii) use a direct acyclic graph representation; (iii) apply algebraic simplification to obtain smaller classification trees with the same semantics. Options (i) and (ii) have the drawback of not giving a classification tree as a solution, which is what we would like to have in the end. When the optimum is found, we could always convert it into a classifier, but of exponential size. Option (iii) may be computationally expensive, and it does not guarantee to counteract the exponential growth. We pursue a further option which consists in designing search operators that while operating on the classification tree representation still guarantee a polynomial growth.

III. Runtime Analysis

Recently, the runtime of GSGP on the class of *all* Boolean problems has been analysed showing that this form of GP can find the optimum efficiently for most problems [18]. As Boolean functions can be regraded as a special type of classifiers, and the analysis does not depend on the solution representation, it turns out that the result can be readily generalised to classification problems with categorical inputs and outputs. We present the generalisation in section III-A. As we are interested in evolving classifiers represented as classification trees, in section III-B we present mutation operators that guarantee polynomial growth of classification trees.

A. Generalisation of GSGP Runtime of Boolean Problems

The problem class we consider is *general black-box classification*, defined as follows. Let $p : IS_1 \times IS_2 \times \cdots \times IS_n \to OS$ be an unknown classifier of n input variables (n is the size of the problem). The domains IS_i of the input variables x_i and OS of the output are finite set of symbols (of constant size in n). Let $T = \{(i_1^1, \cdots, i_n^1, o^1), ..., (i_1^k, \cdots, i_n^k, o^k)\}$ be a fixed set of *polynomially many* input/output pairs of the function p where the inputs were sampled *uniformly at random* in the input domain. The pair (p, T) is an instance of the problem. The aim is to find a classifier h that best approximates p on the training examples, in the *black-box settings*, without the knowledge of the training set, but only via consulting an oracle that returns the errors of candidate solutions on the training set submitted to it.

The algorithm we consider is Randomised Local Search (RLS) on the space of classifiers using different types of semantic mutation operators. The mutation SGMP, by construction, corresponds to a variant of point mutation on the output vector space, that forces a randomly chosen entry of the output vector to a randomly selected output symbol (forcing mutation).

As discussed in Section II-A, for any choice of problem p and training set T, the fitness landscape seen by GP with geometric semantic operators is *always* a cone landscape, in which the fitness of a classifier (to minimise) is the Hamming distance of the output vector of the classifier queried on the training inputs to the vector of the training outputs. The size N of the output vector is exponential in the problem size n (i.e., it is $N = \prod_{i=1}^{n} |IS_i|$ where $|IS_i|$ is the cardinality of the i-th input alphabet). The fitness landscape can be understood as a generalisation of the OneMax black-box class to finite alphabet elements on exponentially long strings in which most of vector entries are neutral, and the remaining entries (corresponding to elements of training set), whose locations in the output vector are unknown, give each a unitary contribution to the fitness ("sparse" OneMax problem).

Theorem 2: A mutation only GSGP using as mutation operator SGMP finds a classifier tree matching the training set in time $O(R_o N \log N)$ where N is the length of the complete classification table and R_o is the size of the output set OS. In particular, the runtime is exponential in n.

The proof for the case of binary output alphabet is in [18] and the generalisation to finite alphabet is straighforward.

In order to solve the sparse OneMax problem in polynomial time on the training set size, hence in $poly(n)$, we need to randomly change the value, in expectation, of at least one position of the output vector associated with the training set at each generation. Thus at each generation in expectation $\Theta(N/\tau)$ values in the vector need to be changed at randomly chosen positions. Applying many times the point mutation operator (SGMP) is not feasible since it would require an exponential number of operations at each generation. In [18] for the case of Boolean functions, this problem is solved

[2]This is unlike traditional GP, where the mutation operator is defined on the syntax of the parent and the semantics of the offspring depends heavily on the syntax of its parent.

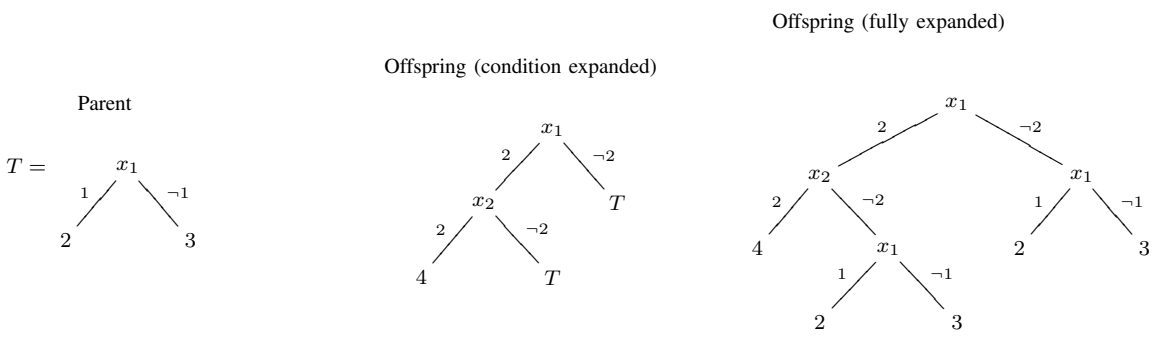

Fig. 1. T is the parent function. The offspring T' is obtained applying the mutation operator SGMP with CONDR = (X_1 == 2 AND X_2 == 2) and OUTR=4.

TABLE II. EXAMPLE OF FBM. THE FIRST FOUR COLUMNS REPRESENT THE TRUTH TABLE OF THE TARGET CLASSIFIER. 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

introducing a *block mutation operator*, which allows to change exponentially many values in a single mutation operation. The idea is to use an incomplete condition (i.e. a condition including just $v < n$ input variables) as parameter of the mutation, which we generalise below.

Definition 2: **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.

Fixing v of the n input variables induces a *partition* of the output vector into $b = \prod_{i \in V} |IS_i|$ blocks each covering N/b entries of the output vector (which has a total of N entries). There is a one-to-one correspondence between the set of all incomplete conditions CONDR made up of the fixed v variables and the set of all the blocks partitioning the output vector. The effect of mutation FBM on the output vector is that of selecting one block uniformly at random and forcing all the entries belonging to the block to OUTR. Table II shows an example of application of FBM.

GSGP with FBM is restricted to search the space of functions whose output vectors have the same output values for all entries of each block. This creates a difficulty: for some training sets of some classifier problems, the optimal function satisfying the training set is not reachable for some choice of the fixed variables, as it lies outside the search space. This happens when at least two training examples with different outputs belong to the same block (e.g., in Table II the values of the entries in the first block of the target output vector Y are 4, 2 and 1. This solution is therefore not reachable as reachable output vectors have all these entries set to the same value).

As in [18] we make the standard Machine Learning assumption that the training set T is sampled uniformly at random from the set of all input-output pairs of the Classification problem p at hand and we give a bound on the probability of success (i.e., probability that the search algorithm can reach the optimum with the training set T). Furthermore, when GSGP with FBM can reach the optimum, we are interested in an upper-bound w.r.t. all Classification problems (any p) and all problem instances (any T of p) of the expected runtime to reach the optimum.

Theorem 3: For any Classification problem p, given a training set T of size τ sampled uniformly at random on the set of all input-output pairs of p, GSGP with FBM with an arbitrarily fixed set of v variables finds the optimal classification function satisfying the training set T in time $O(R_o b \log b)$, with probability $pr \approx e^{-\frac{\tau^2}{2b}}$ for $b >> \tau$, where $b = \prod_{i \in V} |IS_i|$ is the number of blocks partitioning the output vector and R_o is the size of the output set.

Proof of Theorem 3: The proof of Theorem 3 is a simple generalisation of the proof of Theorem 3 in [18]. For the probability of success the only difference is that the number of blocks b in which the search space is partitioned is equal to $\prod_{i \in V} IS_i$ (instead of 2^v). However, when $b >> \tau$ the same reasoning applies: the asymptotic probability to find an optimal solution is $e^{-\frac{\tau^2}{2b}}$. For the runtime, again as in Theorem 3 in [18], searching for the optimum using GSGP with FBM is equivalent to solving OneMax on a string of length b (in this case over an alphabet of size R_o) using only forcing mutation. Hence the expected time to reach the optimum is $O(R_o b \log b)$. ■

As in [18] the number of blocks b partitioning the output vector is critical for the performance of the search. On one hand we need to keep b small to have a short runtime, on the other hand having many blocks increase the probability of success. The number of blocks b is an indirect parameter of the algorithm that can be chosen by fixing the number v of variables in the initial fixed set, as $b = \prod_{i \in V} |IS_i|$. The

theorem below shows how to fix v to have both a polynomial runtime on the number of variables n and high probability of success.

Theorem 4: GSGP with FBM on a problem with n variables and on an uniformly sampled training set of size τ bounded above by n^c, can find the optimum with high probability in polynomial time provided that the number v is fixed as $v = \lceil (2c+1) \log_{r_m}(n) \rceil$, where r_m is the minimum size of the input alphabet across all variables.

Proof: The number b of blocks in which the possible inputs are partitioned is at least r_m^v, hence, since $v = \lceil (2c+1) \log_{r_m}(n) \rceil$, we have $b \geq r_m^v > n^{2c}$ (i.e., $b \geq n^{2+\varepsilon}$ for some $\varepsilon > 0$). Therefore Theorem 3 is applicable to obtain a success probability that is, asymptotically, at least $e^{-\frac{n^{2c}}{2n^{2c+\varepsilon}}} = e^{-\frac{1}{2}n^{-\epsilon}}$. As the argument of the exponent approaches 0 as n grows, we can use the expansion $e^x = 1 + x$ obtaining $p = 1 - \frac{1}{2}n^{-\epsilon}$, which tells us that the runtime holds with high probability for any $\epsilon > 0$.

Recall that the number of blocks b is bounded above by r_M^v, where r_M is the maximum size of the input alphabet across all variables. Then $b \leq r_M^v \leq r_M^{(2c+1)\ \log_{r_m}(n)+1} \leq r_M n^{(2c+1)\log_{r_m}(r_M)}$. Hence, by Theorem 3, the optimization time is $O(R_o b \log b) = O(R_o r_M n^{(2c+1)\log_{r_m}(r_M)} \log n)$, which, assuming the size of the input sets not depending from n, is polynomial in n.

■

B. Space-Efficient Mutations for Classification Trees

GSGP with the FBM operator with an adequate parameter setting can find an optimal classifier, given a polynomial size training set, in polynomial time with high probability. Nevertheless, when classifiers are represented using classification trees, the classifier evolved with FBM grows exponentially in the number of generations. In the following, we introduce two mutation operators that have polynomial runtime and that lead to a polynomial growth of the evolving classifier by exploiting the specific structure of classification trees.

We know that given a complete classification table it is possible to build a classifier tree having height equal to the number of input variable. This is achieved assigning to each level of the tree an input variable to check. Nevertheless the number of nodes of the tree is exponential on the height of the tree and is thus exponential on the problem size. But since we want a classification tree representing a polynomial training set (and thus not the whole classification table) we might obtain a classification tree with a polynomial number of nodes. Particularly using random condition containing just a fixed logarithmic-size set of input variables as stated in Theorem 4 it must be possible to build a classification tree of logarithmic height and thus polynomial number of nodes in the problem size n. The following improved version of FBM forces the tree to assign each level to a particular input variable, obtaining a minimal size tree as explained above.

Definition 3: **Improved Fixed Block Mutation (IFBM)** Let be $V = (X_1, \cdots, X_v)$ a subset of the set of all the input variables containing $v < n$ input variables that is fixed and ordered in some arbitrary way at the initialisation of the algorithm. Given a parent program `P`, the mutation IFBM (Improved Fixed Block Mutation) generates a random incomplete condition `CONDR` comprising all fixed variables as a base for the forcing mutation and `OUTR`, a random output symbol. Then it tries to explore the tree to match `CONDR` without using the else conditions. One of the following can happen:

1) It successfully match `CONDR` ending up on a leaf node. Then it substitute the node with a leaf containing `OUTR` (Fig. 2, case 1)
2) It partially match some conditions ($X_1 = i_1, \cdots, X_p = i_p$) but in ends up on a internal node containing the input variable X_{p+1} without the possibility to match the condition $X_{p+1} = i_{p+1}$ because a subtree for that condition is missing. Then it adds a child to that node, with an edge representing the condition $X_{p+1} = i_{p+1}$, and containing the subtree `ST = IF (`$X_{p+2} = i_{p+2}$ `AND` $\cdots$ `AND` $X_v = i_v$`) THEN OUTR ELSE OUT` (Fig. 2, case 2).

Using this operator it is possible to obtain the same runtime of FBM but avoiding redundancy in the tree and keeping its size minimal. The tree is forced to keep an input variable for each level. Since FBM use a logarithmic number of variables (as suggested by Theorem 4) the evolved classification tree has logarithmic height and thus polynomial number of nodes in the problem size n.

The block mutation operators considered so far (i.e. FBM, IFBM) cannot guarantee to find the optimum in all cases, but when they do they may find it in polynomial time. Instead SGMP can always find the optimum as it can act on the value of any entry of the output vector independently from any other entry. However, it needs exponential time to find the optimum on any problem and any choice of training set. In the following, we introduce a mutation operator that combines both blockwise and pointwise mutations, which attempts to preserve the benefits of both.

Definition 4: **Multiple Size Block Mutation (MSBM)** Let $V = (x_1, \cdots, x_n)$ be randomly ordered list of all the input variables. Then at each iteration MSBM sample v uniformly at random from 1 to n, it builds a random condition `CONDR` using the variables $(x_1, \cdots, x_v)$ and it samples a random output symbol `OUTR`. Then it tries to explore the tree to match `CONDR` without using the else conditions. One of the following can happen:

1) It successfully match `CONDR` ending up on a leaf node. Then it substitute the node with a leaf containing `OUTR` (Fig. 2, case 1)
2) It partially match some conditions ($X_1 = i_1, \cdots, X_p = i_p$) but in ends up on a internal node containing the input variable X_{p+1} without the possibility to match the condition $X_{p+1} = i_{p+1}$ because a subtree for that condition is missing. Then it adds a child to that node, with an edge representing the condition $X_{p+1} = i_{p+1}$, and containing the subtree `ST = IF (`$X_{p+2} = i_{p+2}$ `AND` $\cdots$ `AND` $X_v = i_v$`) THEN OUTR ELSE OUT`. (Fig. 2, case 2)

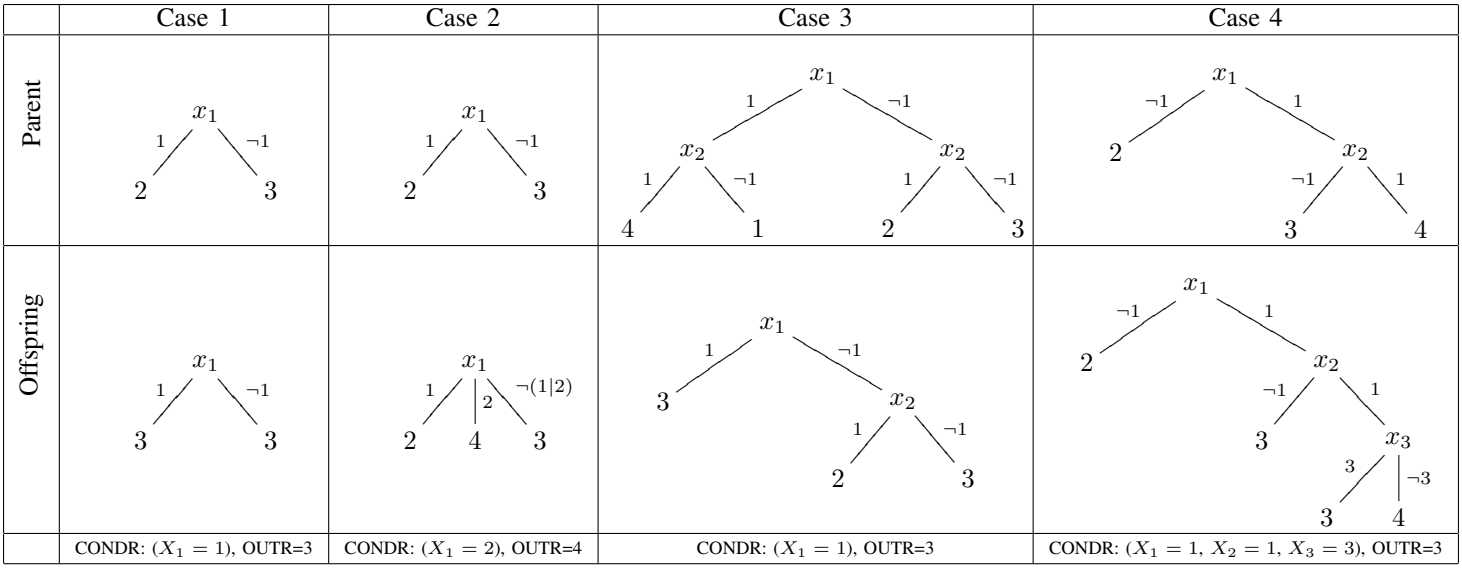

Fig. 2. Examples of mutation cases for IFBM (case 1-2) and MSBM (case 1-4)

3) It successfully match `CONDR` ending up on an internal node. Then it substitute the node with a leaf containing `OUTR` and delete all the subtrees of the old node. (Fig. 2, case 3)
4) It partially match some conditions ($X_1 = i_1, \cdots, X_p = i_p$) but in ends up on a leaf containing `OUT` without matching the renaming variables ($X_{p+1} = i_{p+1}, \cdots X_v = i_v$). Then it replaces the current leaf with the tree `ST = IF` ($X_{p+1} = i_{p+1}$ `AND` $\cdots$ `AND` $X_v = i_v$) `THEN OUTR ELSE OUT`. (Fig. 2, case 4)

The effect of this on the output vector is that, since the number of variables to produce the random condition `CONDR` can be between 1 and n, at each generation the size of the block is not fixed. On one hand, as for SGMP, this allows semantic GP with MSBM to always reach the optimum as each single entry of the output vector can be acted upon independently by the mutation. On the other hand, when the set of variable used by FBM is one of the subsets used by MSBM, then MSBM can solve efficiently any problem that can be solved efficiently with the block mutation FBM. This is because MSBM can simulate FBM on the set of v variables in common in time which is in the worst case n times larger (as the probability of selecting exactly v variables by MSBM is $1/n$).

Even if the classification tree evolved with MSBM use all n variables, the number of nodes is still kept polynomial, when the runtime is polynomial:

Theorem 5: Consider a classification problem with n variables. An individual of GSGP with MSBM mutation grows of $O(n)$ nodes at each generation.

Proof: Consider the possible cases described in Fig. 2. The increase in the number of nodes for each case is the following:

1) One node is replaced. The number of nodes remains the same;
2) At most n condition, each with two children, are added. Hence, the increase in the number of nodes is bounded by $O(n)$.
3) The number of nodes is reduced by at least 2;
4) By the same reasoning of case 2, the increase in the number of nodes if bounded by $O(n)$.

The four cases of Fig. 2 are exhaustive. That is, any tree transformation of MSBM is one of those cases. Therefore, the statement of the theorem follows immediately. ■

Corollary 1: If the tree is evolved in a polynomial number of generations, then the tree has polynomial size.

IV. Experiments

A. Experimental Settings

In order to investigate how the problem size affects the number of generations needed to find the global optimum we have performed an experimental comparison between the proposed mutation operators. We use GSGP with population size 1 and elitism, i.e. at each generation one offspring is produced by mutation and the fittest between the parent and the offspring is kept at the following generation. The other settings are as follows:

- The size of the input and output alphabets are fixed to 3 ($IS_i = OS = 3$);
- The problem sizes tested are $n = 9$, 18, 27, and 36. For any problem size 100 random training sets were generated each with n test cases ($\tau = n$);
- For IFBM the size of the block is fixed to $2\lceil \log_3 n \rceil + 1$;
- The algorithm is stopped when the optimum is found or when a limit of 10^6 generation is reached.

For each training set we run the algorithm with IFBM and MSBM (i.e., 100 runs with IFBM and 100 runs with MSBM for each problem size).

From the theoretical results in the previous sections we expect to always find the optimum when using MSBM (success

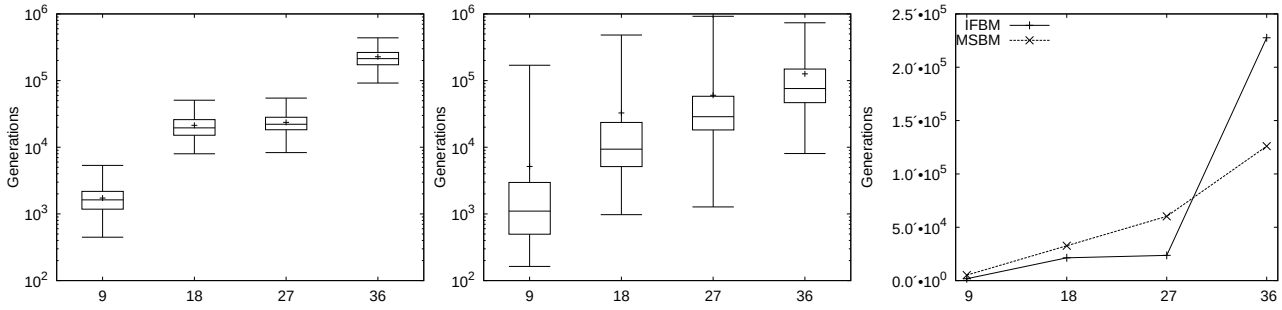

Fig. 3. From left to right: the box plot of the number of generations needed to reach the optimum using IFBM and MSBM respectively, and a comparison of the average number of generations needed to reach the optimum.

Problem size	Success rate (IFBM)	Success rate (MSBM
9	0.92	1.00
18	0.95	0.99
27	0.90	0.97
36	0.99	0.96

TABLE III. THE SUCCESS RATE FOR THE TESTED MUTATIONS AND PROBLEM SIZES.

rate equals to 1), while having a fraction of not solvable instances with IFBM (success rate lower than 1).

B. Experimental Results

In Table III the success rates for all tested mutations and problem sizes are presented. The results for IFBM and MSBM are summarized in the first two plots in Fig. 3. The box plots show, for different problems sizes, the generations in which the optimum was reached (only successful runs are presented). The boxes represents the 25-th and 75-th percentile. The whiskers show the lowest and highest generation in which the optimum was found, the bar represents the median, while the cross the average. In all box plots the problem size is on the x-axis. Finally, in the third plot of Fig. 3, a comparison of the average time needed to reach the optimum for IFBM and MSBM on different problem sizes is presented.

a) Success rate: as it is possible to observe, even if MSBM is supposed to reach the optimum with probability 1, not all the runs were able to obtain a perfect fit. However, the success rate decrease when the problem size gets higher, showing that the reason for a rate lower than 1 is probably the cap imposed on the number of generations. This result is in accordance with the theory, since MSBM is always able to find the optimum. Differently, for IFBM the success rate change abruptly from one problem size to the next, showing that a perfect solution was not found because for some instances the optimum was not reachable with IFBM, and not just because of the cap on the number of generations. This result was also expected from the theory, since for some pairs of problem instance and training set and for some choice of the fixed set variables V, IFBM cannot construct a tree which can correctly classify all the test instances.

b) Time to find the optimum: the comparison shows that IFBM is, for all problem sizes except 36, slightly faster than MSBM in finding a perfect fit. However, for the largest problem size tested, it is almost twice slower than MSBM. It is worth to notice that for each problem size the results have a smaller variance for IFBM than for MSBM. This behaviour might be due to the fact that MSBM has less constraints in the way it can build the classification tree: since the depth of the tree is not fixed, it is able to quickly find a short solutions that are optimal and, at the same time, it is also able to find more complex ones that IFBM cannot find. However this higher flexibility is paid with a large number of generations needed to find the optimum in some runs.

In conclusion, we can summarize the features of the two mutations as follows:

- IFBM, while not able to always find the optimal solution, has more predictable runtime than MSBM.
- MSBM can always find the optimum, but its runtime is more difficult to predict, with some runs unable to find the optimum before the imposed limit of 10^6 generations.

It is interesting to notice that, in the situations when both IFBM and MSBM can find the optimum, the theory says that the expected runtime of MSBM is up to a factor of n greater than the one of IFBM. However, the experimental results shows that in practice this difference is not always visible. A possible explanation is that the slowdown of n happens only in some limited cases or that the runtime results are not tight.

V. CONCLUSIONS AND FUTURE WORK

In this paper Geometric Semantic GP (GSGP) applied to classification problems was studied. This kind of GP has the peculiar property that its syntactic operators are equivalent, in the semantic space (i.e., on the output vectors), to the operators of a GA solving a "sparse" version of OneMax in which only a limited number of entries contribute to the fitness. Therefore GSGP always see a conic fitness landscape.

We have considered the situation in which the training set has polynomial size in the number of input variables of the target classifier. That is, logarithmic in the size of the search space. In this setting we have proposed and analysed two efficient geometric semantic mutation operators. IFBM can find the optimum in expected polynomial time, but in some cases it cannot find it (even if it does it with high probability on a random pair of problem instance and training set). MSBM, on the other hand, is always able to find the optimum even if, for some combinations of training sets and problem instances, it may require exponential time.

There are many possibilities for future investigations. For MSBM we have provided a runtime analysis that gives a

polynomial time bound that holds for a large fraction of all the possible instances of the problem. However, it would be good to have a more precise analysis able to characterize exactly which problem instances are difficult to learn. That is, what are the cases in which IFBM cannot find the optimum and MSBM needs exponential time to reach it.

An important issue to study in the future is the generalization ability of GSGP. Every operator impose a bias on the resulting classification tree. For example, IFBM generates trees of fixed length while MSBM has more flexibility in defining the structure of the classification tree. This obviously impacts the classification on unseen inputs. It would be interesting to characterize class of problems in which GSGP can generalize well. We would also like to compare GSGP against traditional GP on a wide range of classification benchmarks. Finally, one of our goal is to expand the study of GSGP on all commonly used domains.

REFERENCES

[1] L. Beadle and C. G. Johnson, "Semantic analysis of program initialisation in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 10, no. 3, pp. 307–337, 2009.

[2] D. Jackson, "Phenotypic diversity in initial genetic programming populations," in *Proc. of EuroGP 2010*, 2010, pp. 98–109.

[3] L. Beadle and C. G. Johnson, "Sematically driven crossover in genetic programming," in *Proc. of IEEE WCCI '08*, 2008, pp. 111–116.

[4] N. Q. Uy, N. X. Hoai, M. O'Neill, R. McKay, and E. Galván-López, "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, 2011.

[5] K. Krawiec and P. Lichocki, "Approximating geometric crossover in semantic space," in *Proc. of GECCO '09*, 2009, pp. 987–994. [Online]. Available: http://doi.acm.org/10.1145/1569901.1570036

[6] K. Krawiec and B. Wieloch, "Analysis of semantic modularity for genetic programming," *Foundations of Computing and Decision Sciences*, vol. 34, no. 4, pp. 265–285, 2009. [Online]. Available: http://fcds.cs.put.poznan.pl/FCDS2/ArticleDetails.aspx?articleId=219

[7] A. Moraglio, K. Krawiec, and C. Johnson, "Geometric semantic genetic programming," in *Proceedings of Parallel Problem Solving from Nature*, 2012.

[8] ——, "Geometric semantic genetic programming," in *Workshop on Theory of Randomized Search Heuristics*, 2011.

[9] A. Moraglio and R. Poli, "Topological interpretation of crossover," in *Proc. of GECCO '04*, 2004, pp. 1377–1388.

[10] A. Moraglio, "Towards a Geometric Unification of Evolutionary Algorithms," Ph.D. dissertation, University of Essex, 2007.

[11] M. Castelli, L. Manzoni, and L. Vanneschi, "An efficient genetic programming system with geometric semantic operators and its application to human oral bioavailability prediction," *arXiv:cs.NE/1208.2437v1*, 2012.

[12] W. Langdon and R. Poli, *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[13] B. Mitavskiy and J. Rowe, "Some results about the markov chains associated to GPs and to general EAs," *Theoretical Computer Science*, vol. 361(1), pp. 72–110, 2006.

[14] R. Poli and N. F. McPhee, "Parsimony pressure made easy: Solving the problem of bloat in gp," in *Theory and Principled Methods for Designing Metaheuristics*, Y. Borenstein and A. Moraglio, Eds. Springer, 2012, ch. 9.

[15] R. Poli, M. Graff, and N. McPhee, "Free lunches for function and program induction," in *Workshop on Foundations of Genetic Algorithms*, 2009.

[16] N. F. McPhee, B. Ohs, and T. Hutchison, "Semantic building blocks in genetic programming," in *European Conference on Genetic Programming*, 2008.

[17] G. Durrett, F. Neumann, and U.-M. O'Reilly, "Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics," in *Workshop on Foundations of Genetic Algorithms*, 2011.

[18] A. Moraglio, A. Mambrini, and L. Manzoni, "Runtime analysis of mutation-based geometric semantic genetic programming on boolean functions," in *Foundations of Genetic Algorithms*, 2013, (to appear).