Semantic Geometric Initialization

Tomasz P. Pawlak(✉) and Krzysztof Krawiec

Institute of Computing Science, Poznan University of Technology, Poznan, Poland
{tpawlak,krawiec}@cs.put.poznan.pl

Abstract. A common approach in Geometric Semantic Genetic Programming (GSGP) is to seed initial populations using conventional, semantic-unaware methods like Ramped Half-and-Half. We formally demonstrate that this may limit GSGP's ability to find a program with the sought semantics. To overcome this issue, we determine the desired properties of geometric-aware semantic initialization and implement them in Semantic Geometric Initialization (SGI) algorithm, which we instantiate for symbolic regression and Boolean function synthesis problems. Properties of SGI and its impact on GSGP search are verified experimentally on nine symbolic regression and nine Boolean function synthesis benchmarks. When assessed experimentally, SGI leads to superior performance of GSGP search: better best-of-run fitness and higher probability of finding the optimal program.

Keywords: Geometric semantic genetic programming · Semantic initialization · Population

1 Introduction

Geometric Semantic Genetic Programming (GSGP) [16] exploits the spatial properties of program semantics in order to improve the effectiveness of program synthesis. The operators proposed within this branch of genetic programming (GP) have well-understood effects in terms of program behavior on tests, and some of them even guarantee producing programs with semantics that remain in certain geometric relationship with the parent(s). As a result, the dynamics of a GSGP search process is in general more predictable than for conventional GP, GSGP methods are often superior in terms of performance [3,4,16,20,23] and lend themselves conveniently to theoretical analysis [17–20,22].

The majority of research effort in GSGP focuses on search operators, which is not surprising given that successful program synthesis is directly contingent on them. However, like in case of other evolutionary computation algorithms, the performance of GSGP depends also on the starting point of a search process, i.e., on the contents of the initial population. It is often assumed that the exploratory capabilities of evolutionary search weaken that dependency. This claim can be however questioned in GSGP, because of the mentioned above more predictable, more 'directional' and targeted behavior of search operators.

M. Heywood et al. (Eds.): EuroGP 2016, LNCS 9594, pp. 261–277, 2016.
DOI: 10.1007/978-3-319-30668-1_17

Our case in point in this study is the convergent character of semantic geometric crossover (SGX). The exact variant of this search operator [16] is guaranteed to produce an offspring with semantics located in the segment connecting parents' semantics. This, as we showed in [20], implies that for a GP run equipped with SGX only, the set of all semantics that can be reached in a search process is determined by the initial population. A scenario in which the initial population precludes arriving at the program with desired semantics (the *target*) is plausible, and the odds for it grow with the dimensionality of semantics (the number of tests (fitness cases)). This risk was largely ignored in past studies on GSGP, in part due to widespread use of mutation along with crossover. Nevertheless, this effect deserves better understanding. Also, as we will argue further, it may be worth addressing this issue even if mutation accompanies crossover as a search operator.

The main contribution of this study is the observation that an alternative (to mutation) remedy to SGX's high susceptibility to initial conditions is to construct the starting population more carefully. We propose Semantic Geometric Initialization (SGI), a semantically aware initialization method that designs the initial population with the search target in mind. A population initialized with SGI is guaranteed to make the target semantics *achievable* with SGX. SGX, due to its stochastic nature and oblivion to target, is still not guaranteed to synthesize the correct program when initialized with SGI; however, such a success becomes much more likely, as we demonstrate in experimental part of this study.

The following Sect. 2 briefly introduces the necessary formalisms. Section 3 uses that formal framework to identify the problem signaled above, i.e., that initial population imposes strict constraints on the set of semantics that can be reached with SGX. Section 4 presents the SGI algorithm and justifies its design. Section 5 argues that SGI is fundamentally different from population initialization methods proposed in the past (including the semantic-aware ones), and Sect. 6 demonstrates SGI's usefulness empirically on a suite of well-known GP problems. Section 7 discusses the main results, and Sect. 8 summarizes this study and outlines the potential follow-up directions.

2 Background

We define a program $p \in \mathcal{P}$ in a programming language $\mathcal{P}$ as a function that maps a set of inputs $\mathcal{I}$ into a set of outputs $\mathcal{O}$, which we denote by $o = p(in)$, where $in \in \mathcal{I}$ and $o \in \mathcal{O}$. We consider only deterministic programs that feature no side effects, nor memory persistent across executions. Semantics $s \in \mathcal{S}$ is a vector $s = [o_1, o_2, \ldots, o_n] \in \mathcal{O}^n$, where we refer to $\mathcal{O}^n$ as *semantic space* (a vector space), and o_i corresponds to ith element in a given n-tuple of program inputs from $\mathcal{I}^n$ that defines the considered program synthesis task. Semantics $s(p)$ of a program p is a vector of p's outputs when executed on a fixed set of inputs $I \subset \mathcal{I}$, i.e., $s(p) = [p(in_1), p(in_2), \ldots, p(in_n)]$, $in_i \in I$.

The concept of semantics allows reasoning about program behavior in terms of n-dimensional spaces. Each program has a well-defined semantics, i.e., a point

in semantic space $\mathcal{O}^n$. The desired outputs given by a specific synthesis problem uniquely determine a point t in that space called *target* (or *target semantics*). If a fitness function happens to be a metric (which is almost always the case in GP), the fitness landscape defined over $\mathcal{O}^n$ is a unimodal cone with the apex in t [22]. These properties open the door to defining spatial relationships between program semantics, and investigating whether particular search operators obey them or not and what is the impact of those properties on the efficiency of search. The concepts of particular importance here are geometric mutation and geometric crossover.

Definition 1. *Given a parent program p, an r-geometric mutation is an operator that produces an offspring p' with semantics $s(p')$ in a ball of radius r centered in $s(p)$, i.e., $\|s(p), s(p')\| \leq r$.*

Definition 2. *Given parent programs p_1, p_2, a geometric crossover is an operator that produces an offspring p' with semantics $s(p')$ in a segment between $s(p_1)$ and $s(p_2)$, i.e., $\|s(p_1), s(p_2)\| = \|s(p_1), s(p')\| + \|s(p'), s(p_2)\|$.*

A crossover operator with the above property has the ideal 'mixing' characteristics: the semantics of the offspring is located 'in between' of parents' semantics. This is in stark contrast to the highly unpredictable semantics of programs produced by conventional search operators (see, e.g., the arguments in [11]).

The quest for practical algorithms that implement geometric search operators lasted for several years. Among others, multiple approximately geometric crossovers have been proposed [10–12,20,23,24]. The breakthrough came with publication of [16], where exact versions of geometric crossover and mutation have been proposed, defined as follows for particular domains.

Definition 3. *(Algorithms for geometric mutation,* Sgm*)* ***Symbolic regression:*** *Given parent arithmetic program p, an offspring is a program $p' = p + r(m_1 - m_2)$, where m_1 and m_2 are random arithmetic programs that output values in range [0,1].* ***Boolean domain:*** *Given Boolean parent program p, an offspring is a program $p' = m \vee p$ with probability 0.5, $p' = \overline{m} \wedge p$ otherwise, where m is a random minterm.*

Definition 4. *(Algorithms for geometric crossover,* Sgx*)* ***Symbolic regression:*** *Given parent arithmetic programs p_1, p_2, an offspring is a program $p' = mp_1 + (1-m)p_2$, where m is a random arithmetic program that returns values in range [0,1].* ***Boolean domain:*** *Given Boolean parent programs p_1, p_2, an offspring is a program $p' = (p_1 \wedge m) \vee (\overline{m} \wedge p_2)$, where m is a random Boolean program.*

3 The Problem

The Sgm and Sgx operators are geometric by construction: the offspring is guaranteed to be geometric with respect to parents in the sense of Definitions 1 and 2. Expectedly, they deliver superior search performance in all domains, as

shown experimentally in [16] and further studies. However, SGM is the key to that success: it is indispensable for good performance, while SGX used alone sometimes fails to converge to the sought program [16,23].

The reason for this state of affairs is the 'centripetal' character of SGX, which can produce only the offspring with semantics in the segment connecting parents' semantics. On one hand, this is highly desirable given that fitness landscape in semantic space is unimodal: an application of SGX to any pair of solutions in that space is guaranteed to produce an offspring with attractive properties – for instance for fitness and operator's metric being Euclidean distance, an offspring that is not worse that the worse of the parents [22]. On the other hand however, if SGX is the only search operator used for program synthesis, the set of semantics achievable from a given initial population $P \subset \mathcal{P}$ is limited to the convex hull spanning the semantics of programs in P, since that convex hull incorporates all segments between semantics of all pairs of programs in P. Formally (cf. [20]):

Theorem 1. *Consider a population P_1 of programs and a search process that starts from P_1 and uses* SGX *to generate subsequent generations of programs. A program having semantics t can be achieved in that search process iff the convex hull of P_1 includes t.*

Proof. Let P_g be population in generation $g \geq 1$, C_g be convex hull of semantics of programs in P_g. For all given pairs of parent programs $p_1, p_2 \in P_g$, a semantics $s(p')$ of an offspring p' is included in a segment of $s(p_1)$ and $s(p_2)$ that in turn is included in C_g. The set of all offspring $P_{g+1} \subseteq C_g$ constitutes a population of generation $g+1$. By the non-decreasing property of convex hull, $C_{g+1} \subseteq C_g$. By induction, $C_{g+1} \subseteq C_g \subseteq ... \subseteq C_1$. Hence, semantics t can be achieved in generation g iff $t \in C_g \subseteq C_1$.

The choice of 't' as the symbol denoting the semantic in question is not incidental: the above theorem applies in particular to the target, with profound consequences. If t happens to be located outside the convex hull, applications of SGX to P, regardless how many, cannot lead to a program with semantics t. Unfortunately, this is relatively likely for populations initialized in conventional, semantic-unaware ways. For instance, for symbolic regression, the semantics of programs generated by means of the popular Ramped Half-and-Half (RHH) method [9] tend to initialize programs with relatively simple semantics, typically clustered around the origin of coordinate system of semantic space [13]. A target located far away from that origin is likely to be outside the convex hull and thus unreachable by the actions of SGX.

This observation, though rarely formalized in the above way in past literature, was one of the reasons for using mutation operators alongside with SGX (the other reason being SGX's propensity to produce large programs). The natural operator of choice is in this context SGM, as it is provably capable of reaching the target from arbitrary starting location in semantic space (even when used alone; see the semantic stochastic hill climber in [16]). However, it may require multiple iterations and by this token produce large programs. In the following, we propose an alternative way of making target achievable for SGX.

4 Semantic Geometric Initialization

In the light of Sect. 3 it becomes evident that the target becomes reachable for SGX once it belongs to the convex hull spanning the semantics of programs in initial population. In this section, we propose Semantic Geometric Initialization (SGI), a method that achieves that goal by means of semantic- and geometry-aware population initialization.

Algorithm 1. Calculation of a set of semantics such that their convex hull in $\mathcal{O}^n$ encloses the target t. *popsize* is desired population size.

```
1:  function WRAP(t, popsize)
2:      S = ∅                                              ▷ Output set
3:      n = |t|                                            ▷ Dimensionality of semantic space
4:      I = {1, ..., n}                                   ▷ Indices of all dimensions
5:      k = 1                                              ▷ Number of dimensions to change in t
6:      while k ≤ n do
7:          for I' ∈ (I choose k) do                       ▷ I' = k-element combination of dimensions
8:              for b ∈ {0,1}^k do                         ▷ b = combination of directions on dimensions in I'
9:                  s ← t
10:                 for i ∈ I' do
11:                     if b_i = 1 then
12:                         s_i ← ADDONE(t_i)
13:                     else
14:                         s_i ← SUBTRACTONE(t_i)
15:                 if s ∉ S then
16:                     S = S ∪ {s}
17:                     if |S| = popsize then
18:                         return S
19:         k ← k + 1
20:     return S                                           ▷ |S| < popsize, population is not filled
```

The input to the method is the target t. The algorithm proceeds in two steps:

1. Use the function WRAP (Algorithm 1) to generate a set of semantics $S \subset \mathcal{O}^n$ such that the convex hull of S encloses target t, i.e., $t \in C(S)$,
2. For each semantics $s_i \in S$, synthesize a program p such that $s(p) = s_i$.

The realization of each step is domain-dependent. For the first step we provide an abstract Algorithm 1 that will be specialized in the following for symbolic regression and Boolean function synthesis domains. It calculates set of semantics that wrap t in their convex hull in $\mathcal{O}^n$ semantic space. The algorithm iteratively constructs new semantics by modifying k components (dimensions) of the target t. For each subset of k components of t, the algorithm attempts to construct 2^k semantics by applying to these components all combinations of two domain-dependent functions: ADDONE and SUBTRACTONE. We gather the resulting

semantics in the output set S while discarding duplicates. We start with $k = 1$ and increment it each time all k-component sets have been considered.

For the **symbolic regression** domain, we define AddOne and SubtractOne functions using arithmetic operations:

$$\textsc{AddOne}(t_{a_i}) \equiv t_{a_i} + 1 \qquad \textsc{SubtractOne}(t_{a_i}) \equiv t_{a_i} - 1$$

The Wrap algorithm, when instantiated with these functions and invoked with *popsize* ≥ 2, is guaranteed to construct a set of semantics S such that $t \in C(S)$. In other words, there exists a dimension i of t and semantics s^1 and s^2, such that $s^1_i < t_i < s^2_i$ and $\forall_{j \neq i} s^1_j = t_j = s^2_j$. Thus, under any metric, t is included in the segment between s^1 and s^2 – a degenerated case of a convex hull.

Concerning program synthesis, for each semantics $s \in S$ calculated by Algorithm 1, Sgi constructs a program using multivariate polynomial interpolation as described in [26]. The set of points used in interpolation comes from the set of program inputs $in \in I$ on which program's semantics is to be calculated and corresponding components of s, i.e., (in_i, s_i), $k = 1, \ldots, n$.

For the **Boolean function synthesis**, the definitions of AddOne and SubtractOne are Boolean counterparts of the above arithmetic operations, limited to the corners of the unit hypercube $\{0, 1\}^n$. Since there are only two values in Boolean domain: 0 (false) and 1 (true), the Boolean addition of 1 results in 1 (i.e., $q \vee 1 \equiv 1$), the Boolean subtraction of 1 results in 0 (i.e., $q \wedge 0 \equiv 0$) for any term, and the functions reduce to constants:

$$\textsc{AddOne}(t_{a_i}) \equiv 1 \qquad \textsc{SubtractOne}(t_{a_i}) \equiv 0$$

For *popsize* ≥ 2 Wrap is guaranteed to include the target t in $C(S)$, because there exists a dimension i of t and semantics s^1 and s^2, such that $s^1_i = t_i \neq s^2_i$ or $s^1_i \neq t_i = s^2_i$ and $\forall_{j \neq i} s^1_j = t_j = s^2_j$. Sgi synthesizes the Boolean programs p_i for semantics $s_i \in S$ using the following formula:

$$p_i = \bigvee_{j=1..n \,:\, t_j = 1} y_j, \text{ where } y_j = \bigwedge_{k=1..n} \begin{cases} x_k & \text{if } in_{jk} \\ \overline{x_k} & \text{if } \overline{in_{jk}} \end{cases}, \tag{1}$$

where in_{jk} is a value of kth variable of jth input used to calculate semantics, y_j is a minterm that is 1 for jth input and x_k is kth program argument.

5 Related Work

To our knowledge, Sgi is the first semantic *and geometric* population initialization method in GP, however not the first semantic method of this kind.

Looks in [13] proposed Semantic Sampling (Ss) heuristic that produces a population of semantically unique Boolean programs with uniformly distributed program sizes. Ss partitions a population into bins by program size and fills them up to assumed capacity by semantically unique programs. Sgi differs from Ss in its awareness of geometry of semantic space. Also, Sgi can operate in any

Table 1. Parameters of evolutionary algorithms.

	Symbolic regression	Boolean domain
Number of runs	30	
Population size	1000	
Termination condition	50 generations or fitness = 0	
Selection method	Tournament selection of size 7	
Fitness function	L_2 metric	L_1 metric
Instructions	$x_1, x_2, +, -, \times, /, \sin, \cos, \exp, \log^a$	$x_1, x_2, \ldots,^b$ and, or, nand, nor

[a] log is defined as $\log|x|$; / returns 0 if divisor is 0.
[b] The number of inputs depends on a particular problem instance

Table 2. Benchmark problems.

	Problem	Definition (formula)	Variables	Range	Size
Symbolic regression	R1	$(x_1+1)^3/(x_1^2-x_1+1)$	1	$\langle -1,1\rangle$	20
	R2	$(x_1^5-3x_1^3+1)/(x_1^2+1)$	1	$\langle -1,1\rangle$	20
	R3	$(x_1^6+x_1^5)/(x_1^4+x_1^3+x_1^2+x_1+1)$	1	$\langle -1,1\rangle$	20
	Kj1	$0.3x_1\sin(2\pi x_1)$	1	$\langle -1,1\rangle$	20
	Kj4	$x_1^3 e^{-x_1}\cos(x_1)\sin(x_1)(\sin^2(x_1)\cos(x_1)-1)$	1	$\langle 0,10\rangle$	20
	Ng9	$\sin(x_1)+\sin(x_2^2)$	2	$\langle 0,1\rangle^2$	100
	Ng12	$x_1^4-x_1^3+\frac{x_2^2}{2}-x_2$	2	$\langle 0,1\rangle^2$	100
	Pg1	$1/(1+x_1^{-4})+1/(1+x_2^{-4})$	2	$\langle -5,5\rangle^2$	100
	Vl1	$e^{-(x_1-1)^2}/(1.2+(x_2-2.5)^2)$	2	$\langle 0,6\rangle^2$	100
	Problem	Instance	Variables		Size
Boolean domain	Even parity	Par5	5		32
		Par6	6		64
		Par7	7		128
	Multiplexer	Mux6	6		64
		Mux11	11		2048
	Majority	Maj7	7		64
		Maj8	8		128
	Comparator	Cmp6	6		64
		Cmp8	8		256

domain for which the AddOne and SubtractOne functions can be defined (and efficiently computed).

Beadle and Johnson [1] proposed Semantically Driven Initialization (Sdi) that fills population with semantically unique programs. Sdi was designed for Boolean and artificial ant problems and uses a reduced ordered binary decision diagram or a sequence of ant moves, respectively, as the representation of program's semantics. Like Ss, Sdi does not engage geometric considerations.

An approach called Behavioral Initialization (Bi) was proposed by Jackson [7]. Bi is a wrapper on Rhh that accepts a program created by Rhh if it is semantically unique in the population being initialized. Although domain-independent, Bi is oblivious to geometry of the semantic space.

Pawlak proposed Competent Initialization (Ci) [20] for symbolic regression and Boolean domains. Ci repetitively invokes Sdi and accepts (i.e., adds to the population) the created program if its semantics expands the convex hull of the semantics of programs already present in the population. Ci is geometric in the limit of population size approaching infinity, but in contrast to Sgi does not guarantee including the target in the convex hull.

6 Experimental Verification

We compare Sgi to Ramped Half-and-Half (Rhh) [9] – the arguably most common population initialization method in GSGP and in GP in general. We run two groups of GSGP setups, with Sgi and with Rhh as the initialization operator, to determine the advantages and disadvantages of using Sgi. In both groups, we consider a setup with Sgx [16] as the only search operator to verify whether inclusion of the target in the convex hull of the initial population increases Sgx's ability of reaching it (cf. Sect. 3). The second setup in each group uses both Sgx and Sgm [16], in a configuration that is most commonly practiced in GSGP. In addition, we run a canonical control setup, with Rhh and traditional non-semantic search operators. Overall, there are thus five setups:

SgiX – Sgi accompanied with Sgx only,
SgiXM – Sgi with Sgx and Sgm in proportions 50 : 50,
RhhX – Rhh with Sgx only,
RhhXM – Rhh with Sgx and Sgm in proportions 50 : 50,
RhhTxTm – Rhh with tree crossover and tree mutation [9] in proportions 90:10.

Table 1 sums up the parameter settings; other parameters are set to ECJ [14] defaults.

We compare the setups on nine uni- and bi-variate symbolic regression problems, and nine Boolean function synthesis problems. The problems come from [15,23] and are summarized Table 2. In univariate symbolic regression problems, 20 Chebyshev nodes[1] [2] are used for training, and 20 uniformly sampled points for testing. For bivariate problems 10 values are picked in analogous way for each input variable and the Cartesian product of them constitutes a data set. Points are selected from the ranges shown in the table. For the Boolean benchmarks, training sets enumerate all combinations of inputs and there are no testing sets.

Training Set Performance. Figure 1 and Table 3 present average and .95-confidence interval of the best-of-generation and the best-of-run program, respectively. Both Sgi* setups begin from a relatively low fitness of about 1 in all problems in both problem domains. This phenomenon originates in the

[1] Points given by $x_k = \frac{1}{2}(a+b)+\frac{1}{2}(b-a)\cos(\frac{2k-1}{2n}\pi), k = 1..n$, where $[a, b]$ is the range of training set, and n is number of data points. Using Chebyshev nodes minimizes the likelihood of Runge's phenomenon [25].

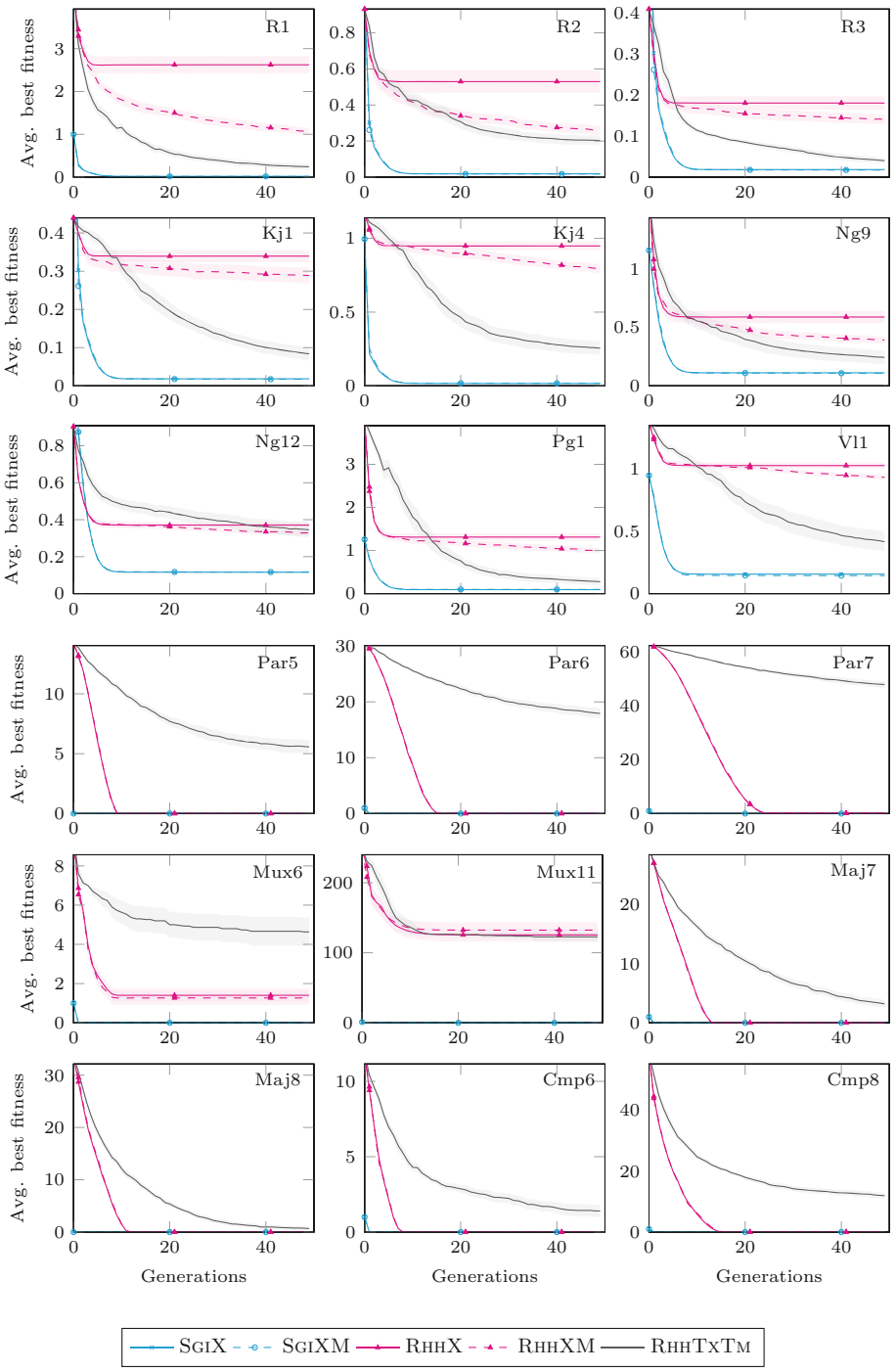


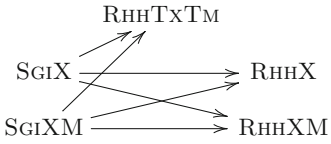
Fig. 1. Average and .95-confidence interval of the best-of-generation fitness.

Table 3. Average and .95-confidence interval of the best-of-run fitness. Last row presents the averaged ranks of setups.

Problem	SgiX	SgiXM	RhhX	RhhXM	RhhTxTm
R1	0.02 ±0.00	**0.02** ±0.00	2.62 ±0.20	1.05 ±0.08	0.24 ±0.07
R2	0.02 ±0.00	**0.02** ±0.00	0.53 ±0.06	0.26 ±0.02	0.20 ±0.03
R3	0.02 ±0.00	**0.02** ±0.00	0.18 ±0.02	0.14 ±0.01	0.04 ±0.01
Kj1	0.02 ±0.00	**0.02** ±0.00	0.34 ±0.02	0.29 ±0.02	0.08 ±0.02
Kj4	**0.02** ±0.00	0.02 ±0.00	0.95 ±0.03	0.79 ±0.03	0.26 ±0.04
Ng9	0.11 ±0.01	**0.11** ±0.01	0.59 ±0.05	0.39 ±0.03	0.24 ±0.06
Ng12	**0.12** ±0.01	0.12 ±0.01	0.37 ±0.02	0.33 ±0.02	0.35 ±0.04
Pg1	**0.09** ±0.00	0.10 ±0.00	1.31 ±0.08	1.00 ±0.06	0.28 ±0.08
Vl1	0.16 ±0.01	**0.14** ±0.01	1.03 ±0.03	0.93 ±0.02	0.42 ±0.08
Par5	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	5.57 ±0.56
Par6	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	17.93 ±0.90
Par7	**0.00** ±0.00	**0.00** ±0.00	0.17 ±0.19	0.10 ±0.11	47.77 ±1.22
Mux6	**0.00** ±0.00	**0.00** ±0.00	1.40 ±0.34	1.27 ±0.37	4.63 ±0.72
Mux11	**0.00** ±0.00	**0.00** ±0.00	125.50 ±9.63	132.33 ±11.25	122.73 ±6.43
Maj7	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	3.20 ±0.62
Maj8	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	0.67 ±0.32
Cmp6	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	**0.00** ±0.00	1.40 ±0.41
Cmp8	**0.00** ±0.00	**0.00** ±0.00	0.03 ±0.06	**0.00** ±0.00	11.93 ±1.08
Rank:	1.89	1.72	4.08	3.36	3.94

Table 4. Post-hoc analysis of Friedman's test on Table 3: p-values of incorrectly judging a setup in a row as achieving better best-of-run fitness than a setup in a column. Significant values ($p < 0.05$) are visualized as outranking graph.

	SgiX	SgiXM	RhhX	RhhXM	RhhTxTm
SgiX		**0.000**	**0.020**	**0.000**	
SgiXM	0.997	**0.000**	**0.006**	**0.000**	
RhhX					
RhhXM		0.567		0.748	
RhhTxTm		0.999			



construction of AddOne and SubtractOne formulas that cause the initial population to consists of multiple programs at distance 1 from the target. These superior starting conditions are especially evident in Boolean domain, where the programs produced by Rhh in the initial generation are 1–2 orders of magnitude worse than those produced by Sgi.

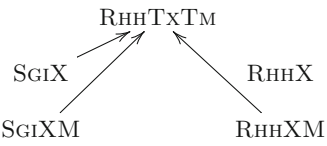
In symbolic regression problems GSGP clearly benefits from Sgi. We observe steep improvement of fitness in the first ten generations that gradually slows down and finally stops after about 20 generations. The initial rate of

Table 5. Probability and .95-confidence interval of success over problems. Problems that were not solved at least once are not shown. Last row presents the averaged ranks of setups.

Problem	SgiX	SgiXM	RhhX	RhhXM	RhhTxTm
Ng9	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	**0.07** ±0.09
Par5	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.00 ±0.00
Par6	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.00 ±0.00
Par7	**1.00** ±0.00	**1.00** ±0.00	0.90 ±0.11	0.90 ±0.11	0.00 ±0.00
Mux6	**1.00** ±0.00	**1.00** ±0.00	0.17 ±0.13	0.23 ±0.15	0.00 ±0.00
Mux11	**1.00** ±0.00	**1.00** ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
Maj7	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.03 ±0.06
Maj8	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.53 ±0.18
Cmp6	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	**1.00** ±0.00	0.30 ±0.16
Cmp8	**1.00** ±0.00	**1.00** ±0.00	0.97 ±0.06	**1.00** ±0.00	0.00 ±0.00
Rank:	2.58	2.58	3.08	2.92	3.83

Table 6. Post-hoc analysis of Friedman's test on Table 5: p-values of incorrectly judging a setup in a row as having higher probability of success than a setup in a column. Significant values ($p < 0.05$) are visualized as outranking graph.

	SgiX	SgiXM	RhhX	RhhXM	RhhTxTm
SgiX		1.000	0.503	0.827	**0.001**
SgiXM			0.503	0.827	**0.001**
RhhX					0.119
RhhXM			0.984		**0.029**
RhhTxTm					



improvement is slower for RhhX and RhhXM, which gradually improve in the first 5–8 generations and then saturate. In Boolean domain both Sgi* setups find the optimum in 1–2 generations, while Rhh* GSGP setups need 10–25 generations to solve all problems, except for multiplexers.

There is no noticeable difference in generational characteristic between both Sgi* setups. In contrast, both Rhh* GSGP setups differ noticeably: RhhXM fares better in most symbolic regression problems. In the Boolean domain, there are no significant differences. Best-of-generation fitness of RhhTxTm is in between those of the GSGP setups for greater part of runs for the symbolic regression problems, and worse than all GSGP setups for the Boolean problems.

Friedman's test [8] on the best-of-run fitness signals significant differences between setups ($p = 9.68 \times 10^{-6}$), so we conduct post-hoc analysis with symmetry test [6]. Table 4 presents the p-values for the hypothesis that a setup in a row is better than a setup in a column, and the graph of significant outrankings. The setups initialized with Sgi are significantly better than all other setups.

Table 5 presents the empirical probability of solving problems by particular setups, which we define as achieving best-of-run fitness lower than 2^{-23}

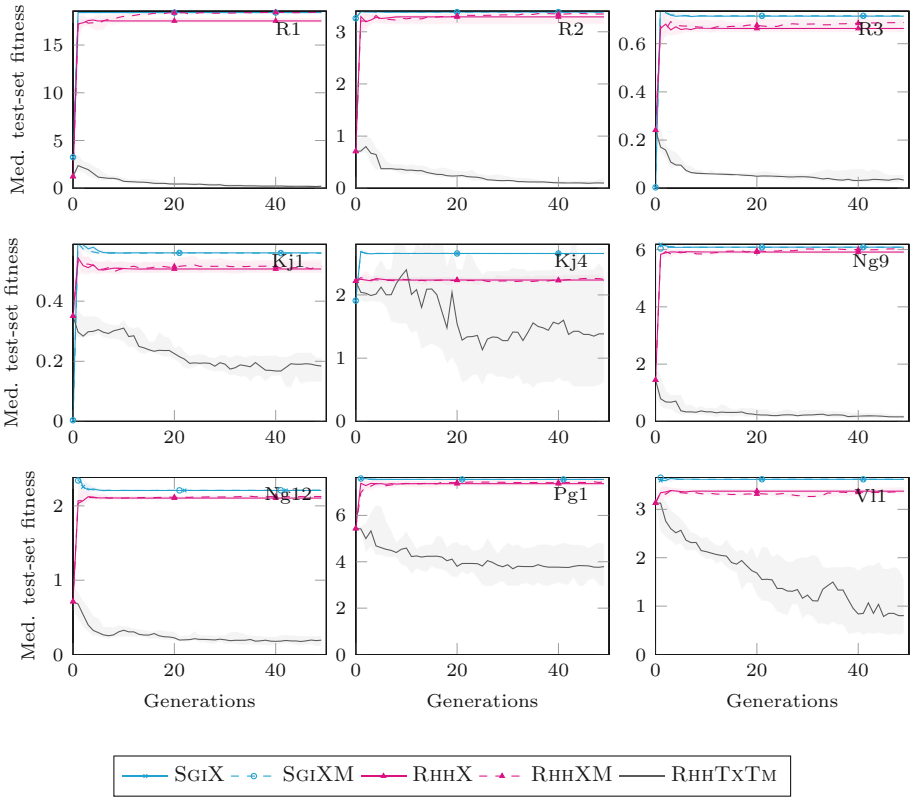


Fig. 2. Median and .95-confidence interval of test set fitness of the best-of-generation program on training set.

Table 7. Median and .95-confidence interval of test set fitness of the best-of-run program on training set. Last row presents the averaged ranks of setups.

Problem	SGIX	SGIXM	RHHX	RHHXM	RHHTXTM
R1	18.40 ±0.00	18.40 ±0.00	17.53 ±0.25	18.39 ±0.12	**0.21** ±0.07
R2	3.38 ±0.00	3.38 ±0.00	3.29 ±0.08	3.34 ±0.02	**0.10** ±0.06
R3	0.71 ±0.00	0.72 ±0.00	0.66 ±0.02	0.69 ±0.01	**0.03** ±0.02
Kj1	0.56 ±0.00	0.56 ±0.00	0.51 ±0.02	0.52 ±0.02	**0.18** ±0.04
Kj4	2.66 ±0.00	2.66 ±0.00	2.24 ±0.03	2.25 ±0.04	**1.38** ±0.89
Ng9	6.08 ±0.00	6.08 ±0.00	5.91 ±0.06	6.02 ±0.03	**0.16** ±0.07
Ng12	2.21 ±0.00	2.21 ±0.00	2.10 ±0.02	2.13 ±0.03	**0.19** ±0.06
Pg1	7.55 ±0.00	7.55 ±0.01	7.37 ±0.10	7.45 ±0.19	**3.79** ±0.90
Vl1	3.62 ±0.01	3.62 ±0.01	3.38 ±0.04	3.37 ±0.06	**0.80** ±0.57
Rank:	4.44	4.56	2.11	2.89	1.00

Table 8. Post-hoc analysis of Friedman's test on Table 7: p-values of incorrectly judging a setup in a row as achieving better test set fitness than a setup in a column. Significant values ($\alpha = 0.05$) visualized as outranking graph.

	SGIX	SGIXM	RHHX	RHHXM	RHHTXTM
SGIX		1.000			
SGIXM					
RHHX	**0.015**	**0.009**		0.835	
RHHXM	0.226	0.166			
RHHTXTM	**0.000**	**0.000**	0.569	0.083	

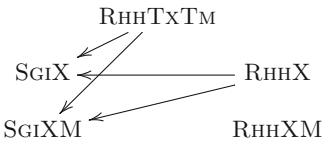
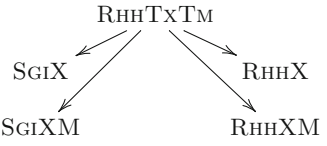


Table 9. Average and .95-confidence interval of number of nodes in the best of run program. Values $\geq 10^4$ are rounded to an order of magnitude. Last row presents the averaged ranks of setups.

Problem	SGIX		SGIXM		RHHX		RHHXM		RHHTXTM	
R1	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{14}	$\pm 10^{14}$	10^{14}	$\pm 10^{14}$	**103.73**	± 13.76
R2	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{14}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**69.90**	± 12.79
R3	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{14}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**123.37**	± 16.20
Kj1	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{14}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**115.70**	± 11.87
Kj4	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{15}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**127.37**	± 14.44
Ng9	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{15}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**65.07**	± 10.01
Ng12	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{15}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**52.50**	± 12.88
Pg1	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{15}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**77.07**	± 10.70
Vl1	10^{17}	$\pm 10^{16}$	10^{17}	$\pm 10^{16}$	10^{14}	$\pm 10^{14}$	10^{15}	$\pm 10^{14}$	**97.23**	± 13.71
Par5	**199.00**	± 0.00	**199.00**	± 0.00	10^4	± 1905.85	10^4	± 1458.59	321.80	± 26.53
Par6	694.90	± 25.58	700.10	± 20.84	10^6	$\pm 10^5$	10^6	$\pm 10^5$	**353.27**	± 34.98
Par7	1458.30	± 42.05	1461.20	± 42.64	10^{15}	$\pm 10^{15}$	10^{15}	$\pm 10^{15}$	**378.27**	± 39.02
Mux6	354.30	± 5.46	354.63	± 9.73	10^{15}	$\pm 10^{15}$	10^{16}	$\pm 10^{15}$	**186.13**	± 30.18
Mux11	9563.97	± 18.39	9416.97	± 219.98	10^{16}	$\pm 10^{14}$	10^{16}	$\pm 10^{15}$	**159.13**	± 19.26
Maj7	1400.20	± 23.91	1401.30	± 24.95	10^5	$\pm 10^4$	10^5	$\pm 10^4$	**406.13**	± 35.64
Maj8	1739.00	± 0.00	1739.00	± 0.00	10^5	± 5010.51	10^5	± 5096.79	**313.20**	± 34.53
Cmp6	579.77	± 13.04	578.87	± 9.58	1899.90	± 294.67	2350.37	± 354.01	**245.73**	± 25.83
Cmp8	2949.40	± 9.08	2892.53	± 38.50	10^{14}	$\pm 10^{14}$	10^5	$\pm 10^5$	**249.87**	± 38.20
Rank:	3.28		3.61		3.17		3.83		1.11	

Table 10. Post-hoc analysis of Friedman's test on Table 9: p-values of incorrectly judging a setup in a row as producing smaller programs than a setup in a column. Significant values ($\alpha = 0.05$) visualized as outranking graph.

	SGIX	SGIXM	RHHX	RHHXM	RHHTXTM
SGIX		0.970		0.828	
SGIXM				0.993	
RHHX	1.000	0.916		0.711	
RHHXM					
RHHTXTM	**0.000**	**0.000**	**0.001**	**0.000**	



(the difference between 1.0 and the closest IEEE754 single precision number). Both Sgi* setups solve all Boolean problems, while the setups using Rhh for population initialization do not solve Mux11 in any run, and do not always solve the other 3 out of 9 Boolean problems. The probabilities are a bit higher for the setup that uses mutation. On the other hand, in symbolic regression, none of the GSGP setups solved any of problems and only Ng9 problem is solved by RhhTxTm. Friedman's test ($p = 6.73 \times 10^{-4}$) and post-hoc analysis in Table 6 show that both Sgi* setups and RhhXM are better than RhhTxTm, however the evidence is too weak to conclude about the differences between Sgi and Rhh in GSGP.

Test-Set Performance. Figure 2 and Table 7 present the median and .95-confidence interval of test-set fitness of the best-of-generation and the best-of-run program on the training set, respectively. All GSGP setups significantly overfit to training data: the test set fitness quickly increases in early generations and remains high for the rest of runs. The values are slightly worse for the Sgi* setups. This observation is consistent with previous studies, [20,23] and can be attributed to the well-known *bias-variance dilemma* [5] in machine learning. The use of Sgi leads to better adaptation to training data, and thus reduces bias. That in turn increases the variance of performance on the unknown test data, and makes GP more prone to overfitting. The conventional RhhTxTm setup is the only one that generalizes well to the test set. Friedman's test ($p = 2.18 \times 10^{-5}$) and post-hoc analysis in Table 8 confirm: RhhTxTm and RhhX are better than both Sgi* setups.

Program Size. Table 9 presents the average and .95-confidence interval of the number of nodes in the best-of-run programs. The exponential growth of Sgx's offspring is clearly visible in the data. For the setups that involve that operator, we report the total number of nodes/instructions in 'unrolled' trees. The actual number of unique program nodes held in memory is many orders of magnitude lower, because a given program may refer to its ancestor programs multiple times, due to the 'aggregative' nature of exact semantic operators (see Definitions 3 and 4).

In contrast, RhhTxTm produces programs smaller than 500 nodes. For the setups initialized with Sgi, it is worth noting that they produce large programs only for problems that they failed to solve in some of runs (cf. Table 5). For the remaining problems, the average number of nodes in a program is smaller, however still bigger than of RhhTxTm. An exception is Par5 problem, for which both Sgi* setups produce the smallest programs. Fortunately, use of Sgi increases the probability of success and thus rises the likelihood of producing small programs. Friedman's test ($p = 2.74 \times 10^{-6}$) and post-hoc analysis in Table 10 confirm that RhhTxTm produces smallest programs, but there is no sufficient evidence to reveal the differences between the remaining setups.

7 Discussion

The results presented above clearly corroborate the importance of population initialization for GSGP. In particular, the geometric and semantic-aware initialization offered by SGI brings to zero the differences of the setups that use and do not use SGM (i.e., SGIX and SGIXM), on virtually all performance indicators (fitness, test-set performance, program size). In other words, the convex hull of semantics built around the target by SGI makes the use of SGM optional. We hypothesize that this characteristics may be convenient in scenarios where SGM is slow at traversing search space, and where SGX may be better in that respect. Concerning low generalization capabilities, they are not due to SGI, but are common to all setups that involve SGM and SGX, and reveal the more fundamental problem of semantic geometric GP, i.e., its inherent lack of bias resulting in high variance [5] (Sec. 6).

A vigilant reader might have noticed the seemingly paradoxical feature of the proposed initialization technique. SGI employs *exact* techniques to synthesize the programs that support the convex hull around the target (multivariate polynomial interpolation for the symbolic regression domain and disjunctive normal forms for the Boolean domain). Then it relies on *heuristic* GP search to synthesize the program that solves the original task, i.e., has semantics in the target. It does not take long to realize that the above exact techniques could be directly used to synthesize the sought program, without using GP altogether.

Note however that the above paradox applies to the entire domain of GP, and not only to SGI or this particular study. Our goal was to verify the relevance of geometric and semantic-aware initialization for search conducted by means of GSGP, and the empirical evidence gathered here confirms the theoretical suppositions. We explore the possibility of one-step construction of a perfect program from a population in another study published in this very volume [21].

SGI offers certain advantages for program size in the Boolean domain. As it follows from Table 9, GSGP starting from traditionally initialized populations (with RHH) may grow monstrously large programs before reaching the target (even when using SGM, which is known to increase program size only by fixed factor in every application, compared to the exponential growth of SGX). When an initial population forms a convex hull around the target, a few moves of SGM and/or SGX may be sufficient to solve a synthesis task. We hypothesize thus that the positive impact of SGI is not only due to its convex hull property, but also due to the proximity of the initial population to the target.

8 Conclusions

We have brought theoretical evidence that the possibility of finding a program with the optimal semantics by GSGP running solely geometric crossover depends on whether the convex hull spanning the programs in the initial population includes the search target. Experimental verification has shown that the above is true also for GSGP equipped with crossover and mutation. The commonly used

RHH initialization does not provide this guarantee. To overcome this problem, we provided the SGI algorithm that seeds the initial population with programs that form an appropriate convex hull.

Further work is needed to develop SGI for other domains than those considered in this paper, e.g., for the categorical one. Another interesting research topic is the influence of the initial distribution of programs' semantics on the analogous distributions in subsequent populations and search performance. Last but not least, the convex hull property is only the *necessary* condition to reach the target. It remains an open question how to efficiently prevent GSGP operators from losing the target from the population's convex hull.

Acknowledgements. T. Pawlak acknowledges support from grant no. DEC-2012/07/N/ST6/03066, K. Krawiec from grant no 2014/15/B/ST6/05205, both funded by the National Science Centre, Poland.

References

1. Beadle, L., Johnson, C.G.: Semantic analysis of program initialisation in genetic programming. Genet. Program Evolvable Mach. **10**(3), 307–337 (2009)
2. Burden, R., Faires, J.: Numerical Analysis. Cengage Learning, Boston (2010)
3. Castelli, M., Henriques, R., Vanneschi, L.: A geometric semantic genetic programming system for the electoral redistricting problem. Neurocomputing **154**, 200–207 (2015)
4. Castelli, M., Vanneschi, L., Silva, S.: Prediction of the unified Parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. Expert Syst. Appl. **41**(10), 4608–4616 (2014)
5. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Comput. **4**(1), 1–58 (1992)
6. Hothorn, T., Hornik, K., van de Wiel, M.A., Zeileis, A.: Package 'coin': conditional inference procedures in a permutation test framework (2015). http://cran.r-project.org/web/packages/coin/coin.pdf
7. Jackson, D.: Phenotypic diversity in initial genetic programming populations. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 98–109. Springer, Heidelberg (2010)
8. Kanji, G.: 100 Statistical Tests. SAGE Publications, London (1999)
9. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
10. Krawiec, K.: Medial crossovers for genetic programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 61–72. Springer, Heidelberg (2012)
11. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: GECCO 2009, 8–12 July 2009, pp. 987–994. ACM, Montreal (2009)
12. Krawiec, K., Pawlak, T.: Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. Genet. Program Evolvable Mach. **14**(1), 31–63 (2013)
13. Looks, M.: On the behavioral diversity of random programs. In: GECCO 2007, 7–11 July 2007, vol. 2, pp. 1636–1642. ACM Press, London (2007)

14. Luke, S.: The ECJ Owner's Manual - A User Manual for the ECJ Evolutionary Computation Library, zeroth edition, October 2010. http://cs.gmu.edu/~eclab/projects/ecj/docs/
15. McDermott, J., White, D.R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., O'Reilly, U.M.: Genetic programming needs better benchmarks. In: GECCO 2012, 7–11 July 2012, pp. 791–798. ACM, Philadelphia, Pennsylvania, USA (2012)
16. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012)
17. Moraglio, A., Mambrini, A.: Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In: GECCO 2013, 6–10 July 2013, pp. 989–996. ACM, Amsterdam, The Netherlands (2013)
18. Moraglio, A., Mambrini, A., Manzoni, L.: Runtime analysis of mutation-based geometric semantic genetic programming on Boolean functions. In: FOGA, 16–20 Jan 2013, pp. 119–132. ACM, Adelaide, Australia (2013)
19. Moraglio, A., Sudholt, D.: Runtime analysis of convex evolutionary search. In: Soule, T., Moore, J.H. (eds.) GECCO, pp. 649–656. ACM (2012)
20. Pawlak, T.P.: Competent algorithms for geometric semantic genetic programming. Ph.D. thesis, Poznan University of Technology, Poznan, Poland, 21 September 2015. http://www.cs.put.poznan.pl/tpawlak/link/?PhD
21. Pawlak, T.P.: Geometric semantic genetic programming is overkill. In: Heywood, M., McDermott, J., Castelli, M., Costa, E., Sim, K. (eds.) EuroGP 2016. LNCS, vol. 9594, pp. 246–260. Springer, Switzerland (2016)
22. Pawlak, T.P., Krawiec, K.: Properties of progress and fitness bounds for geometric semantic genetic programming. Genet. Program Evolvable Mach., 1–19 (2015) (Online first)
23. Pawlak, T.P., Wieloch, B., Krawiec, K.: Review and comparative analysis of geometric semantic crossovers. Genet. Program Evolvable Mach. **16**(3), 351–386 (2015)
24. Pawlak, T.P., Wieloch, B., Krawiec, K.: Semantic backpropagation for designing search operators in genetic programming. IEEE Trans. Evol. Comput. **19**(3), 326–340 (2015)
25. Runge, C.: Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten. Zeitschrift für Mathematik und Physik **46**, 224–243 (1901)
26. Saniee, K.: A Simple Expression for Multivariate Lagrange Interpolation, pp. 1–9. Society for Industrial and Applied Mathematics (2007)