Runtime Analysis of Mutation-Based Geometric Semantic Genetic Programming on Boolean Functions

Alberto Moraglio & Andrea Mambrini
CERCIA, University of Birmingham
Birmingham B15 2TT, UK
a.moraglio@cs.bham.ac.uk
a.mambrini@cs.bham.ac.uk

Luca Manzoni
DISCo, University of Milano-Bicocca
20126 Milano, Italy
luca.manzoni@disco.unimib.it

ABSTRACT
Geometric Semantic Genetic Programming (GSGP) is a recently introduced form of Genetic Programming (GP), rooted in a geometric theory of representations, that searches *directly* the semantic space of functions/programs, rather than the space of their syntactic representations (e.g., trees) as in traditional GP. Remarkably, the fitness landscape seen by GSGP is *always* – for any domain and for any problem – unimodal with a linear slope by construction. This has two important consequences: (i) it makes the search for the optimum much easier than for traditional GP; (ii) it opens the way to analyse theoretically in a easy manner the optimisation time of GSGP in a *general setting.* The runtime analysis of GP has been very hard to tackle, and only simplified forms of GP on specific, unrealistic problems have been studied so far. We present a runtime analysis of GSGP with various types of mutations on the class of *all* Boolean functions.

1. INTRODUCTION
Traditional Genetic Programming searches the space of functions/programs by using search operators that manipulate their syntactic representation, regardless of their actual semantics/behaviour. For instance, subtree swap crossover is used to recombine functions represented as parse trees, regardless of trees representing Boolean expressions, mathematical functions, or computer programs. Although this guarantees that offspring are always syntactically well-formed, it is unclear why such a blind syntactic search should work well for different problems and across domains.

In recent literature, there are a number of approaches that use the semantics of programs in various ways to guide the search of GP. Beadle & Johnson and Jackson use reduction to a canonical representation, in which individuals encoding the same function have a unique representation, to enforce semantic diversity throughout evolution, by creating semantically unique individuals in the initial population [3, 6], and by discarding offspring of crossover and mutation when semantically coinciding with their parents [2].

FOGA'13, January 16–20, 2013, Adelaide, Australia.

The semantics of a program can be directly and uniquely represented by enumerating the input-output pairs making up the computed function, or equivalently, by the vector of all output values of the program for a certain fixed order of all possible input values. Uy et al. [21] have proposed a probabilistic measure of semantic distance between individuals based on how their outputs differ for the same set of inputs sampled at random. This distance is then used to bias semantically the search operators: mutation rejects offspring that are not sufficiently semantically similar to the parent; crossover chooses only semantically similar subtrees to swap between parents.

Geometric crossover and geometric mutation [17, 13] are formal, representation-independent search operators that can be, in principle, instantiated to any search space and representation, once a notion of distance between individuals is provided. Simply stated, the offspring of geometric crossover are in the space-specific segment between parents, and the offspring of geometric mutation are in a space-specific ball around the parent. Many crossover and mutation operators across representations are geometric operators (w.r.t. some distance). Krawiec et al. [7, 8] have used a notion of semantic distance to propose a crossover operator for GP trees that is approximately a geometric crossover in the semantic space (i.e., a semantic geometric crossover). The operator was implemented approximately by using the traditional sub-tree swap crossover, generating a large number of offspring, and accepting only those offspring that were sufficiently "semantically intermediate" with respect to the parents. An analogous approach can be used to implement a semantic geometric mutation, with offspring lying in a small ball around the parent in the semantic space.

Whereas the semantically aware methods above are promising, as they have been shown to be better than traditional GP on a number of benchmark problems [3, 6, 21, 7], 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. This drawback seems unavoidable. It was in fact believed [7, 8] that due to the complexity of the genotype-phenotype mapping in GP, a *direct* implementation of semantic operators that, acting on the syntax of the parent programs, produce offspring that are *guaranteed* to respect some semantic criterion/specification by construction, is probably impossible.

Geometric Semantic Genetic Programming [16, 15] is a form of genetic programming that uses semantic geometric crossover and semantic geometric mutation to search *directly* the semantic space of functions/programs. This is possible because, seen from a geometric viewpoint, the genotype-phenotype mapping of GP becomes *surprisingly easy*, and allows us to derive explicit algorithmic characterization of semantic geometric operators for different domains following a *simple formal recipe*, 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 its semantic distance to the optimum individual. This has the consequence that GP search on functions with semantic geometric operators is formally *equivalent* to a GA search on the corresponding output vectors with standard crossover and mutation operators. For example, for Boolean functions, GSGP search is equivalent to GA search on binary strings on the OneMax landscape, for *any* Boolean problem.

This equivalence 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 [16, 15]. Furthermore, GSGP has been found more efficient and generalising better than standard GP on some initial studies on real-world problems [4].

Genetic programming has been hard to analyse from a theoretical point of view. The current literature on GP theory is heterogeneous. Perhaps the most developed theory of GP is the schema theory [9]. There is also some work on Markov models of GP search [12]. There are theory-laden methods to combat bloat based on an exact formalisation of the dynamics of average program size [20]. Other works focus on the analysis of some static structural features of the search space of GP programs (e.g., proportions of programs encoding the same function for different program sizes), and experimental hardness studies of fitness landscapes [9]. There is also some theoretical works on GP from a semantic perspective. In [19], a notion of geometric mutation based on a semantic distance was used to show that the No Free Lunch theorem does not apply to GP. Furthermore, the work [11] analyses traditional subtree crossover in terms of "semantic building blocks" in Boolean functions, reporting that most of the times this crossover does not make useful search in the semantic space.

Runtime analysis is the standard approach to analyse analytically algorithmic performance. In the last decade it has been applied, with an ever increasing success, to randomised search heuristics and it is establishing itself as a leading theory [18, 1]. Despite its success, the analysis is done on a per-algorithm-and-per-problem basis. Obtaining interesting, general runtime results holding on a large class of problems for non-trivial search algorithms would be a major progress. Due to the difficulty of analysing GP, there is only very initial work on its runtime analysis. Durrett et al. [5] present the runtime analysis of a mutation-based GP with a tree representation on very simplified problems, in which trees do not represent functions (i.e., objects that return different output values for different input values) but, rather, structures (i.e., objects whose fitness depends on some structural properties of the tree representation). This deviates quite significantly from the very essence of GP, which is about evolving functions.

GSGP is very attractive from a theoretical point of view. The equivalence of GSGP search to a GA search on cone landscapes opens the way to a rigorous theoretical analysis of the optimisation time of GSGP by simply extending known runtime results for GAs on OneMax-like problems. This analysis is not only relatively easy to obtain but it is also remarkably general, as it applies to all problems of a certain domain (e.g., all Boolean functions are seen as OneMax by GSGP). Furthermore, unlike existing runtime analysis for GP, the solutions of the problem considered are functions and not structures. Therefore, there is the potential to develop a general runtime analysis of GSGP on interesting problems. We start this line of theory and present a runtime analysis of GSGP with various types of mutations on the class of all Boolean functions.

Section 2 describes the theory of geometric semantic genetic programming framework. Section 3 introduces a number of mutation operators, and analyses their runtime. Section 4 reports experimental results comparing the various types of mutation on randomly generated problems. Section 5 presents conclusions and future work.

2. GEOMETRIC SEMANTIC GENETIC PROGRAMMING

Next, we describe the GSGP framework formally reporting and expanding on the relevant results from [16]. We first give abstract definitions of geometric semantic operators and their properties. Then we explain how to construct these operators in Section 2.2.

2.1 Abstract Geometric Semantic Search

A search operator $CX : S \times S \to S$ 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 $M : S \to S$ 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 $f : S \to \mathbb{R}$, the geometric search operators induce or see the fitness landscape identified by the triple (f, S, d). Many well-known recombination operators across representations are geometric crossovers [13]. In particular for binary strings, any type of homologous crossover is a geometric crossover w. r. t. Hamming distance (HD), and point mutation is geometric 1-mutation w. r. t. Hamming distance [13]. Geometric operators can also be derived for new spaces and representations by using in their definitions a distance based on a target representation (e.g., edit distance). If the distance between solutions is not directly linked to their representation, the geometric operators are well-defined in an abstract sense but their algorithmic description may be hard to derive.

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 $h : X \to Y$ 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 $F_T : \mathrm{H} \to \mathbb{R}$ 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 $P : \mathrm{H} \to Y^{|X|}$ 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: $P_{X'} : \mathrm{H} \to Y^{|X'|}$ 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 Boolean functions, the input and output spaces are $X = \{0, 1\}^n$ and $Y = \{0, 1\}$, and the output vector is a binary vector of size $N = 2^n$ (i.e., Y^N). A suitable metric D to measure the error as a distance between binary vectors is the Hamming distance.

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. As P is generally non-injective (i.e., different genotypes may have the same phenotype), SD is only a pseudometric (i.e., distinct functions can have distance zero). This naturally induces an equivalence relation on genotypes. Genotypes h_1 and h_2 belong to the same semantic class $\overline{h}$ iff their semantic distance is zero, i.e., $h_1, h_2 \in \overline{h}$ iff $SD(h_1, h_2) = 0$. Therefore the set of all genotypes H can be partitioned in equivalence classes, each one containing all genotypes in H with the same semantics. Let $\overline{H}$ be the set of all semantic classes of genotypes of H. The set of semantic classes $\overline{H}$ is by construction in one-to-one correspondence with the set of phenotypes (i.e., output vectors). Then, as D is a metric on the set of phenotypes, it is naturally inherited as a metric on the set $\overline{H}$ of semantic classes.

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 Boolean functions returns offspring Boolean functions such that the output vectors of the offspring are in the Hamming segment between the output vectors of the parents (w. r .t. all $x_i \in X$). I.e., any offspring function $h_3 = SGX(h_1, h_2)$ of parent functions h_1 and h_2 meets the condition $SD(h_1, h_3) + SD(h_2, h_3) = SD(h_1, h_2)$ which for the specific case of Boolean functions becomes $HD(P(h_1), P(h_3)) + HD(P(h_2), P(h_3)) = HD(P(h_1), P(h_2))$. The geometric crossover SGX can be also seen as a geometric crossover on the space of semantic classes of functions $\overline{H}$ endowed with the metric D. From the definition of SGX above, it is evident that if $h_3 = SGX(h_1, h_2)$ then it holds that $h'_3 = SGX(h'_1, h'_2)$ for any $h'_1 \in \overline{h_1}, h'_2 \in \overline{h_2}, h'_3 \in \overline{h_3}$ because $P(h_1) = P(h'_1), P(h_2) = P(h'_2)$ and $P(h_3) = P(h'_3)$. In words, the result of the application of SGX depends only on the semantic classes of the parents $\overline{h_1}$, $\overline{h_2}$ and not directly on the parents' genotypes h_1, h_2, and the returned offspring can be any genotype h_3 belonging to the offspring semantic class $\overline{h_3}$. Therefore, SGX can be thought as searching directly the semantic space of functions.

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 particularly nice type of unimodal landscape in which the fitness of a solution is its distance in the search space to the optimum [1] (i.e., a *cone landscape*). This observation is *remarkably general*, as it holds for any domain of application of GP (e.g., Boolean, Arithmetic, Program), any specific problem within a domain (e.g., Parity and Multiplexer problems in the Boolean domain) and for any choice of metric for the error function. Furthermore, there is some formal evidence [14] that EAs with geometric operators can optimise cone landscapes efficiently very generally for most metric. 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". The various forms of mutations that will be introduced and analysed in Section 3

[1]Notice that the optimum in the space of function classes is unique, as the output target vector is unique (since the training set is fixed prior to evolution), and it identifies uniquely the target function class.

have different strategies to cope with this form of neutrality and produce an efficient search.

GP search with geometric operators w. r .t. the semantic distance SD on the space of functions H 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 are isomorphic to geometric operators on output vectors, as SD is induced from D via the genotype-phenotype mapping P (see also diagram (1) and explanation in the next section). Despite this formal equivalence, actually encoding a function in a EA using its output vector instead of, say, a parse tree, is futile: in the end we want to find a function represented in an *intensive form* that can represent concisely "interesting" functions and that allows for meaningful generalisation of the training set.

For the specific case of Boolean functions with n input variables and a single output variable, GSGP search with a training set of size N is equivalent to GA search with standard mutation and crossover on binary strings of length 2^n (i.e., the number of all possible inputs of n Boolean variables). When the training set covers all possible inputs, the fitness landscape seen by the GA is OneMax because minimising the error means minimising the Hamming distance between the output vector of candidate solutions and the target output vector, which is the same as minimising the number of wrong outputs, or equivalently as maximising the number of the right outputs, which on binary strings is equivalent to maximising the number of ones. When the training set covers only a subset of all possible inputs, the fitness landscape seen by the GA is OneMax on τ "active" bits that contribute to the fitness, and it is neutral on the remaining bits that do not affect the fitness. The position of the active bits are fixed but unknown, as we operate under the black-box scenario in which the algorithm cannot have direct knowledge of the training set but can only access the errors on the training set of candidate solutions via evaluation. Furthermore, *all Boolean functions are seen as equivalent* from GSGP search. This is because, whereas any distinct target training set gives rise to a different fitness landscape whose optimum is a different target string, any unbiased black-box search algorithm [10] does not assume a priori the knowledge of the location of the optimum and sees all these landscapes as equivalent.

2.2 Construction of Geometric 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 $P(T3) = GX_D(P(T1), P(T2))$.

$$\begin{tikzcd} T1 \arrow[d, "P"] & \times & T2 \arrow[d, "P"] \arrow[r, "GX_{SD}"] & T3 \arrow[d, "P"] \\ O1 & \times & O2 \arrow[r, "GX_D"] & O3 \end{tikzcd} \qquad (1)$$

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 Boolean functions with fitness measure based on Hamming distance, output vectors are binary strings and GX_D is a mask-based crossover. We want to derive a recombination operator acting on Boolean functions that corresponds to a mask-based crossover on their output vectors. Note that there is a different type of semantic geometric crossover for each choice of space H and distance D. Consequently, there are different semantic crossovers for different GP domains. Furthermore, note that as the semantic crossover works directly on the semantic space of functions, *it does not matter how functions are actually represented* as the representation does not affect the search behaviour. For the sake of contrasting this framework with traditional GP, we will represent functions as trees.

DEFINITION 1. ***Boolean semantic crossover:*** *Given two parent functions $T1, T2 : \{0,1\}^n \to \{0,1\}$, the recombination SGXB returns the offspring Boolean function $T3 = (T1 \wedge TR) \vee (\overline{TR} \wedge T2)$ where TR is a randomly generated Boolean function (see Fig. 1).*

DEFINITION 2. ***Boolean semantic mutation:*** *Given a parent function $T : \{0,1\}^n \to \{0,1\}$, the mutation SGMB returns the offspring Boolean function $TM = T \vee M$ with probability 0.5 and $TM = T \wedge \overline{M}$ with probability 0.5 where M is a random minterm of all input variables.*

THEOREM 1. *SGXB is a semantic geometric crossover for the space of Boolean functions with fitness function based on Hamming distance, for any training set and any Boolean problem. SGMB is semantic 1-geometric mutations for Boolean functions with fitness function based on Hamming distance.*

The proof of the previous theorem can be found in [16]. In the following, we give an example to illustrate the theorem for the crossover. Let us consider the 3-parity problem, in which, we want to find a Boolean function $F(X_1, X_2, X_3)$ that returns 1 when an odd number of input variables is 1, 0 otherwise. Its truth table is in Table 1 (first 4 columns). As we have 3 input variables, there are 2^3 possible input combinations (first 3 columns of Table 1). In this example, we consider the training set to be made of all 8 entries of the truth table. However, normally the training set comprises only a small subset of all input-output pairs. The target output vector Y is the binary string 01101001 (column 4 of Table 1). For each tree representing a Boolean function, one can obtain its output vector by querying the tree with

```
         AND                   Crossover Scheme               Offspring
T1 =   /   \
      X1   X2                        OR                           OR
                                   /    \                        /   \
          OR                     AND      AND                  AND    X3
T2 =   /  \            T3 =    /   \    /   \         =       /   \
      X2  X3                  T1   TR  NOT   T2             AND   NOT
                                        |                  /  \    |
         NOT                            TR                X1   X2  X3
TR =      |
         X3
```

Figure 1: T1 and T2 are parent functions and TR is a random function. The offspring T3 is obtained by substituting T1, T2 and TR in the crossover scheme and simplifying algebraically.

Table 1: Truth table of 3-parity problem (first 4 columns). Output vectors of trees in Figure 1 (last 4 columns): of parents T1 and T2, of random mask TR, and of offspring T3.

X_1	X_2	X_3	Y	$P(T1)$	$P(T2)$	$P(TR)$	$P(T3)$
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

all possible input combinations. The output vectors of the trees in Figure 1 are in the last 4 columns of Table 1. The fitness $f(T)$ of a tree T (to minimise) is the Hamming distance between its output vector $P(T)$ and the target output vector Y (restricted to the outputs of the training set), e.g., the fitness of parent $T1$ is $f(T1) = HD(P(T1), Y) = HD(00000011, 01101001) = 4$. The semantic distance between two trees $T1$ and $T2$ is the Hamming distance between their output vectors $P(T1)$ and $P(T2)$, e.g., the semantic distance between parent trees $T1$ and $T2$ is $SD(T1, T2) = HD(P(T1), P(T2)) = HD(00000011, 01110111) = 4$. Let us now consider the relations between the output vectors of the trees in Table 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 of $T3$ (in $P(TR)$, a 1 indicates that $P(T3)$ gets a bit from $P(T1)$ for that position, and 0 that the bit to $P(T3)$ is from $P(T1)$). This crossover on output vectors is a geometric crossover w.r.t. Hamming distance, as $P(T3)$ is in the Hamming segment between $P(T1)$ and $P(T2)$ (i.e., it holds that $HD(P(T1), P(T3)) + HD(P(T3), P(T2)) = HD(P(T1), P(T2))$), as we can verify on the example:
$HD(00000011, 01010111) + HD(01010111, 01110111) = 3 + 1 = 4 = HD(00000011, 01110111)$. This shows that the crossover on trees in Figure 1 is a semantic geometric crossover w.r.t. Hamming distance.

Intuitively, the reason the theorem holds in general is that the crossover scheme in Figure 1 describes, using the language of Boolean functions, the selecting action of the recombination mask bit on the corresponding bits in the parents to determine the bit to assign to the offspring (i.e., it is a 1-bit multiplexer function piloted by the mask bit).

As the syntax of the offspring of semantic operators contain at least one parent, the size of individuals grows quickly in the number of generations. To keep their size manageable during evolution, we need to *simplify* offspring sufficiently and efficiently (not optimally, as that is NP-Hard on many domains) *without changing the computed function.* The search of semantic crossover and semantic mutation is completely unaffected by syntactic simplification, which can then be done at any moment and in any amount. For Boolean functions, there are quick function-preserving simplifiers. For the other domains, computer algebra system and formal methods can be used for these purpose.

3. RUNTIME ANALYSIS OF SEMANTIC MUTATIONS

We consider two families of semantic mutations, pointwise and blockwise mutations. They differ on the level of granularity of the changes to the output vector they can make. Pointwise mutations can change single entries independently. Blockwise mutations make the same change to entire blocks of entries.

3.1 Pointwise Mutations

We design semantic mutations that correspond to traditional mutations on the output vector. This will highlight a number of fundamental issues of semantic mutation and allow us to identify the requirements a semantic mutation should meet to be efficient.

Let us first consider the case in which the training set encompasses all possible input-output pairs of a Boolean function with n input variables. The size of the output vectors is $N = 2^n$, which is also the number of examples in the training set. A natural definition of *problem size* is the number of input variables n of the functions in the search space. Let us also assume that we have a GSGP (e.g., searching the space of functions represented as trees) that is exactly equivalent to a (1+1) EA (with mutation probability $1/N$) or to a RLS on the space of output vectors. The runtime of GSGP would be the same as of (1+1) EA or RLS on OneMax on the space of output vectors, which is $O(NlogN) = O(n2^n)$. This highlights a first issue: although the fitness landscape seen by GSGP is OneMax, since the size of the output vectors N is exponentially long in the problem size n, the runtime of GSGP is exponential.

A second issue is that with an exponential size of the training set in n, each single fitness evaluation takes exponential time as it requires to evaluate all the outputs of a function against the target vector on exponentially many fitness cases. Naturally, in practice, the training set encompasses only a small fraction of all the input-output pairs of a function. To be able to evaluate the fitness of a function in polynomial time the size of the training set needs to be polynomial in n. In the following, we will consider the size of the training set (τ) to be linear in n (it is easy to extend the analysis to polynomial size of the training set). This transforms the problem seen by the EA on output vectors into a "sparse" version of OneMax in which most of vector entries are neutral, and the remaining entries, whose locations in the output vector are unknown, give each a unitary contribution to the fitness. Even with a training set of polynomial size, it is easy to see that both RLS and (1+1) EA (with mutation probability $1/N$) take exponential time to find the optimum. This is because at each generation the probability of mutating a non-neutral position of the output vector is exponentially small, hence it takes exponential time to get an improvement. What would it be then a mutation operator that gives rise to a polynomial runtime on the sparse OneMax problem?

3.1.1 Initialisation operator

Before attempting answering this question, let us consider another issue with semantic mutation. Can we *actually* implement semantic mutation operators corresponding to (1+1) EA and RLS *efficiently*? Even implementing the initialisation operator that generates a function uniformly at random in the sematic space takes exponential space and time most of the time! This is because it is equivalent to sampling its output vector uniformly at random, which is a random binary string exponentially long in n. From Kolmogorov complexity theory, we know that only a logarithmically small fraction of random binary strings can be compressed. So, most of the initial random functions are exponentially long and take exponential time to generate. Fortunately, the problem is easily solved by starting from an arbitrary initial solution which admits a short representation (e.g., the True function) rather than from a random one. This does not have a significant impact on the runtime, and on the runtime analysis that will be done w.r.t. the worst-case initial solution.

3.1.2 Point mutations

The original semantic mutation for Boolean functions [16] can be implemented efficiently, and it is reported below.

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.*

Seen it on the output vectors, this operator forces to a random value exactly one randomly selected entry of the parent output vector. This is because adding ($\vee$) a minterm to a Boolean expression has the effect of forcing the corresponding single entry in the truth table to 1, and multiplying it ($\wedge$) by a negated minterm forces the corresponding entry to 0. This operator can be also rephrased as: it flips a randomly selected entry with probability 0.5 and it does not change anything with probability 0.5. So, it can be looked at as a semi-point mutation. Like point mutation, the runtime of GSGP with this operator is exponential.

Table 2: First three columns from left: truth table of 2-parity problem with inputs X_1 and X_2, and output Y. Three rightmost columns: output vectors of the random minterm M, of the parent P and of the produced offspring O obtained by applying the bit-flip point mutation.

X_1	X_2	Y	M	P	O
1	1	0	**1**	**1**	**0**
1	0	1	0	1	1
0	1	1	0	1	1
0	0	0	0	0	0

The semantic mutation below (bit-flip point mutation) induces exactly point mutation on output vectors. This mutation can be seen as crossover of the parent with the negation of itself with a crossover mask which selects all bits but one from the parent.

DEFINITION 4. ***Bit-flip point mutation:*** *Given a parent function $\mathcal{X} : \{0,1\}^n \to \{0,1\}$ the mutation returns the offspring Boolean function $\mathcal{X}' = (\mathcal{X} \wedge \overline{M}) \vee (M \wedge \overline{\mathcal{X}})$, where M is a random minterm of all input variables.*

Seen it on the output vectors, this operator flips the output of the parent function corresponding to the combination of the input variables that makes the random minterm M true. Let us illustrate this mutation with an example. Let us consider the 2-parity problem, so $n = 2$ input variables. Its truth table is in Table 2, in the first three columns from left. Let us consider the following application of the bit-flip point mutation operator:

Random minterm:	M $= X_1 \wedge X_2$
Parent:	P $= (X_1 \wedge X_2) \vee (X_1 \vee X_2)$
Offspring:	$O = (P \wedge \overline{M}) \vee (\overline{P} \wedge M)$

The three rightmost columns of Table 2 report the corresponding output vector view of the application of the bit-flip point mutation above. Note that the output vector of the offspring is obtained by flipping the bit of the output vector of the parent corresponding to the '1' in the output vector of the random minterm (boldface in Table 2).

3.1.3 Bitwise mutation

Let us now consider bitwise mutation, which flips each bit independently with a certain probability. Before considering a semantic mutation that induces bitwise mutation on the output vectors, we show that bitwise mutation with an adequate mutation probability can lead to a polynomial runtime on the sparse OneMax with exponentially long chromosome.

THEOREM 2. *On the sparse OneMax problem with an exponentially long chromosome with $N = 2^n$ entries and with $\tau < N$ non-neutral entries, (1+1) EA with bitwise mutation with $p = \Omega(1/\tau)$ finds the optimum in time $O(\tau \log \tau)$. In particular, when τ is polynomial in n, the runtime is polynomial.*

PROOF. We define the potential k as the number of incorrect bits. At each iteration the potential decreases if an incorrect bit is flipped. This happens with probability

$$p_k > k\frac{1}{\tau}\left(1-\frac{1}{\tau}\right)^{N-1} > k\frac{1}{\tau}\left(1-\frac{1}{N}\right)^{N-1} > \frac{k}{e\tau}$$

Since the potential can decrease at most τ times and the expected time for the potential to decrease is $1/p_k$, the expected time to reach $k = 0$ (an finding the solution to the problem is)

$$E(T) = \sum_{i=0}^{\tau} 1/p_k = \sum_{i=0}^{\tau} \frac{e\tau}{i} = O(\tau \log \tau)$$

□

How can we implement a semantic mutation that corresponds to bitwise mutation with mutation probability p on the output vectors?

DEFINITION 5. ***Bitwise mutation:*** *Given a parent function $\mathcal{X} : \{0,1\}^n \to \{0,1\}$ the mutation do the following:*

- *Sample an integer number x from $x \sim Bin(p, N)$*
- *Generate x minterms uniformly at random without repetitions $\{M_1, \cdots, M_x\}$*
- *The offspring is $\mathcal{X}' = (\mathcal{X} \wedge \overline{M}) \vee (M \wedge \overline{\mathcal{X}})$, where $M = M_1 \vee \cdots \vee M_x$.*

Unfortunately, also this operator has a problem. Using a probability of mutation $p = \Omega(\frac{1}{\tau})$, which by theorem 2 makes the runtime of GSGP efficient for a training set size τ polynomial, this implementation becomes exponential in space hence in time because the expected number of minterms making up M is pN, which is exponential in n. So, by changing the probability p, what it is gained in terms of runtime of GSGP, it is then lost in terms of efficiency of the implementation of a single application of the mutation operator, and vice versa. The challenge is therefore finding a semantic mutation operator that, at the same time, (i) can be implemented efficiently and (ii) makes the runtime of GSGP polynomial in n for any Boolean problem. Does such operator even exist?

3.2 Blockwise Mutations

In this section, we consider four semantic mutations, which extend the forcing point mutation introduced in the previous section in different ways. The extension is obtained by replacing the mutating random minterm M with an *incomplete minterm* which may include only a subset of all the input variables. The four mutations differ on the family of

Table 3: Example of FBM. First four columns: truth table of 3-parity problem. Remaining columns: output vectors of the drawn random incomplete minterm $M = X_1 \wedge \overline{X_2}$, of the parent $P = X_2 \vee (X_1 \wedge \overline{X_2} \wedge X_3)$, and of the offspring $O = P \vee M = X_1 \wedge X_2$. 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	M	P	O
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	**1**	**0**	**1**
1	0	1	0	**1**	**1**	**1**
1	1	0	0	0	1	1
1	1	1	1	0	1	1

incomplete minterms considered and their probability distributions. Like the forcing point mutation, these mutation operators can be implemented efficiently. The outstanding issue is therefore to see if they lead to a polynomial runtime for GSGP. Using an incomplete, rather than a full, minterm in the forcing mutation has the effect to force more than an entry of the output vector to the same value, i.e., it forces a block of entries to the same value, hence the name block mutations. These operators could work well on the sparse OneMax problem on exponentially long strings. As a first approximation, the idea behind these operators is that, by using sufficiently few variables in the incomplete minterm, the mutation would affect sufficiently many entries of the output vector, so that typically a non-neutral bit will be affected after a single application of the mutation operator. This, in effect, would be equivalent to searching the OneMax problem on the non-neutral entries, which are polynomially many, hence leading to a polynomial optimisation time. However, the analysis of block mutations is complicated by the fact that, unlike traditional mutations, they force *dependencies* between values at different entries of the string, as they cannot act separately on a single entry. In the following, we propose four block mutations that are analysable. These mutations are introduced in increasing order of the complexity of dependency between entries they produce.

3.2.1 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.*

Fixing v of the n input variables induces a *partition* of the output vector into $b = 2^v$ blocks each covering 2^{n-v} entries of the output vector (which has a total of 2^n entries). There is a one-to-one correspondence between the set of all incomplete minterms M 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 the same value, 0 or 1, selected at random with equal probability. Table 3 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 Boolean 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 3 the values of the entries in the first block of the optimum output vector Y are 0 and 1. This solution is therefore not reachable as reachable output vectors have both these entries set at zero or at one).

The particular settings of the analysis are as follows. 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 Boolean problem P at hand. The training set is sampled only once, before the search for a function satisfying it has started, and, in particular, it remains unchanged during evolution. Therefore, from an optimisation viewpoint, the training set T defines the specific instance of the Boolean problem P being tackled by the search. The reachability of the optimum is uniquely determined when the training set T of a problem P is fixed, and when also the set V of variables inducing the partition of the output vector is fixed. We are interested in the worst-case probability across all Boolean problems that GSGP with FBM initialised with a random fixed set of variables V can reach the optimum on a training set sampled uniformly at random from the problem at hand [2]. Furthermore, when GSGP with FBM can reach the optimum, we are interested in an upper-bound w.r.t. all Boolean problems (any P) and all problem instances (any T of P) of the expected runtime to reach the optimum.

Theorem 3. *For any Boolean 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 Boolean function satisfying the training set T in time $O(b \log b)$, with probability $p \approx e^{-\frac{\tau^2}{2b}}$ for $b >> \tau$, where $b = 2^v$ is the number of blocks partitioning the output vector.*

Proof. GSGP with FBM can reach the optimum provided that each distinct training example of the sampled training set T belongs to a distinct block. This holds irrespective of the prescribed output of the training examples, hence on any problem P. The optimum can be reached because with each training example in a different block, one application of FBM can flip independently the output value of each training example in the current solution. So, a function with any configuration of the outputs on the training set can be reached by the search.

[2]This is different from the traditional notion of probability of success of a search algorithm, as the source of the probabilistic outcome is the randomisation on the sampled problem instance (i.e., sampled training set T from P), and not the randomisation of the search algorithm.

The τ training examples are sampled uniformly at random on the set of all input-output pairs, i.e., uniformly on the output vector. The output vector is partitioned in b blocks of equal size. The probability of having each training example in a distinct block can be calculated by looking at it as a balls and bins process: blocks are bins of equal size, training examples are balls thrown uniformly at random on the bins. The probability of having at most one ball in each bin is the probability of throwing the first ball in an empty bin (1) times the probability of throwing the second ball in a bin left empty after the first throw ($(b-1)/b$), and so on, until all balls have been thrown. So we have:

$$P(b, \tau) = \prod_{i=1}^{\tau} \frac{b-(i-1)}{b}$$

When the number of bins is much larger than the number of balls, i.e., $b >> \tau$, $\frac{i-1}{b}$ is very small, as $e^x \approx 1 + x$ for x close to 0, then $P(b, \tau) \approx \prod_{i=1}^{\tau} e^{-\frac{i-1}{b}} = e^{-\frac{\tau(\tau-1)}{2b}} \approx e^{-\frac{\tau^2}{2b}}$.

When each example of the training set belongs to a distinct block, GSGP with FBM, which at each generation creates an offspring by forcing all the entries of an entire block to the same value, can modify independently each single output value of the current solution corresponding to an entry in the training set. This search is therefore equivalent to that of RLS with point forcing mutation on binary strings of length b on a sparse OneMax problem with τ non-neutral entries. As point forcing mutation makes no change half of the time, else it makes a point mutation, RLS with point forcing mutation takes twice the time of standard RLS with point mutation. With a routine argument, we obtain that the runtime of RLS on the sparse OneMax problem is $O(b \log \tau)$, which is bounded from above by $O(b \log b)$ as $b > \tau$. $\square$

The number of blocks b partitioning the output vector is critical for the performance of the search. On one hand, from the theorem above, the runtime of GSGP with FBM becomes larger for a larger number of blocks. Therefore, we would like to have as few blocks as possible. On the other hand, the probability of success becomes higher as b gets larger [3]. So, in this respect, the more blocks the better. The number of blocks b is an indirect parameter of the algorithm that can be chosen by choosing the number v of variables in the initial fixed set, as $b = 2^v$. Furthermore, the number of blocks b should be chosen relative to the size of the training set τ (e.g., we must have $b \geq \tau$ to have enough bins to host all balls individually). The question is therefore if we can always choose the number of blocks relative to the number of training examples such that we have both a polynomial runtime and high probability of success. It turns out that this is always possible, as stated in the theorem below.

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,*

[3]This can be intuitively understood, as increasing the number of bins (i.e., blocks) while keeping the number of balls unchanged (i.e., training examples) increases the chance of getting each ball in a separate bin.

GSGP 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. The number of blocks $b = 2^v$ is then a polynomial in n with degree strictly larger than $2c$, i.e., $b = n^{2c+\epsilon}$ for some $\epsilon > 0$. From Theorem 3, we have a success probability of $p \approx e^{-\frac{\tau^2}{2b}} = e^{-\frac{n^{2c}}{2n^{2c+\epsilon}}} = e^{-\frac{1}{2}n^{-\epsilon}}$. 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 $\epsilon > 0$. Again for Theorem 3 the running time is $O\left(n^{2c}\log n\right)$, that is polynomial in the size of the problem. $\square$

3.2.2 Varying Block Mutation

The Fixed Block Mutation operator is an unnaturally "rigid" operator as it requires to fix a set of variables at the beginning of the search in some arbitrary way (e.g., selecting them at random), which are then used throughout evolution as sole components of the incomplete minterms used in the forcing mutation. On one hand, this operator is appealing because it fixes a partition structure on the output vectors that remains unchanged during evolution, and that makes it particularly suitable for theoretical analysis. On the other hand, fixing the partition structure may restrict the search space too much and make the the optimum function lie outside the search space, hence unreachable. In the following, we introduce a more "flexible" mutation operator that extends the Fixed Block Mutation, and enlarges its search space.

DEFINITION 7. ***Varying Block Mutation (VBM):*** *Let $v < n$ be a parameter. VBM draws uniformly at random an incomplete minterm M made of v of the n input variables as a base for the forcing mutation.*

The VBM operator can be thought as constructing an incomplete minterm in two stages. First, it draws uniformly at random v distinct variables from the set of n input variables to form a set V variables. Then, it draws uniformly at random an incomplete minterm M comprising all variables in V as a base for the forcing mutation. The effect on the output vector of a single application of the VBM operator is, therefore, as follows: first, it draws uniformly at random a partitioning of the output vector with $b = 2^v$ blocks, each covering 2^{n-v} entries of the output vector; then, it selects one block of the current partitioning uniformly at random and it forces all the entries belonging to the block to the same value, 0 or 1, selected at random with equal probability.

Since when using VBM the partition structure on the output vector is not fixed, the search space seen by GSGP with this operator is larger than that with FBM, and in particular, the former search space covers completely the latter. The reason for that is that GSGP with VBM has always a chance to select the fixed variable set used by FBM to feed to VBM, and explore this part of the search space.

We say that an operator is *more expressive* than another when the search space covered by the former includes the search space covered by the latter. In this case VBM is more expressive than FBM. When the optimum of a certain Boolean problem is within the search space of a less expressive operator is also within the search space of a more expressive search operator, but the viceversa is not true in general. From its definition, the probability of success of an operator is higher than the probability of success of all operators less expressive than it. We say that an operator is completely expressive when its search space covers all solutions of all Boolean problems and all training sets, thus its probability of success is 1. Since VBM is more expressive than FBM its probability of success is higher. However, as FBM, also VBM is not able to always reach the optimum for any choice of Boolean problem and training set: there exists certain training set configurations such that for *all* partitionings induced by any choice of v input variables, there are always at least two training examples with different output values belonging to the same block.

PROPOSITION 1. *Consider GSGP with VBM using $v < n$. Then there exists a training set of size $\tau = n + 1$ of the Boolean parity problem on which this algorithm cannot find the optimum.*

PROOF. Consider the training set with input entries $\mathcal{T} = \{x \in \{0,1\}^n \mid \exists! i \in \{1,...,v\}\ x_i = 1\} \cup \{(0,\ldots,0)\}$ and as output values those of the parity problem (i.e., $(0\cdots,0)$ has output 0, while all the other vectors have output 1, as in Example 1).

Let M be an incomplete minterm of $v < n$ variables. If the all-zero vector satisfy M, then there exists a vector with output value 1 in $\mathcal{T}$ that satisfy M. Since VBM is a forcing mutation and every incomplete minterm that is satisfied by the all-zero vector is always satisfied also by another vector in τ with different output value, it is not possibile to obtain a perfect score on the training set τ. $\square$

EXAMPLE 1. *We will illustrate the training set of Proposition 1 for $n = 3$ variables. In this case the training set is:*

	v_1	v_2	v_3	$f(v_1, v_2, v_3)$
$x_1 =$	0	0	0	0
$x_2 =$	1	0	0	1
$x_3 =$	0	1	0	1
$x_4 =$	0	0	1	1

where v_1, v_2, v_3 are variables and $x_1, \ldots, x_4$ elements of the training set. Note that for any choice of two variables we cannot separate x_1 from another element of the training set: with v_1 and v_2 we cannot separate x_1 and x_4, with v_1 and v_3 we cannot separate x_1 and x_3, and with v_2 and v_3 we cannot separate x_1 and x_2.

In the previous section, we have shown that GSGP with FBM, when it can reach the optimum, it finds it, quite remarkably, in polynomial time in the worst case w.r.t. all Boolean problems and training sets. Unfortunately, GSGP with VBM requires exponential time to find the optimum in the worst case.

PROPOSITION 2. *Consider GSGP with VBM using $v = c \log n < n$ variables for some constant $c > 0$. Then there exists a training set of size $\tau = n^c$ of the Boolean parity problem on which this algorithm needs superpolynomial time to find the optimum.*

PROOF. Consider the training set with input entries $\mathcal{T} = \{x \in \{0,1\}^n \mid \forall i \in \{v+1, ..., n\}\ x_i = 0\}$ and as output values those of the parity problem (see Example 2). Note that the number of entries in the training set is $\tau = 2^v = n^c$.

For every selection of variables different from the first v variables, any incomplete minterm M made with those variables will be such that the subset $\mathcal{T}'$ of $\mathcal{T}$ of all train instances that satisfy M contains exactly $|\mathcal{T}'|/2$ instances with output value 1 and $|\mathcal{T}'|/2$ instances with output value 0. Since VBM is a forcing mutation, using M as the incomplete minterm for the mutation it is not possible to increase the fitness (i.e., if all output are forced to 1 then we obtain the incorrect value for half of the instances in $\mathcal{T}'$, the same if we force 0).

There is only one selection of variables that allows to increase the fitness (i.e., the first v variables). As the selection is uniform across all the subsets of v variables, in expectation only one step every $\binom{n}{v} \geq \left(\frac{n}{v}\right)^v = \frac{n^{c \log n}}{(c \log n)^{c \log n}}$ can produce an individual with a better fitness and that can be accepted by the mutation. □

EXAMPLE 2. *Consider the following training set in four variables:*

	v_1	v_2	v_3	v_4	$f(v_1, v_2, v_3, v_4)$
$x_1 =$	0	0	0	0	0
$x_2 =$	1	0	0	0	1
$x_3 =$	0	1	0	0	1
$x_4 =$	1	1	0	0	0

where $v_1, \ldots, v_4$ are variables and $x_1, \ldots, x_4$ elements of the training set. Note that the only choice of variables that allow us to select elements of the training set with equal output value (and thus increase the fitness) is $\{v_1, v_2\}$ However there are $\binom{4}{2} = 6$ different possible subsets of two variables. Hence the fitness increase just once every 6 generations, in expectation.

3.2.3 Fixed Alternative Block Mutation

In the following we introduce a mutation operator which is half-way between FBM and VBM, which has higher probability of success than FBM and finds the optimum in polynomial time in the worst case.

DEFINITION 8. ***Fixed Alternative Block Mutation (FABM):*** *Let $v < n$ be a parameter. Let us consider a* fixed *partition of the set of the n input variables (fixed in some arbitrary way at the initialisation of the algorithm) into n/v groups of v variables each. These groups of variables are the set of fixed alternatives of the mutation operator. FABM selects uniformly at random a group of variables among the fixed alternatives, and then draws uniformly at random an incomplete minterm M comprising all the variables in that group as a base for the forcing mutation.*

The FABM operator is half-way between FBM and VBM: as FBM, it fixes the choice of (groups of) variables at the initialisation, however, as VBM, it can use each time different (groups of) variables to construct the incomplete minterm to feed to the forcing mutation. From their definitions we have that FABM is more expressive than FBM, but less expressive than VBM. So, the probability of success of FABM is bounded below by the probability of success of FBM, and bounded above by the probability of success of VBM. We saw in the previous section that VBM is not completely expressive as there are some problems it cannot solve. Therefore, FABM is also not completely expressive as it is less expressive than VBM. The following theorem relates the performance of GSGP with FABM to those with FBM.

THEOREM 5. *For any Boolean problem of size n on which GSGP with FBM with v fixed variables finds an optimal solution with probability p in time T, GSGP with FABM with groups of v variables finds an optimal solution in time $T' = O\left(\frac{n}{v}T\right)$ with success probability $p' \geq 1 - (1-p)^{n/v}$.*

PROOF. For GSGP with FABM with groups of v fixed variables, it holds that:

- if FBM finds the optimum for a given problem and training set then it exists a group of variables among the alternatives for which we can always improve any current non-optimal solution;
- the worst case happens when there is only one group of variables that can be used to improve on the current solution. In this case the runtime is n/v times slower than the runtime of GSGP with FBM as, in expectation, the algorithm can draw a group of variables that allows for an improvement only once every n/v trials. Thus the running time of GSGP with FABM is $T' = \frac{n}{v}T$.

GSGP with FABM tries to find the optimum using n/v disjoint groups of the input variables. The probability that the optimum cannot be found using any of those subsets is bounded above by $(1-p)^{n/v}$, thus the probability of success of FABM is $p' \geq 1 - (1-p)^{n/v}$. □

As a corollary, as GSGP with FBM finds an optimal solution in polynomial time with high probability, GSGP with FABM finds it with higher probability and higher, but still polynomial, time.

3.2.4 Multiple Size Block Mutation

The block mutation operators considered so far cannot guarantee to find the optimum in all cases, but when they do they may find it in polynomial time. Instead, pointwise mutation operators can always find the optimum as they can act on the value of any entry of the output vector independently from any other entry. However, they need 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 9. ***Multiple Size Block Mutation (MSBM):*** *The operator MSBM samples the number of variables v to consider uniformly at random between 0 and n. Then, it selects v variables at random from the set of n input variables, and it generates uniformly at random an incomplete minterm M using those variables, which is then used as a base for the forcing mutation.*

The effect on the output vectors of the feature of MSBM that the number of variable v is not fixed but it can be any number between 0 to n at each application is that the number of blocks partitioning the output vectors can vary from 1 block with 2^n entries to 2^n blocks with a single entry each. On one hand, as for pointwise mutation, this allows GSGP 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, GSGP with MSBM can solve efficiently any problem that can be solved efficiently by GSGP with the block mutation VBM. This is because MSBM can simulate VBM on v variables in time which is in the worst case n times larger (as the probability of selecting exactly v variables by MSBM is $1/n$). However, the time needed by GSGP with MSBM to reach an optimal solution can be exponential on some training set, as shown below.

PROPOSITION 3. *There exists a training set for which GSGP with MSBM takes expected exponential time to find the optimum.*

PROOF. Consider the training set with input entries $\mathcal{T} = \{x \in \{0,1\}^n \mid \exists! i \in \{1,...,v\}\ x_i = 1\} \cup \{(0,\ldots,0)\}$ and as output values those of the parity problem. Except for the all-zero vector, that has output value 0, all the others output values are 1 (See Example 1).

By the same reasoning as the proof of Proposition 1, the only way to reach the optimum is to obtain a minterm M that is satisfied by the all-zero vector and no other vector. Note that there exists only one minterm of n variables with this property.

Since there are 2^n complete minterm and we select each of them with equal probability, selecting the correct one requires an exponential number of trials in expectation. $\Box$

The training set of Example 1 is also a training set in which MSBM takes expected exponential time to find the optimum.

On Boolean problems whose solutions can be written as Boolean formulas with a small number of conjunctions, and each conjunction uses few variables, it is possible to find the optimum in polynomial time.

THEOREM 6. *Let ϕ be a DNF formula with $\alpha = Poly(n)$ conjunctions and every conjunction with at most $\beta = O(1)$ variables. Then ϕ can be obtained by GSGP with MSBM in expected polynomial time.*

PROOF. The time to select a correct conjunction is bounded by the number of conjunctions of β variables, i.e., by $\binom{n}{\beta}2^\beta = O(n^\beta 2^\beta)$ conjunctions, and the time to select the correct conjunction size, i.e., $O(n)$.

The action of selection preserves correctly selected conjunctions, thus α is only a multiplicative factor, since only α conjunctions must be selected. Therefore, the time needed to reach the optimum is $O\left(\alpha n^{\beta+1} 2^\beta\right)$. Since β is a constant and α is polynomial in n, the resulting time is polynomial in n. $\Box$

It is interesting to note that the previous result is independent from the choice of the training set, and it depends solely on the class of Boolean problem.

Note that for all the block mutation considered, the length of the Boolean function found as optimum is bounded above by the time complexity. In fact, as all the block mutations considered add a single (incomplete) minterm at each generation of GSGP, the number of minterms forming the optimum is bounded from above by the runtime, so it is polynomial. Thus, when the time complexity is polynomial, the length of the Boolean function found is also polynomial .

4. EXPERIMENTS

In this section, we present an experimental investigation of the time to reach the optimum and the success rate for GSGP with the four block mutations introduced in this paper. The theoretical analysis has focused on worst case analysis and asymptotic behaviour. The empirical investigation complements the theoretical one focusing instead on the average case for growing finite problem size.

A problem instance is a pair of a Boolean function P and a training set $\mathcal{T}$. We set the size of the training set τ equal to the number on input variables n of the Boolean function, i.e., the problem size. At each run, a randomly selected training set of a randomly selected function is generated and tested on GSGP with the four mutations. For each problem size from 8 to 48 with step 8, 100 runs were performed. For all mutations except MSBM, the number of variables selected was $v = 2\lceil \log_2 n \rceil$. We used a GP with population of one, initialised with a random minterm, mutation applied with probability 1, and the offspring replaced the parent only if it had better fitness (i.e., a higher number of correct outputs on the training set). A run was stopped when either the optimal solution was found or 10^5 generations had passed [4].

The results on the success rates for the different mutations is presented in Table 4. As for FBM, with the chosen v theory says that asymptotically GSGP has a constant probability of success different from 1. The fluctuations and deviations from constancy for growing problem size seen experimentally are due to the rounding in the used expression for v and to the fact that the theory does not apply for "small" problem size. As for both VBM and FABM, from theory we expect an asymptotical rate of convergence higher than

[4]The choice of using a limit as high as 10^5 generations was due to the necessity to avoid stopping a run too early, thus decreasing our estimates of both the success rate and the expected time to reach an optimal solution. The value 10^5 was empirically chosen both to reach that goal and to allow a reasonable completion time for the experiments.

Table 4: Success rate of GSGP on random Boolean problems for different problem sizes and mutations.

	Problem Size					
	8	16	24	32	40	48
FBM	0.95	0.76	0.93	0.74	0.87	0.88
VBM	0.95	0.99	1.00	1.00	1.00	1.00
FABM	1.00	1.00	1.00	1.00	1.00	1.00
MSBM	1.00	1.00	1.00	1.00	1.00	1.00

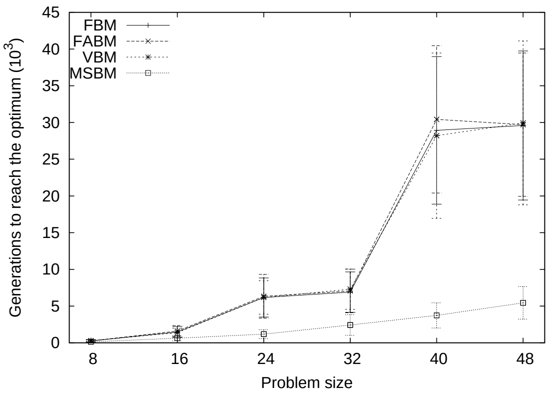

Figure 2: Average number of generations GSGP took to attain an optimal solution on random Boolean problems. The error bars represents one standard deviation above and below the average.

for FBM. This is confirmed experimentally. It is also not surprising that the rate of convergence approaches 1 for increasing problem size for these two mutations, as the chosen v is a threshold point for the asymptotic behaviour of FBM between constant probability of success different from one, and probability of success one. As expected from the theory, MSBM converged to the optimum at all times given enough time.

The results on the number of generations to reach an optimal solution are presented in Table 5. A plot of the optimisation time for increasing problem size is presented in Fig. 2. Experimentally FBM, VBM and FABM have very similar performance. The experiments estimate the average-case performance which draw quite a different picture from the worst-case performance determined theoretically in which FBM and FABM have a polynomial worst case, and VBM an exponential worst case. The rather non-smooth shape of the performance curves for these three mutations is caused by the rounding effect in the used expression for v. Furthermore, it is striking that MSBM performs significantly better than the other mutations, and, unlike those, MSBM seems to present a linear trend between optimisation time and problem size. Again, the experimental average-case picture is different from the theoretical one, which prescribes an exponential worst case for this mutation.

5. SUMMARY AND FUTURE WORK

Geometric semantic genetic programming searches directly the semantic space of functions. Seen from a geometric viewpoint, the genotype-phenotype mapping of GP becomes very simple, and allows us to derive explicit algorithmic characterizations of semantic operators for different domains. The search of GP with semantic operators on functions (genotypes) is formally equivalent to the search of a GA with standard search operators on their output vectors (phenotypes). Remarkably, the landscape seen by the equivalent GA, hence by GSGP, is always a cone by construction, for any problem and any domain. This, at the same time, makes the search for the optimum much easier than for traditional GP, and it opens the way to analyse theoretically the runtime of GP in a *general settings* – an important open challenge – in an easy manner by extending known results for GA on one-max-like problems. In this paper, we have started this line of theory and presented a runtime analysis of GSGP with various types of mutation on the class of all Boolean functions.

There are a number of peculiar issues arising with GSGP, which required a careful design of mutation operators to obtain an efficient algorithm. The fitness landscape seen by GSGP is a heavily neutral extension of OneMax on exponentially long bit strings, in which only a polynomial number of entries contribute to the fitness. Standard GA mutation operators, i.e., point mutation and bitwise mutation, give rise to an exponential runtime. Furthermore, some semantic mutation operators do not admit an efficient implementation on Boolean functions (i.e., may require exponential time for generating a single offspring). Blockwise mutations are mutation operators that can be implemented efficiently and that at each application force a whole block of bits to a random value. We proposed four block mutations, two of which (i.e. FBM and FABM) reach the optimum in polynomial time with high probability on any Boolean problem. This is a *surprisingly general positive result* about the worst case performance of GSGP. Experimental results testing the average-case complexity of the block mutations have shown that one of the mutation (i.e. MSBM) seems, on the average case, much superior to the others, as it finds the optimum all the times and its runtime grows only linearly in the problem size.

Table 5: Minimum (min), median (med) and maximum (max) number of generations for GSGP to reach the optimum on random Boolean problems.

		Problem Size					
		8	16	24	32	40	48
FBM	Min	34	294	1659	2353	14445	10434
	Med	207	1330	5862	6544	28456	27916
	Max	649	4959	14229	17068	69617	61842
VBM	Min	44	471	1815	1736	11970	11225
	Med	235	1531	5406	6432	29358	28437
	Max	759	3841	17355	22863	60965	56365
FABM	Min	18	438	2316	2289	9900	11901
	Med	254	1244	5756	6719	25827	28499
	Max	863	3925	13675	16792	81845	66043
MSBM	Min	5	117	182	477	1072	2291
	Med	115	547	1053	2222	3333	4928
	Max	450	2174	2677	8229	11907	14044

There is plenty of future work. As training sets, which define specific problem instances, are sampled at random, it would be interesting to analyse theoretically the average-case complexity beside the worst-case as done in the current paper. We also would like to do further experimental investigations of GP with the new semantic operators on standard GP benchmarks, to see how they perform on more practical problems. As there are now a number of runtime results for GA with crossover, we intend to extend those to analyse GSGP with semantic crossover. At present, how functions trained by GP generalise on unseen inputs is a big mystery. As the effect of semantic operators on the output vectors is transparent, this may allow us to explicitly characterise the dependencies between training and testing sets reveling exactly what the inductive bias of GSGP is. Finally, we want to analyse GSGP on other domains. This seems to be within reach as, e.g., semantic operators for arithmetic functions and classifiers give rise to cone landscapes on real vectors and integer vectors, which have been studied already for traditional GA and ES, and whose analysis may be extended to GSGP.

6. ACKNOWLEDGEMENTS

The authors are grateful to Dirk Sudholt for helping check the proofs.

7. REFERENCES

[1] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics – Foundations and Recent Developments.* World Scientific, 2011.

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

[3] L. Beadle and C. G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, 2009.

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

[5] 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.

[6] D. Jackson. Phenotypic diversity in initial genetic programming populations. In *Proc. of EuroGP 2010*, pages 98–109, 2010.

[7] K. Krawiec and P. Lichocki. Approximating geometric crossover in semantic space. In *Proc. of GECCO '09*, pages 987–994, 2009.

[8] K. Krawiec and B. Wieloch. Analysis of semantic modularity for genetic programming. *Foundations of Computing and Decision Sciences*, 34(4):265–285, 2009.

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

[10] P. K. Lehre and C. Witt. Black-box search by unbiased variation. In *Proceedings of Genetic and Evolutionary Computation Conference*, 2010.

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

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

[13] A. Moraglio. *Towards a Geometric Unification of Evolutionary Algorithms.* PhD thesis, University of Essex, 2007.

[14] A. Moraglio. Abstract convex evolutionary search. In *Workshop on the Foundations of Genetic Algorithms*, pages 151–162, 2011.

[15] A. Moraglio, K. Krawiec, and C. Johnson. Geometric semantic genetic programming. In *Workshop on Theory of Randomized Search Heuristics*, 2011.

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

[17] A. Moraglio and R. Poli. Topological interpretation of crossover. In *Proc. of GECCO '04*, pages 1377–1388, 2004.

[18] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity.* Springer, 2010.

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

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

[21] 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*, 12(2):91–119, 2011.