Geometric Semantic Genetic Programming

Alberto Moraglio[1], Krzysztof Krawiec[2], and Colin G. Johnson[3]

[1] School of Computer Science, University of Birmingham, UK
A.Moraglio@cs.bham.ac.uk
[2] Institute of Computing Science, Poznan University of Technology, Poland
kkrawiec@cs.put.poznan.pl
[3] School of Computing, University of Kent, UK
C.G.Johnson@kent.ac.uk

Abstract. Traditional Genetic Programming (GP) searches the space of functions/programs by using search operators that manipulate their syntactic representation, regardless of their actual semantics/behaviour. Recently, semantically aware search operators have been shown to outperform purely syntactic operators. In this work, using a formal geometric view on search operators and representations, we bring the semantic approach to its extreme consequences and introduce a novel form of GP – Geometric Semantic GP (GSGP) – that searches *directly* the space of the underlying semantics of the programs. This perspective provides new insights on the relation between program syntax and semantics, search operators and fitness landscape, and allows for principled formal design of semantic search operators for different classes of problems. We derive specific forms of GSGP for a number of classic GP domains and experimentally demonstrate their superiority to conventional operators.

1 Introduction

Traditional genetic programming ignores the *meaning* of programs, as the search operators it employs act on their syntactic representations, regardless of their semantics. E.g., subtree swap crossover is used to recombine functions represented as parse trees, regardless of trees representing boolean expressions, mathematical functions, or computer programs. Whereas this guarantees producing syntactically well-formed expressions, why should such a *blind* syntactic search work well for different problems and across domains? In the end, it is the meaning of programs that determines how successful search is at solving the problem.

The semantics of a program can be formally defined in a number of ways. It can be a canonical representation, so that any two programs with the same semantics/behaviour have the same canonical representation (e.g., Binary Decision Diagrams (BDD) for boolean expressions). It can be a description of the behaviour of the program using a logical formalism. This is used in formal methods to reason formally about programs. From a strict search viewpoint, it may be argued that the semantics of a program is just its fitness. Finally, it can also be defined as the mathematical function computed by a program, i.e., the set of input-output pairs making up the computed function.

In the literature, there are a number of works using the semantics of programs to improve GP. As many individuals encode the same function, some researchers use canonical representations of functions to enforce semantic diversity throughout evolution, by creating semantically unique individuals in the initial population [2, 4], and by discarding offspring of crossover and mutation when semantically coinciding with their parents [3, 1]. Uy et al. [11] propose a 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. Also Krawiec et al. [5, 6] have used a notion of semantic distance to propose a crossover operator for GP trees that is approximately a geometric crossover [10, 8] in the semantic space. Interestingly, the fitness landscape induced by this operator has perfect fitness-distance correlation. The operator was implemented approximately by using a traditional crossover, generating a large number of offspring, and accepting only those offspring that were "semantically intermediate" with respect to the parents.

Whereas, overall the semantically aware methods above produced superior performance to traditional methods, they are *indirect*: 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. This has two drawbacks: (i) these implementations are very wasteful as heavily based on trial-and-error; (ii) they do not provide insights on how syntactic and semantic searches relate to each other. Would it then be possible to search *directly* the semantic space of programs? More precisely, would it be possible to build search operators that, acting on the syntax of the parent programs, produce offspring that are *guaranteed* to respect some semantic criterion/specification by construction? Krawiec et al. [5, 6] stated that due to the complexity of the genotype-phenotype mapping in GP, a direct implementation of exact semantic operators is probably impossible.

The present paper brings the following contributions: (i) it formalises the notions of semantic distance, semantic geometric operators and semantic fitness landscapes; (ii) it proves that the fitness landscapes seen by geometric semantic operators are always cone landscapes, which are easy to search; (iii) it shows that, contrary to widespread belief, the genotype-phenotype map of commonly considered GP domains is, in an important sense, very easy, not complex; (iv) it introduces a general method to derive *exact* semantic geometric crossovers and mutations for different problem domains that search *directly* the semantic space; (v) it derives semantic operators for the Boolean domain, arithmetic domain, and program domain; (vi) it reports experimental results for a standard test-bed of GP problems.

2 Abstract Geometric Semantic Search

In this section, we report non-operational definitions of geometric semantic operators and their properties. They are characterised algorithmically in Section 3.

A search operator $CX : S \times S \to S$ is a *geometric crossover* w. r. t. the metric d if for any choice of parents p_1 and p_2, any of their offspring $o = CX(p_1, p_2)$ is in the metric segment between parents. A search operator $M : S \to S$ is a *geometric ϵ-mutation* w. r. t. the metric d if for any choice of the parent p, any of its offspring $o = M(p)$ is in the metric ball of radius ϵ centered in the parent. Given a fitness function $f : S \to \mathbb{R}$, the geometric search operators induce or see the fitness landscape (f, S, d). Many well-known recombination operators across representations are geometric crossovers [8], e. g., all mask-based crossovers on binary strings are geometric crossovers w. r. t. Hamming distance. Point mutation on binary strings is geometric 1-mutation w. r. t. Hamming distance. 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 is not directly linked to a representation, the geometric operators are well-defined but an algorithmic description for them can be hard to derive.

Genetic programming is essentially a supervised learning method: given a fixed set of input-output pairs $T = \{(x_1, y_1), ..., (x_N, y_N)\}$ (i.e., training set or fitness cases), a function $h : X \to Y$ belonging to a certain fixed class H – specified by the chosen terminal and function sets – is sought (evolved) that interpolates the known input-output pairs, i.e., $\forall (x_i, y_i) \in T : h(x_i) = y_i$. The fitness function $F_T : \mathrm{H} \to \mathbb{R}$ measures the error of a function 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.

Let $I = (x_1, ..., x_N)$ and $O = (y_1, ..., y_N)$ be the input and the output vectors, respectively, associated with the training set T. Let $O(h)$ be the vector of the outputs of a function h when queried with the inputs I, i.e., $O(h) = (h(x_1), ..., h(x_N))$. The function $O : \mathrm{H} \to Y^N$ can be interpreted as *genotype-phenotype mapping* as it maps a representation of a function h (i.e., genotype) to the actual outcome of the application of function h on the input vector I (i.e., phenotype) represented by its output vector.

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 $O(h)$ measured using some suitable metric D, i.e., $F_T(h) = D(O, O(h))$ (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 (i.e., Y^N). A suitable metric D to measure the error as a distance between binary vectors is the Hamming distance. When considering functions returning real values (e.g., regression), the output vectors are real vectors, and suitable metrics to measure the error are Euclidean and Manhattan distances, each of which gives rise to a different type of fitness function.

We define *semantic distance* SD between two functions $h_1, h_2 \in \mathrm{H}$ as the distance between their corresponding output vectors w. r .t. the input vector of all possible inputs (i.e., $I = (x_i)$ for all $x_i \in X$) measured with the metric D used in the definition of the fitness function F_T, i.e., $SD(h_1, h_2) = D(O(h_1), O(h_2))$. The semantic distance SD is a genotypic distance induced from a phenotypic metric D, via the genotype-phenotype mapping O. As O 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 belong to the same semantic class $\overline{h}$ iff their semantic distance is zero. Then, SD can be interpreted as a metric on the set of semantic classes of genotypes $\overline{H}$.

We define *semantic geometric operators* as geometric crossover and mutation specified on the space of (classes of) functions endowed with the distance SD. E.g., semantic geometric crossover 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 $x_i \in X$). The effect of SD being defined on the space of classes of functions $\overline{H}$, rather than on the space of functions H, is that the geometric crossover is only function of the semantic classes of the parents $\overline{h_1}$, $\overline{h_2}$ rather than directly of the parents h_1, h_2 (i.e, their specific representations), and the returned offspring can be any function h_3 belonging to the offspring class $\overline{h_3}$ (i.e., any function with the prescribed output vector/semantics).

The *semantic fitness landscape* seen by an evolutionary algorithm with semantic geometric operators is a *cone landscape by construction* [4] as, from the definition of semantic distance, the fitness of a solution is its distance in the search space to the optimum. 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 [9] that EAs with geometric operators can optimise cone landscapes efficiently very generally for virtually any metric.

GP search with geometric operators w. r .t. the semantic distance SD on the space of function classes $\overline{H}$ 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 (see diagram (1)). [5] E.g., for Boolean

[4] The landscape includes also a form of neutrality. As the training set covers a fraction of all possible input-output pairs of a function, only that part of the output vector of a function affects its fitness, the remaining large part is "inactive". This does not affect crossover, but it may make mutation ineffective.

[5] 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.

functions, semantic GP search is equivalent to GA search on binary strings on OneMax of dimension N.

3 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 O, at the bottom. It holds that for any $T1, T2$ and $T3 = GX_{SD}(T1, T2)$ then $O(T3) = GX_D(O(T1), O(T2))$.

$$\begin{array}{ccc} T1 \times T2 & \xrightarrow{GX_{SD}} & T3 \\ \downarrow O \quad \downarrow O & & \downarrow O \\ O1 \times O2 & \xrightarrow{GX_D} & O3 \end{array} \tag{1}$$

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 O 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. We will give a recipe to derive specific semantic crossovers for new domains.

Definition 1. *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).*

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.*

Proof. The offspring function is $T3 = (T1 \wedge TR) \vee (\overline{TR} \wedge T2)$. Expanding it for any input i: $T3(i) = (T1(i) \wedge TR(i)) \vee (\overline{TR(i)} \wedge T2(i))$. So, for any entry i of the output vectors: $O(T3)(i) = (O(T1)(i) \wedge O(TR)(i)) \vee (\overline{O(TR)(i)} \wedge O(T2)(i))$. In the last expression, the Boolean expression at each position i is a multiplexer function which, depending on the bit-value of $O(TR)(i)$ (piloting bit), assigns either $O(T1)(i)$ or $O(T2)(i)$ to $O(T3)(i)$. Then, the output vector $O(TR)$ acts as a crossover mask on the parent output vectors $O(T1)$ and $O(T2)$ to produce the offspring output vector $O(T3)$. This is a geometric crossover on output vectors w. r. t. the Hamming distance.

Let us now consider the Real Functions domain (e.g., for symbolic regression).

Definition 2. *Given two parent functions $T1, T2 : \mathbb{R}^n \to \mathbb{R}$, the recombinations SGXE and SGXM return the real function $T3 = (T1 \cdot TR) + ((1 - TR) \cdot T2)$ where TR is a random real constant in $[0, 1]$ (SGXE), or a random real function with codomain $[0, 1]$ (SGMX).*

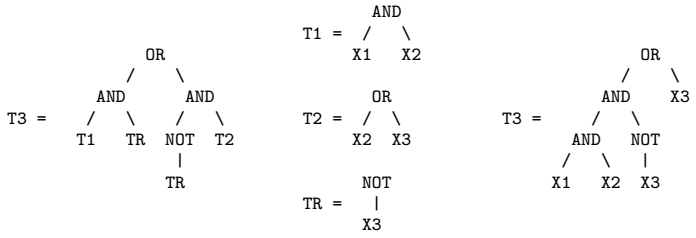

Fig. 1: Left: Semantic Crossover scheme for Boolean Functions; Centre: Example of parents (T1 and T2) and random mask (TR); Right: Offspring (T3) obtained by substituting T1, T2 and TR in the crossover scheme and simplifying.

Theorem 2. *SGXE and SGXM are semantic geometric crossovers for the space of real functions with fitness function based on Euclidean and Manhattan distances, respectively, for any training set and any real problem.*

Proof. By expanding the offspring function on the inputs and considering every entry i of the output vectors: $O(T3)(i) = (O(T1)(i) \cdot O(TR)(i)) + ((1 - O(TR)(i)) \cdot O(T2)(i))$. As $O(TR)(i) \in [0, 1]$, at each position the value of $O(T3)(i)$ is a convex combination of the values of $O(T1)(i)$ and $O(T2)(i)$. So, the vector $O(T3)$ is within the hyper-box delimited by $O(T1)$ and $O(T2)$, i.e., it is in their Manhattan segment. Expressing the above relation in functional form: $O(T3) = (O(T1) \cdot O(TR)) + ((1 - O(TR)) \cdot O(T2))$. When additionally $O(TR)$ is constant in i, we see that $O(T3)$ is a convex combination of the vectors $O(T1)$ and $O(T2)$, i.e., it is in their Euclidean segment.

Let us now consider the Computer Program domain intended as functions with symbols as inputs (IS) and outputs (OS). The following can be easily extended to other types of inputs and outputs.

Definition 3. *Given two parent programs* `T1`, `T2` $: IS^n \to OS$*, the recombination SGXP returns the offspring program* `T3 = IF CONDR THEN T1 ELSE T2` *where* `CONDR` *is a random program whose output is interpreted as a logical value.*

Theorem 3. *SGXP is a semantic geometric crossovers for the space of programs with fitness function based on Hamming distance, for any training set and any problem.*

Proof. By expanding the offspring program on the inputs and considering every entry i of the output vectors: `O(T3)(i) = IF O(CONDR)(i) THEN O(T1)(i) ELSE O(T2)(i)`. This means that for each input, the output value of `T3` is that of `T1` or `T2` depending of the value of `CONDR`, which is then acting as a crossover mask on `T1` and `T2`. This is a geometric crossover on the output vectors w. r. t. the Hamming distance (for symbolic vectors).

Definition 4. ***Semantic Mutations. Boolean:*** *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.* ***Arithmetic:*** *Given a parent function $T : \mathbb{R}^n \to \mathbb{R}$, the mutation SGMR with mutation step ms returns the real function $TM = T + ms \cdot (TR1 - TR2)$ where $TR1$ and $TR2$ are random real functions.* ***Programs:*** *Given a parent program* `T`*, the mutation SGMP returns the offspring program* `TM = IF CONDR THEN OUTR ELSE T` *where* `CONDR` *is a condition which is true only for a single random setting of all input parameters, and* `OUTR` *is a random output symbol. The offspring can be expressed as nested* `IF-THEN-ELSE` *statements with simple conditions of a single input parameter each.*

Theorem 4. *SGMB and SGMP are semantic* 1*-geometric mutations for boolean functions and of programs, respectively, with fitness function based on Hamming distance. SGMR is a semantic* ϵ*-geometric mutation for real functions with fitness function based on Euclidean and Manhattan distances. The mean of its probability distribution is the parent, and* ϵ *is proportional to the step ms.*

General construction method: it can be obtained by reversing the common argument in the proofs above: (i) take the geometric crossover on output vectors associated with the distance used in the fitness function; (ii) consider the action of the recombination operator on a single entry of the output vectors; (iii) use the domain-specific language of the particular class of functions considered to describe the recombination action on a single entry; (iv) that *description* is the scheme to produce the offspring. Note that the offspring is not only the *effect* of crossover, it is also the *description* of how to crossover its parents. The target domain-specific language must be expressive enough to describe the recombination. This seems to be the case for most GP problems.
Simplification: As the syntax of the offspring of semantic crossover contains both parents, the size of individuals *grows exponentially* in the number of generations. To keep their size manageable, 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 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 (e.g., Espresso). Computer algebra systems (e.g., Maple) can be used to simplify symbolically mathematical functions, like polynomials, and more complicated expressions including *sin*, *cos*, *exp*, etc. if used in disciplined ways (e.g., nested *sin* not allowed). Formal methods (e.g., static analysis) can be used to simplify computer programs (but loops/recursion may be challenging).
Does syntax matter? *In theory*, it does not matter! The offspring is a function obtained from a functional combination of parent functions. The offspring is defined purely functionally and does not depend on how functions are actually represented (e.g., trees, graphs, sequences) and what language is used (e.g., Java, Lisp, Prolog), as long as the semantic operators can be described in that language. *In practice*, syntax does matter! As genotype structure and language influence the way random genotypes are generated, as different representations suggest different "natural" ways of generating them. This affects the offspring distribution of semantic operators, the semantic diversity in the initial population, and the dependencies in the crossover mask. Furthermore, some representations may be easier to simplify, and may have preferable inductive bias (i.e., generalise better on unseen inputs).

4 Computational Experiments

We compare GP, semantic GP (SGP), and semantic stochastic hill climber (SSHC), which employs semantic mutation to explore the neighbourhood. In all experiments GP and SGP use a generational scheme with tournament selection (size 5), crossover and mutation, which are always engaged. We give the

algorithms the same number of evaluations, set as the number needed by SSHC to typically find the optimum. We also compare algorithms on CPU time: GPt is GP running for the same time as the greater of average execution times of SGP and SSC. Below are the main settings of the experimental setups considered. Other parameters are set to ECJ's defaults [7].

Boolean Functions (Table 1): *Test-bed*: standard GP benchmark. *Fitness function*: Hamming distance to the output vector of the target function queried on all inputs. *GP*: standard GP with instruction set: 'And', 'Or', 'Not'. *SGP and SSHC*: individuals are Boolean expressions in disjunctive normal form; SGMB and SGXB with a mask TR being a random minterm of a random subset of input variables; simplification of offspring by Espresso. *Comparison*: budget of $2n \cdot 2^n$ evaluations, where n is the number of input variables; as to population size, GP and SGP have $\max\{\sqrt{2^n}, 10\}$, and GPt has $\max\{\sqrt{2^n}, 50\}$ (and from 20 to 200 times more evaluations).

Polynomial Regression (Table 2): *Test-bed*: univariate polynomials of degrees from 3 to 10, with real-valued coefficients uniformly drawn from $[-1, 1]$. *Fitness function*: Euclidean distance to the output vector of the target function queried on 20 inputs in $[-1, 1]$. *GP*: Standard GP with instruction set: '+', '-', '*', 'x', constant. *SGP and SSHC*: individuals are polynomials of degree 10, initialised with coefficients drawn uniformly from $[-1, 1]$; SGXE and SGMR with step $ms = 0.001$; implicit simplification (i.e, weighted sums of polynomials). *Comparison*: budget of 100,000 evaluations, with population size 1,000 for GP, and 20 for SGP.

Classifiers (Table 3): *Test-bed*: $IS = \{1, ..., n_c\}$, $OS = \{1, ..., n_{cl}\}$, target functions $f : IS^{n_v} \rightarrow OS$ are $f(x_1, x_2, ..., x_{n_v}) = ((x_1 + x_2) \mod n_{cl}) + 1$, for all combinations of $n_v = 3, 4$, $n_c = 3, 4$ and $n_{cl} = 2, 4, 8$. *Fitness function*: Hamming distance to the output vector of the target function queried on all inputs. All algorithms use classifiers of the form: `<CF> := IF <COND> THEN <CF> ELSE <CF> || <OS>; <COND> := <`x_i`> = <IS>`, and Ramped-half-and-half initialisation. *SGP and SSHC* use SGXP and SGMP, and simplification of classifiers done by an Espresso-like simplifier. *Comparison*: budget of $2n_{cl}n_v n_c^{n_v}$ evaluations; as to population size, GP and SGP have $\max\{\sqrt{n_c^{n_v}}, 10\}$, and GPt has $\max\{\sqrt{n_c^{n_v}}, 50\}$ (and from 10 to 130 times more evaluations).

Analysis: *Performance*: for all domains and problems, SSHC and SGP find consistently near-optimal solutions, beating by far GP with the same budget of evaluations, and also GPt with the same CPU time. *Size*: SSHC and SGP produce individuals larger than GP. This is due to a limited amount of simplification applied that finds shorter but usually not the shortest expressions, and, for some problems, to the optimal solution having a long encoding in the chosen representation. Importantly, experiments show that the simplification counteracts effectively the exponential growth of individuals inherent in the semantic operators, within affordable computational cost. *Bias*: despite seeing any problem as a cone landscape, hence potentially easy, semantic operators may have heavy biases in the offspring distributions that hinder performance. Experiments show that these biases do not prevent achieving very good performance.

Table 1: Problems: standard boolean benchmark suite. Hits %: percentage of training examples correctly predicted by best solution; average (avg) and standard deviation (sd) of 30 runs. Length: logarithm base 10 of the length of the largest solution encountered in the search.

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

Table 2: Problems: Random Polynomials of degrees 3 to 10. Hits %: percentage of training examples correctly predicted by best solution with tolerance 0.01; avg and sd of 30 runs.

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

Table 3: Problems: see text. Hits % and Length same as in Table 1.

Problem			**Hits %**								**Length**			
			GP		GPt		SSHC		SGP					
n_v	n_c	n_{cl}	avg	sd	avg	sd	avg	sd	avg	sd	GP	GPt	SSHC	SGP
3	3	2	80.00	8.41	97.30	4.78	99.74	0.93	99.89	0.67	1.6	1.9	2.3	2.3
3	3	4	49.15	9.96	78.89	8.93	99.89	0.67	99.00	1.63	1.6	2.1	2.3	2.3
3	3	8	37.04	5.07	59.52	14.26	99.74	0.93	96.04	2.85	1.2	1.9	2.3	2.3
3	4	2	67.92	7.05	93.80	5.41	99.95	0.28	99.58	0.80	1.8	2.3	2.7	2.7
3	4	4	39.11	7.02	68.48	8.66	99.84	0.47	98.08	1.64	1.7	2.3	2.7	2.7
3	4	8	28.02	3.73	46.98	14.48	99.73	0.58	94.22	1.72	1.1	2.0	2.7	2.7
4	3	2	88.31	6.98	98.89	2.89	99.96	0.22	100.00	0.00	1.6	1.9	2.9	2.9
4	3	4	48.85	6.54	88.15	10.10	100.00	0.00	99.54	0.68	1.4	2.2	2.9	2.9
4	3	8	36.54	9.01	60.37	17.14	100.00	0.00	96.63	1.23	1.0	1.9	2.9	2.9
4	4	2	82.75	8.21	99.79	1.12	100.00	0.00	99.86	0.23	2.2	2.3	3.3	3.3
4	4	4	44.13	8.75	77.55	6.30	100.00	0.00	99.68	0.29	2.0	2.4	3.3	3.3
4	4	8	30.63	5.33	50.21	15.08	99.96	0.12	98.84	0.58	1.4	2.1	3.3	3.3

5 Conclusions and Future Work

We presented a new GP framework rooted in a geometric theory of representations to search *directly* the semantic space of functions/programs. Remarkably, the landscape seen by the semantic operators is *always* a cone by construction, hence easy to search. Seen from a geometric viewpoint, the genotype-phenotype mapping of GP becomes *very easy*, and allows us to derive explicit algorithmic characterization of semantic operators for different domains following a *simple recipe*. Semantic operators require *simplification*, which in practice is not a problem. Experiments showed that the semantic approach *systematically outperforms* standard GP. There is plenty of future work: (i) construct semantic operators for more complex domains, to explore potentialities and limits of the framework; (ii) use formal methods to simplify non-trivial programs with loops/recursion, and use CAS to simplify non-polynomial functions, and, more generally, devise quick heuristic simplifiers for complex domains; (iii) investigate the practical advantages of different types of syntax/languages: e.g., programs written in minimalistic languages with strong theory, like lambda calculus, may be much easier to simplify; also, certain syntax may allow to implement easily semantic operators with probabilistic biases that make them more effective in practice; (iv) derive analytical runtime: as semantic GP search is equivalent to standard GAs/ES on cone landscapes, it should be easy to transfer analytical runtime results to semantic GP, and determine the optimal parameter settings.

References

1. L. Beadle and C. G. Johnson. Sematically driven crossover in genetic programming. In *Proc. of IEEE WCCI '08*, pages 111–116, 2008.
2. L. Beadle and C. G. Johnson. Semantic analysis of program initialisation in genetic programming. *Genetic Programming and Evolvable Machines*, 10(3):307–337, 2009.
3. L. Beadle and C. G. Johnson. Semantically driven mutation in genetic programming. In *Proc. of IEEE CEC '09*, pages 1336–1342, 2009.
4. D. Jackson. Phenotypic diversity in initial genetic programming populations. In *Proc. of EuroGP '10*, pages 98–109, 2010.
5. K. Krawiec and P. Lichocki. Approximating geometric crossover in semantic space. In *Proc. of GECCO '09*, pages 987–994, 2009.
6. K. Krawiec and B. Wieloch. Analysis of semantic modularity for genetic programming. *Foundations of Computing and Decision Sciences*, 34(4):265–285, 2009.
7. S. Luke. *The ECJ Owner's Manual – A User Manual for the ECJ Evolutionary Computation Library*, 2010.
8. A. Moraglio. *Towards a Geometric Unification of Evolutionary Algorithms*. PhD thesis, University of Essex, 2007.
9. A. Moraglio. Abstract convex evolutionary search. In *Proc. of FOGA '11*, pages 151–162, 2011.
10. A. Moraglio and R. Poli. Topological interpretation of crossover. In *Proc. of GECCO '04*, pages 1377–1388, 2004.
11. N. Q. Uy et al. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, 2011.