Review and comparative analysis of geometric semantic crossovers

Tomasz P. Pawlak · Bartosz Wieloch ·
Krzysztof Krawiec

Received: 3 July 2014 / Revised: 4 December 2014 / Published online: 23 December 2014
© The Author(s) 2014. This article is published with open access at Springerlink.com

Abstract This paper provides a structured, unified, formal and empirical perspective on all geometric semantic crossover operators proposed so far, including the exact geometric crossover by Moraglio, Krawiec, and Johnson, as well as the approximately geometric operators. We start with presenting the theory of geometric semantic genetic programming, and discuss the implications of geometric operators for the structure of fitness landscape. We prove that geometric semantic crossover *can* by construction produce an offspring that is not worse than the fitter parent, and that under certain conditions such an offspring *is guaranteed* to be not worse than the worse parent. We review all geometric semantic crossover operators presented to date in the literature, and conduct a comprehensive experimental comparison using a tree-based genetic programming framework and a representative suite of nine symbolic regression and nine Boolean function synthesis tasks. We scrutinize the performance (program error and success rate), generalization, computational cost, bloat, population diversity, and the operators' capability to generate geometric offspring. The experiment leads to several interesting conclusions, the primary one being that an operator's capability to produce geometric offspring is positively correlated with performance. The paper is concluded by recommendations regarding the suitability of operators for the particular domains of program induction tasks.

Keywords Geometry · Semantics · Fitness landscape · Crossover · Theory · Experiment

T. P. Pawlak (✉) · B. Wieloch · K. Krawiec
Institute of Computing Science, Poznan University of Technology, Poznan, Poland
e-mail: tpawlak@cs.put.poznan.pl

B. Wieloch
e-mail: bwieloch@cs.put.poznan.pl

K. Krawiec
e-mail: krawiec@cs.put.poznan.pl

1 Introduction

A crossover operator has a special place in evolutionary computation (EC). Conceptually, this search operator is intended to ‘blend’ the parent candidate solutions and produce an offspring that is similar to them [12, 15]. Such *recombination* is intended to support the formation and maintenance of potentially useful solution components (*building blocks*) and the discovery of synergies between them. This characteristic of crossover makes EC quite unique—it can be argued that it was the first widely recognized metaheuristic that featured such capability.

The particular design of crossover obviously depends on the representation of candidate solutions. For solutions represented as vectors of variables, the mixing effect can be achieved by swapping or fusing the values of corresponding (homologous) variables. The popular recombination operators like one-point, two-point, and uniform crossover are based on these premises [15].

The way in which a crossover operator mixes solutions at the genotype level may have substantial impact on the efficiency of evolutionary search. What is, however, even more important, is how such manipulations affect fitness or, more generally, propagate to the functional, phenotypic level. If modifications of individual solution components (e.g., variables) have an independent impact on fitness, i.e., the interactions between them are few and far between, crossover is likely to be helpful. However, if the components are highly interlinked, their modifications affect fitness in a complex, hard to predict way. This characteristic gave rise to several lines of research within EC, including those pertaining to genotype-phenotype mapping [1, 35], linkage [14] and modularity (separable vs. inseparable modules [43]).

The branch of EC that is particularly susceptible to the above problem is genetic programming (GP). In GP, the genotypes are programs, i.e., syntactic structures composed of symbols (instructions), while the actual evaluation concerns the *effects* of computations conducted by programs rather than the programs themselves. The mapping from program code to the effects is particularly complex: a minute variation in a program (e.g., replacement of a single instruction) may result in dramatic changes of the output it produces. On the other hand, a overhauling of program syntax may not affect its computation effects at all.

As a result, designing crossover operators that ‘blend’ program behavior has been notoriously difficult in GP. It was only with the advent of semantic genetic programming (SGP) that this state of affairs started to change. Contrary to the conventional GP where fitness is the only computation effect of interest, SGP brings the detailed effects of program execution to the foreground. SGP methods usually inspect program behavior separately for particular examples, and use the information acquired in this way for the sake of search efficiency (e.g., in the form of semantic-aware search operators [2, 4, 13, 21, 24, 25, 29, 31–33, 38, 39, 44], population initialization [3, 16] or selection [10]). A comprehensive survey of such methods has been recently provided in [40].

The geometric semantic GP (GSGP, [29]) goes even further. Not only does it inspect computation effects individually for particular examples, but it also embeds them in a multidimensional metric space. This formalism has important

implications, in particular helping to design crossover operators with desirable semantic properties (see Sect. 3). As a matter of fact, the pursuit of recombination operators that would mix parents' semantic properties was the main driving force also for the work that predated [29]. However, those older studies resulted in crossover operators that only approximated the desired geometric behavior, while the method proposed in [29] attained that goal. Nevertheless, we will refer to all crossover operators designed with the geometric aspect in mind as 'geometric crossovers'.

The contributions in GSGP crossovers are scattered in multiple papers [21, 24, 25, 29, 33], where they have been presented using different formal backgrounds, tested in various configurations and different benchmark suites, with not necessarily identical parameter settings and software implementations that may have varied in details. The recently published survey [40] neatly categorizes the semantic methods, but does not provide a unified formal framework nor does it empirically compare the methods. In this paper, we consolidate and extend that past work by providing a structured, multi-aspect perspective on, to our knowledge, *all geometric* crossover operators proposed so far. This allows us to bring two major contributions. Firstly, we propose a common formal framework, presented in Sects. 2 and 3, followed by a review of geometric semantic crossover operators in Sect. 4. The second main contribution is a thorough experimental analysis involving various performance indicators (Sect. 5). This analysis includes comparison of conventional performance measures (program error and success rate), as well as generalization, computational cost, bloat, degree of geometricity and its impact on search performance, and diversity analysis. In Sects. 6, 7 and 8 we discuss the conclusions, identify the open issues and challenges, and outline the future research directions.

2 Program semantics in genetic programming

In this section, we provide the formal background for program semantics in GP, in which we attempt to reconcile different formulations proposed in the past works.

Let $p \in P$ be a *program* from a set P of all possible programs. A program is a discrete structure built of atomic instructions. Typically, it is a hierarchy built upon smaller programs, where single instructions are terminal elements of the hierarchy. P is usually implicitly defined by an adopted programming language.

From the formal perspective, a program is a function $p : I \rightarrow O$, where I is the set of inputs and O is the set of outputs. Application of a program p to input datum in , $in \in I$ is denoted as $p(in)$. Such application produces an output $out = p(in)$, $out \in O$.

We consider only programs that halt and have no side effects; i.e., a program is a structure that maps a given input in into an output out and has no state nor memory persistent between particular program executions.

Given this background, we define semantics implicitly by means of semantic space and semantic mapping:

Definition 1 A *semantic space* of a program set P is a set S such that a *semantic mapping* exists for it, i.e., there exists a function $s(\cdot)$ that maps every program $p \in P$ into its *semantics* $s(p) \in S$ and has the following property:

$$s(p_1) = s(p_2) \iff \forall in \in I : p_1(in) = p_2(in)$$

Each program in P has thus exactly one semantics, but two different programs can have the same semantics. Some semantics $x \in S$ can be *infeasible* in the considered set P , i.e., such that $\nexists p \in P : s(p) = x$. Programs having the same semantics will be called *semantically equivalent*.

A semantics can be any formal object that fulfills the above constraint, i.e., such that captures the *final effect* of computation for all considered input instances. Among the formal frameworks known in computer science, certain variants of operational semantics (so-called *big-step* or *natural* semantics) conform to this definition. As it will become clear in the following, the particular formalization of semantics used in the past GP work [5, 10, 13, 21, 25, 29, 33, 38] and adopted in this study meets the above requirement as well.

In GP, a semantics is typically contextualized within a specific *programming task* that is to be solved in a given program set P . A programming task is specified by an objective function f calculated with respect to a set of *fitness cases* $FC \subseteq I \times O$. A *fitness case* is a pair of program input $in \in I$ and a corresponding *desired output* $out \in O$. In this sense, FC forms the *training set* of the programming task. The tuple of desired outputs of cases in FC forms the *target* of a programming task. We assume that the inputs are unique in FC , and thus FC defines a function and a programming task features a single target.

Note that by requiring the desired outputs (the target) to be explicitly given for every fitness case, we leave out the GP tasks in which such information is not available and the output of a program is only implicitly evaluated (e.g., control problems, like inverted pendulum or games). We adopt this assumption for consistency with most of the earlier works on semantic GP [2, 13, 21, 25, 33, 38, 39], and also to ease the understanding of the properties of semantic space. However, as it will become clear in the further reading, none of the crossover operators considered here requires explicit access to the desired outputs. They only assume that a unique desired output exists for every fitness case, and that the fitness function measures a distance of such outputs from those desired outputs. If these conditions hold for a given problem (be it a control problem or a game), each of the considered operators can be legitimately applied.

Given the above formalization of a programming task, we can propose a GP-specific definition of semantics (and adopt it for the rest of this paper):

Definition 2 A *semantics in a programming task* (FC, f) that features n fitness cases (i.e., $|FC| = n$) is an n -tuple of elements from O .

A semantics is thus any tuple of values from O , where the elements of the tuple correspond to particular fitness cases in FC . The set of all such tuples forms the abovementioned semantic space S , with some of the tuples being feasible, and some

infeasible semantics. For instance, in a real-valued symbolic regression task, a semantics is an n -tuple of real numbers, i.e., a point in $\mathbb{R}^n$.

This fitness case-based definition of semantics can be used to express the desired program output within a particular programming task or to capture the *actual* output of a given program. In the latter case, we talk about *program semantics*.

Definition 3 A semantics $s(p)$ of a program p (program semantics) is an n -tuple of outputs $p(in)$ produced by p for all fitness cases $(in, out) \in FC$:

$$s(p) = (p(in))_{(in, out) \in FC} = (p(in_1), p(in_2), \dots, p(in_n))$$

When the output values are numerical, $s(p)$ is formally a *vector* [10, 24, 25, 33, 38, 39]. However, in the following we operate on tuples to emphasize that programs can in general return arbitrary data types.

By delineating semantics and program semantics, we can manipulate a semantics without knowing if a program with that semantics exists. We can even synthesize completely ‘artificial’ semantics if needed. This will ease our further analysis, in particular in the case of search operators like crossover.

Let us now define the last component of a programming task, i.e., the fitness function. Contrary to the common convention which assumes that its domain is the space of programs P , we assume that it assigns fitness to the elements of the semantic space S .

Definition 4 A fitness function $f : S \rightarrow \mathbb{R}_{\geq 0}$ is a distance between the semantics s and the target t of the associated programming task (FC, f) , i.e.:

$$f(s) = d(s, t) \quad (1)$$

where $d(\cdot)$ is some distance metric. In particular, f can be indirectly applied to programs, i.e., $f(s(p))$. The fitness function defined in this way captures the disparity between any semantics and the desired semantics (the target), and our objective will be thus to minimize it.

It becomes clear now why S is referred to as semantic *space* (rather than simply a set): it is being endowed with a structure by f . If the elements of O are numbers, semantics in S and the target t become *vectors*, and S becomes a vector space. This structure has fundamental implications for the search process, which we elaborate on in the next section.

3 Geometric structure of the semantic space

Defining the fitness function as a distance (Def. 4) is natural and has been practiced in GP from its infancy. It is only recently that, with the advent of semantic GP, the profound implications of this fact have been widely realized.

With S being a metric space, the fitness at any point in that space is the distance from the target, i.e., from the desired semantics as specified by the programming task. Thus, the fitness function spanning S has a conic shape. As a consequence, it is

unimodal (f assumes the minimum value of 0 at exactly one point in S , i.e., at t) and does not feature plateaus. This holds for any semantic space, any data type associated with the output set O , and any metric (albeit the cone may have an unintuitive interpretation in non-Euclidean spaces). A visualization of fitness function spanning S under the city-block and the Euclidean metrics is presented in Fig. 1.

Minimizing f in such a rigorously structured space should be in principle easy. But in general it is not, the reason being that the semantic space S is *not the space being searched*. The search space is the set of programs P , because it is programs, not semantics, that are constructed and manipulated by the search operators. The semantic space is searched only implicitly, intermediated by the semantic mapping s . As a result, applying a minor change to a program in P can correspond to a big leap in S and, conversely, a major change of a program can leave its semantics intact. Moreover, since some semantics in S can be infeasible in P , the cone may feature ‘holes’.

In other words, the shape of the fitness function surface ceases to be a cone when spanned over P rather than over S . Once a specific search operator is adopted, it endows P with a certain structure, and determines a *fitness landscape*, i.e., the surface of f graphed against P [45]. The fitness landscape may feature multiple global and local optima, as well as plateaus.

Despite its intricate relationship with the fitness landscape, the properties of the semantic space help formulate guidelines for designing search operators. In particular, we can attempt to determine the desired semantic properties of the outcome of a search operator (even if designing a search operator with such properties is technically difficult or impossible). The most prominent attempt to formalize such properties is [29], where the class of *semantic geometric* search operators has been delineated, and specific examples thereof have been proposed.

In this paper, we focus on the geometric crossover operators, so it is desirable to redefine the notion of geometricity after [29].

Fig. 1 The conic shapes of fitness functions under the city-block and Euclidean metrics. The diamond and circle shapes mark the isoquants (level sets), i.e., the semantics with the same fitness

Definition 5 An offspring program $p = c(p_1, p_2)$ resulting from an application of a crossover operator $c : P \times P \rightarrow P$ to a pair of parent programs (p_1, p_2) is *geometric with respect to* p_1 and p_2 under metric d iff its semantics is located in the d -metric segment connecting the semantics of its parents, i.e.,

$$d(s(p_1), s(p_2)) = d(s(p_1), s(p)) + d(s(p), s(p_2)). \quad (2)$$

A crossover operator that guarantees (2) for any pair of parents and the resulting offspring is called a *geometric semantic crossover*, or *geometric crossover* for short.

This definition formalizes the property that is widely considered desirable in crossover, namely that an offspring should have some traits in common with both of its parents. In semantic GP, the traits of a program p can be identified with the output it produces for particular fitness cases, i.e., the elements of its semantics $s(p)$. In this sense, inheritance takes place here at the level of program semantics: by belonging to the segment (2), an offspring is guaranteed to minimize the total distance (dissimilarity) to its parents in the semantic space. Note that this is unrelated to the equidistance from the parents, which has been studied elsewhere [20, 21, 28].

The use of the same metric d in the definition of geometric offspring and in the definition of fitness is not incidental. As it will become clear in the following, this is essential for the geometric semantic crossover to guarantee a certain form of progress. We will now demonstrate this guarantee for the continuous semantic space $\mathbb{R}^n$ and two common metrics, Euclidean and city-block. To simplify our discourse and notation, we will assume that the target $t = \mathbf{0}$. As this can be always achieved by translating the original semantic space by $-t$, no loss of generality is incurred.

For both the Euclidean and city-block metrics, we consider a geometric offspring p of parent programs p_1 and p_2 , i.e., located in the d -segment between the parents' semantics. This condition can be written as

$$s_i(p) = \alpha_i s_i(p_1) + (1 - \alpha_i) s_i(p_2) \quad (3)$$

where $s_i(p)$ denotes the i th element of p 's semantics, $1 \leq i \leq n$, and $\alpha_i \in [0, 1]$ for both Euclidean and city-block distances. For the Euclidean distance, additionally $\alpha_1 = \alpha_2 = \dots = \alpha_n$ holds.

The fitness of an offspring is $f(p) = d(s(p), t) = d(s(p), \mathbf{0})$, which for d written as an L_z metric is

$$f(p) = d(s(p), t) = \sqrt[z]{\sum_i |s_i(p) - t_i|^z} = \sqrt[z]{\sum_i |s_i(p)|^z}, \quad (4)$$

where $z = 1$ for the city-block distance (L_1) and $z = 2$ for the Euclidean distance (L_2). We are interested in finding the minimal (best) value of (4) within segment (3). As the root is monotonic, we can equivalently seek $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ such that

$$\arg \min_{\alpha} \sum_i |\alpha_i s_i(p_1) + (1 - \alpha_i) s_i(p_2)|^z \quad (5)$$

Given specific $s(p_1)$ and $s(p_2)$, expression (5) is a piecewise linear function of α_i s for L_1 , or a polynomial of degree 2 of α_i s for L_2 (in which case the absolute value operator can be dropped). Let us now consider these two metrics separately.

For L_1 , each term under the sum symbol is a piecewise linear function of α_i and can be minimized independently. For every dimension i , if $s_i(p_1)s_i(p_2) \geq 0$, such a term reaches its minimum either for $\alpha_i = 0$ or $\alpha_i = 1$, and such a minimum amounts to $s_i(p_2)$ or $s_i(p_1)$, respectively. Otherwise, a value of $\alpha_i \in (0, 1)$ exists such that the term reaches zero.

Most importantly, only if all $\alpha_i = 0$ or all $\alpha_i = 1$, the offspring's semantics $s(p)$ overlaps with one of the parents (cf. (3)), and has thus the same fitness as that parent. In any other case, the minimizing combination of α_i s either picks, on each dimension i , the smaller value (in absolute terms) from one of the parents, or combines them linearly to reach zero, and thus brings no contribution to the overall distance. In such cases, $f(p) < \min(f(p_1), f(p_2))$, i.e., the offspring is strictly better than both parents.

For L_2 , expression (5) is a quadratic function of α_i s, but because all α_i s are equal, there is effectively only one free variable α and (5) is a parabola. Analogously to the L_1 metric, two scenarios are possible. The expression can be minimized at one of the extremes ($\alpha = 0$ or $\alpha = 1$), in which case the offspring is semantically equivalent to one of the parents ($s(p) = s(p_1)$ or $s(p) = s(p_2)$) and receives the same fitness. Otherwise, by the convexity of parabola, the minimum has to be smaller than the fitness of both parents, i.e., $f(p) < \min(f(p_1), f(p_2))$.

We have shown that, for L_1 and L_2 , the set of all geometric offspring always contains an offspring that is at least as fit as the better of the parents. But more importantly, we also demonstrated that geometric offspring may exist that are *strictly better* than both parents. It is these two properties combined that make geometric crossover so attractive: the former makes it likely to produce an offspring that maintains the quality of parents, while the latter gives it a chance to make progress in search.

However, in general geometric offspring does not *guarantee* progress. Of the two metrics considered above, only L_2 guarantees progress in a weak sense, i.e., the offspring cannot be *worse* than the worse of the parents (because, by the convexity of parabola, no α exists that would cause $f(p)$ to exceed $\max(f(p_1), f(p_2))$). For the L_1 metric, such a worse geometric offspring can exist—which can be trivially demonstrated by replacing minimization with maximization in (5)—in that case α_i s may exist that cause $f(p)$ to exceed $\max(f(p_1), f(p_2))$.

Similar analysis has been conducted by Moraglio et al. [30], who derived a probabilistic model of improving the worst fitness in the population and reaching the target. Here, we focused on bounds of fitness change in a single application of the geometric crossover operator.

The overall conclusion of the above considerations is that an evolutionary search equipped only with a geometric crossover produces offspring located between parents' semantics. This implies that, at the level of the entire population, the semantics of the offspring are located inside the *convex hull* that spans the semantics of parent programs.

4 Review of the geometric crossover operators

In the previous section, we showed that geometric offspring have the potential of surpassing the fitness of their parents while maintaining the semantic similarity to them. This characteristic, though rarely formalized to the extent shown above, initiated the research on crossover operators capable of producing geometric offspring. That, however, proved to be difficult due to the fact that GP search takes place in the space of programs, which is related to the semantic space in an intricate way. Most of the proposed operators do not guarantee the offspring to be geometric, but only strive to do so. Despite this, we will for brevity refer to them as geometric operators.

In this section, we review all, to the best of our knowledge, geometric crossovers for tree-based GP, and put them in the formal context provided in the previous section. It is worth noting that geometry is only one of the aspects of semantic GP (albeit arguably the most important one). In the first study on non-geometric semantic crossover [2], the motivation was to prevent production of Boolean programs that were semantically equivalent to parents. A similar rationale was behind the work of Quang et al., conducted in the domain of symbolic regression [38]. The authors proposed two crossover operators: a simpler one, which allows crossing over only the subtrees of parent programs that are sufficiently distanced; and a more complex one, that allows the crossover act only if the distance between parents' subtrees is in a given range. In [39], the same team conducted a study on the importance of locality in semantic crossover and proposed a crossover operator that breeds multiple offspring and selects the one being the most similar to any of the parents but different from both of them. Hara et al. [13] proposed a crossover operator that picks subtrees to be crossed over according to a probability distribution defined on semantic distance between the subtrees.

Research efforts in semantic GP also included mutation. For instance, authors in [4] proposed a mutation operator for Boolean problems that forces the offspring to be semantically different from its parent. Nguyen et al. [31] brought the same idea to the symbolic regression domain, however they compare semantics at the point of mutation. Finally, research on generic and operator-independent methods was conducted in [6].

To the best of our knowledge, the first approximately geometric semantic crossover was proposed in [21]. Later in [29], an exact geometric crossover was proposed that guarantees the offspring to be strictly geometric in the sense of Definition 5. Recently, [25] introduced a locally geometric semantic crossover, which approximates geometric crossover 'locally', on the level of subprograms. Finally, in [33] an operator was proposed that approximates the geometric recombination by propagating the desired semantics of an offspring through the parent's program tree. These last operators are the subject of this study and will be detailed in the following.

4.1 Krawiec and Lichocki geometric crossover

The Krawiec and Lichocki Geometric Crossover (KLX) can be considered a form of brood selection [37]. In its operation, KLX relies on an arbitrary *base crossover operator* to generate an approximately geometric offspring. Given two parent program trees p_1 and p_2 , KLX applies the base crossover operator k times to them and stores the resulting *candidate offspring* in a breeding pool. Next, it calculates the following expression for every candidate offspring p :

$$\underbrace{d(s(p_1), s(p)) + d(s(p), s(p_2))}_{\text{distance sum}} + \underbrace{|d(s(p_1), s(p)) - d(s(p), s(p_2))|}_{\text{penalty}}, \quad (6)$$

and returns as the final offspring two candidates that provide the lowest values of this indicator. Following [21], in the experimental part we employ the tree swapping crossover [19] as the base crossover operator.

The *distance sum* term in Formula (6) captures the ‘degree of geometricity’ of the offspring, and achieves the minimum for the strictly geometric offspring [cf. Eq. (2)]. This minimum is achieved by parents’ semantics too, so minimizing solely this term would promote picking the candidate offspring that are parents’ clones, which are quite likely to be present in the breeding pool. As cloning parents renders crossover ineffective, Formula 6 also involves the *penalty* term, which promotes the offspring that is equidistant or close-to-equidistant from both parents (in the semantic sense). An ideal offspring according to (6) is thus a program that is both geometric with respect to its parents and equidistant from them. The reader is referred to [21] for a more detailed description of KLX (called SX+ in that paper).¹

4.2 Exact geometric semantic crossover

Geometric semantic genetic programming [29] is a semantically-grounded approach to designing exact geometric genetic operators, including crossover and mutation. Since mutation is out of the scope of this study, we focus on crossover only, and refer to it as Semantic Geometric Crossover (SGX).

Given two parent programs p_1, p_2 , SGX randomly generates a random program p_r , and then combines p_1, p_2 and p_r into an offspring p using one of the following formulas, depending on the problem domain:

$$p = (p_1 \wedge p_r) \vee (\overline{p_r} \wedge p_2) \quad (\text{Boolean domain}) \quad (7)$$

$$p = (p_1 \times p_r) + ((1 - p_r) \times p_2) \quad (\text{regression domain}) \quad (8)$$

This construction of offspring takes place at the level of expression trees, so the symbols aggregating the parents above ($\wedge, \vee, \neg, \times, +, -$) are assumed to be available in the underlying instruction set (notice the absence of semantic mapping in the above formula). For the regression domain, p_r must return a value in $[0, 1]$ for every

¹ To prevent confusion with semantic crossover (SX), in the text to follow we replace the original SX+ acronym with KLX.

fitness case to cause p 's semantics to be located on the segment between $s(p_1)$ and $s(p_2)$ (c.f. the analysis at the end of Sect. 3). If the metric d used in the fitness function (Eq. (1)) is L_2 , p_r has to return the same value in $[0, 1]$ for all fitness cases; for L_1 , it can vary across the fitness cases. For the Boolean domain (which uses Hamming distance as the fitness function), no constraints are imposed on p_r .

Comparison of Eqs. (7) and (8) with the definition of geometric offspring [Eq. (2)] leads to an immediate conclusion that SGX is guaranteed to produce an exactly geometric offspring. For consistency with the other operators presented here, we assume that each application of SGX produces two offspring, the second one by swapping p_1 and p_2 in Eqs. (7) and (8).

Since it produces a strictly geometric offspring, SGX might be thought to be the pinnacle of semantic GP. However, the offspring program is on average roughly two times larger than its parents. Specifically, the average number of nodes in a program tree is given by the formula:

$$\overline{p_0} + (2^g - 1)(\overline{p_0} + \overline{p_r}), \quad (9)$$

where $\overline{p_0}$ is the average number of nodes in a program in the initial population, $\overline{p_r}$ is the average number of nodes in a random program p_r [cf. Eqs. (7) and (8)], and g is the generation number. Thus, programs clearly grow exponentially with time.

To work around this problem, Moraglio et al. [29] proposed to simplify the offspring of every crossover act. Indeed, efficient simplification procedures exist for the constant-length genotypes employed there (disjunctive normal forms for the Boolean domain and vectors of polynomial coefficients for the symbolic regression domain). Unfortunately, for the tree-based programs considered in this study simplification is known to be NP-hard [8]. Using an exact simplification procedure is thus technically infeasible in many real-world applications.

To mitigate this problem, Castelli et al. [5] proposed an approach that implicitly transforms a population of programs into a directed acyclic graph. Because the parent programs get incorporated into the offspring without being modified, SGX can be applied multiple times to the same parents and over the course of multiple generations, without ever copying the parents' code. Technically, it is enough for the offspring to *refer* to parents' code rather than to copy it. The only new code added in every generation is the expressions that implement Eqs. (7) and (8) and the subprograms p_r . As a result, the technique introduced in [5] causes the memory consumption to grow only linearly with the number of generations.

4.3 Locally geometric semantic crossover

Locally geometric semantic crossover (LGX) proposed in [22, 23] and analyzed in [25], combines the ideas of homologous and geometric crossover. The working principle of this operator is an approximate geometric recombination at the level of homologous subprograms in the parents, in expectation that geometric changes propagate towards tree roots and so cause the offspring to be approximately geometric. The motivation for focusing on subprograms is that an approximately geometric subprogram can be efficiently retrieved from a previously prepared library of small programs.

An illustrative example of LGX applied to two parent programs p_1 and p_2 is presented in Fig. 2. First, LGX calculates the *common region*, i.e., the set of loci that occur in both of them (informally, the common part of the *shapes* of both parent program trees [34]). Then, a crossover point is drawn at random from the common region. Next, for the subtrees p'_1 and p'_2 rooted at the crossover points in both parent trees, LGX calculates the *desired semantics* s_D as the midpoint between their semantics $s(p'_1)$ and $s(p'_2)$:

$$s_D = (s(p'_1) \wedge s_r) \vee (\overline{s_r} \wedge s(p'_2)) \quad (\text{Boolean domain}) \quad (10)$$

$$s_D = \frac{1}{2}(s(p'_1) + s(p'_2)) \quad (\text{regression domain}) \quad (11)$$

where s_r is a randomly generated semantics such that the numbers of bits in s_D taken from each parent are equal (concerning only the bits that differ). Then, a *library* of previously prepared programs (typically small-sized trees) is sought for a program with the semantics most similar to s_D . Finally, LGX implants the found program into the crossover points in both parents, returning two offspring.

More details on LGX can be found in [25].

4.4 Approximately geometric crossover

Approximately Geometric Crossover (AGX), proposed in [24, 33] attempts to produce a geometric offspring by *semantic backpropagation* of geometric semantics through parents' programs.

Fig. 2 An exemplary application of locally geometric semantic crossover

Fig. 3 An example of semantic backpropagation

Given two parent program trees, p'_1 and p'_2 , AGX employs the same formulas as LGX [Eqs. (10) and (11)] to calculate the desired midpoint semantics s_D in the segment connecting $s(p_1)$ and $s(p_2)$. The subsequent steps are performed for each parent separately. A crossover point is first randomly drawn in the parent. Then, s_D is propagated backwards through the path from the parent's root node to the crossover point. In a single step of this process, the instruction's argument is calculated in such a way that it causes the instruction to return the desired semantics (Fig. 3). For instance, given the expression $(1 - x)$ and the desired value -2 (for a specific fitness case), the desired value to be propagated to subtree x would be 3 , as this value causes the expression to return the value of -2 . This process descends recursively, node by node, along the path to the crossover point, resulting in the desired semantics s'_D for the crossover point. Next, a library of programs is searched for a program having the semantics that is the most similar to s'_D . The best-matching program then replaces the subtree rooted at the crossover point, creating the offspring.

The details and a more thorough discussion of AGX operator and semantic backpropagation can be found in [33].

5 Experiment

In the second part of this study we present a comprehensive experimental comparison of geometric crossover operators. In addition to the operators presented in Sect. 4, we include two control setups: the canonical tree-swapping crossover (GPX) [19] and the tree-swapping crossover combined with subtree mutation [19] with proportions 90 : 10% (GPX90). GPX90 is thus the only setup that explicitly involves a mutation operator—all other setups use only crossover to search the program space. If not stated otherwise, experiments use the configuration shown in Table 1.

Though an attempt has been made to make the configurations of particular methods as similar as possible, some differences could not be avoided. For instance, AGX, LGX, and GPX select the nodes to be crossed-over uniformly with respect to

Table 1 Parameters of evolutionary runs

Parameter	Value
Population size	1,024
Fitness function	<i>Symbolic regression</i> : Euclidean distance <i>Boolean domain</i> : Hamming distance
Termination condition	At least 100 generations and 100 s of CPU time or find of individual having fitness 0
Initialization method	Ramped Half-and-Half algorithm, height range 2–6
Duplicate retries	100 (until accepting a syntactic duplicated individual)
Selection method	Tournament selection, tournament size 7
Max program height	17 (except for SGX, which has no limit—see Sect. 4.2)
Node selection	SGX: Root only AGX, KLX, GPX: Equal depth probability ^a LGX: Homologous selection
Crossover probability	1.0
Type of library	AGX, LGX: Population-based library, other: n/a
Number of retries	KLX: 10, other: n/a
Instructions	<i>Symbolic regression</i> : x , $+$, $-$, $\times$, $/$, $\sin$, $\cos$, $\exp$, $\log$, ERC <i>Boolean domain</i> : D1 , D11^c , and , or , nand , nor , ERC^b
Number of runs	30

^a Selection draws uniformly a random number r from the interval $[1, \text{height}(p)]$, where $\text{height}(p)$ is the height of the program tree p . Then, from all nodes at depth r in p , it selects a random node with uniform probability and returns it

^b $\log$ is defined as $\log |x|$; $/$ returns 0 if divisor is 0

^c The number of inputs depends on a particular problem instance

the depth of a node in a tree. On the other hand, SGX always uses programs' root nodes by design. Similarly, the homologous node selection is an inherent part of the LGX algorithm [25].

SGX uses random program trees p_r [cf. Eqs. (7) and (8)] of height at most 3.

AGX and LGX make use of a library of programs with known semantics. In our previous study [33], they were compared using two types of library: a precomputed static library, and a dynamic, population-based library. The experiments indicated the population-based library as computationally less demanding, so AGX and LGX rely on it in the experiments described below. The library is simply the set of all subtrees (subprograms) present in the current population.

We consider two problem domains: symbolic regression and Boolean function synthesis. In consistency with the formalisms introduced in Sect. 2, and adhering to the practice that is common in GP, we pose the programming tasks as single-objective problems, using fitness minimization as the only objective.² The

² Though program size is ultimately important, we do not consider it as an objective here, as the geometric semantic framework is not prepared to embrace it explicitly. Though this does not preclude one from controlling the program size using other means (like, e.g., lexicographic parsimony pressure), we do not apply them here to avoid possible 'interference' with the semantic crossover operators.

Table 2 Benchmarks

Problem	Definition (formula)	Range	
<i>(a) Symbolic regression benchmarks</i>			
Septic	$x^7 - 2x^6 + x^5 - x^4 + x^3 - 2x^2 + x$	$[-1, 1]$	
Nonic	$\sum_{i=1}^9 x^i$	$[-1, 1]$	
R1	$(x + 1)^3 / (x^2 - x + 1)$	$[-1, 1]$	
R2	$(x^5 - 3x^3 + 1) / (x^2 + 1)$	$[-1, 1]$	
R3	$(x^6 + x^5) / (x^4 + x^3 + x^2 + x + 1)$	$[-1, 1]$	
Nguyen6	$\sin(x) + \sin(x + x^2)$	$[-1, 1]$	
Nguyen7	$\log(x + 1) + \log(x^2 + 1)$	$[0, 2]$	
Keijzer1	$0.3x \sin(2\pi x)$	$[-1, 1]$	
Keijzer4	$x^3 e^{-x} \cos x \sin x (\sin^2 x \cos x - 1)$	$[0, 10]$	
Problem	Instance	Bits	Fitness cases
<i>(b) Boolean benchmarks</i>			
Even parity	PAR5	5	32
	PAR6	6	64
	PAR7	7	128
Multiplexer	MUX6	6	64
	MUX11	11	2,048
Majority	MAJ7	7	128
	MAJ8	8	256
Comparator	CMP6	6	64
	CMP8	8	256

For symbolic regression the training set consists of 20 fitness cases distributed *equidistantly* in the given range, whereas the test set consists of 20 points drawn *uniformly* from the same range

benchmarks come from [19, 27, 41] and are presented in Table 2. There are nine benchmarks in each domain. For symbolic regression, the target semantics is defined by 20 equidistant points in the assumed range of program output; for the test set, the target semantics is defined by 20 randomly drawn points from the same range for each benchmark separately. For the Boolean benchmarks, the target semantics determines the desired output for all possible combinations of program inputs. Therefore, the number of fitness cases is problem-dependent and there is no test set in this domain.

The Java source code of the experimental suite including benchmarks and implementation of crossover algorithms is available online at <http://www.cs.put.poznan.pl/tpawlak/link/?GPGeoCrossOverReview>.

5.1 Performance of the operators

Figure 4 presents the best-of-generation fitness averaged over 30 evolutionary runs, accompanied by a 95 % confidence interval. The best-of-run results are provided in detail in Table 3. SGX clearly converges very fast, usually in less than 25

Fig. 4 Average and 95 % confidence interval of the best-of-generation fitness

Table 3 Average fitness and 95 % confidence interval achieved by the best of run individual as of 100th generation

Problem	SGX	AGX	LGX	KLX	GPX	GPX90
Keijzer1	0.256 ± 0.011	0.001 ± 0.000	0.032 ± 0.019	0.134 ± 0.022	0.074 ± 0.024	0.067 ± 0.018
Keijzer4	0.786 ± 0.087	0.010 ± 0.007	0.140 ± 0.038	0.455 ± 0.093	0.183 ± 0.040	0.178 ± 0.041
Nguyen6	0.137 ± 0.025	0.001 ± 0.000	0.002 ± 0.001	0.027 ± 0.013	0.021 ± 0.008	0.024 ± 0.010
Nguyen7	0.008 ± 0.003	0.000 ± 0.000	0.010 ± 0.011	0.052 ± 0.014	0.028 ± 0.012	0.021 ± 0.011
Nomic	1.051 ± 0.262	0.005 ± 0.001	0.041 ± 0.015	0.223 ± 0.033	0.158 ± 0.043	0.167 ± 0.030
R1	1.012 ± 0.170	0.005 ± 0.003	0.014 ± 0.007	0.177 ± 0.041	0.141 ± 0.036	0.179 ± 0.049
R2	0.098 ± 0.015	0.003 ± 0.001	0.034 ± 0.010	0.163 ± 0.032	0.186 ± 0.034	0.159 ± 0.031
R3	0.042 ± 0.005	0.001 ± 0.000	0.003 ± 0.001	0.038 ± 0.007	0.020 ± 0.004	0.021 ± 0.006
Septic	2.709 ± 0.667	0.004 ± 0.003	0.023 ± 0.006	0.266 ± 0.052	0.210 ± 0.042	0.193 ± 0.045
CMP6	0.000 ± 0.000	0.267 ± 0.205	0.300 ± 0.188	0.533 ± 0.221	0.867 ± 0.303	0.800 ± 0.283
CMP8	0.067 ± 0.089	12.033 ± 1.387	6.600 ± 1.427	7.300 ± 0.956	9.433 ± 1.182	6.733 ± 0.946
MAJ7	0.000 ± 0.000	0.700 ± 0.322	1.267 ± 0.423	0.133 ± 0.153	0.567 ± 0.272	0.633 ± 0.375
MAJ8	0.000 ± 0.000	0.267 ± 0.205	0.133 ± 0.122	0.000 ± 0.000	0.067 ± 0.128	0.000 ± 0.000
MUX6	1.300 ± 0.264	5.800 ± 0.526	1.700 ± 0.425	5.667 ± 0.642	3.567 ± 0.683	3.567 ± 0.875
MUX11	119.600 ± 10.576	140.700 ± 9.801	78.267 ± 10.561	160.067 ± 14.555	118.467 ± 7.048	116.867 ± 8.162
PAR5	0.000 ± 0.000	2.400 ± 0.503	1.767 ± 0.503	2.033 ± 0.510	3.400 ± 0.567	3.433 ± 0.420
PAR6	0.000 ± 0.000	14.767 ± 1.296	11.267 ± 0.794	13.367 ± 0.792	14.200 ± 0.893	13.767 ± 1.312
PAR7	0.100 ± 0.107	43.133 ± 1.705	36.967 ± 1.690	40.967 ± 1.047	41.033 ± 1.899	41.033 ± 1.814

generations, however its final performance is strongly problem-dependent. In six out of nine symbolic regression benchmarks it converges to suboptimal solutions that are worse than the programs evolved by some of other algorithms. Nevertheless, in the Boolean domain it performs very well, obtaining the best fitness in eight out of nine benchmarks. What is worth noting, the fitness achieved by SGX as of 25 generation on the Boolean problems (except MUX11) remains unbeatable by the other operators until the end of evolution.

The AGX operator quickly converges to solutions of high quality and achieves the best fitness in the symbolic regression domain. On the other hand, it is less efficient in the Boolean domain, where it usually ceases to make progress after 30 generations. The second best operator on the symbolic regression benchmarks (except Nguyen7) is LGX. In the Boolean domain, it usually performs better than AGX, except the relatively simple benchmarks MAJ7 and CMP6. KLX is usually worse than LGX and seems to achieve relatively better results in the Boolean domain than on the symbolic regression problems. The two variants of GPX perform very similarly, however the additional diversity provided by mutation in GPX90 helps in achieving a bit better fitness in 10 out of 18 benchmarks.

To provide a different perspective on the results, we also report the success ratio of particular methods, even though this performance measure depends on the choice of definition of ‘success’ and is known to have certain limitations [26]. Here, a run is considered successful if it finds an individual with fitness less than 10^{-3} , which for the Boolean domain boils down to fitness equal to zero.

Table 4 reports the success ratio together with 95 % confidence interval, as discussed in [42]. AGX has the highest success ratio for all the symbolic regression benchmarks, while SGX has the highest success ratio for all Boolean benchmarks except the MUX problems. Note that in five out of nine Boolean benchmarks, SGX achieves the success ratio of 1.0. The highest ratio for MUX6 is achieved by LGX.

5.2 Generalization

To assess the generalization abilities of operators, we analyze the test-set error, meant as the fitness of an individual calculated with respect to the test set. In Fig. 5, we report the median and the 95 % confidence intervals of the error committed by the best-of-generation individual, i.e., the individual that achieved the best fitness on the training set in a given generation. Table 5 reports the test-set error for the best-of-run individuals. Because aggregation by averaging is very sensitive to outliers, we switch to medians here. Also, as the errors committed by particular operators vary by orders of magnitude, we rely on the log scale in the plots. Figure 5 and Table 5 report the results for symbolic regression benchmarks only, since there is no test-set in the Boolean benchmarks.

It is clear that SGX suffers from major overfitting for all the benchmarks. The test-set error achieved by SGX is at least an order of magnitude worse than the error of the second worst operator for all the benchmarks, except both Keijzer benchmarks, for which the difference is less significant. When it comes to the remaining operators, AGX seems to be the most attractive one in achieving low test-

Table 4 Success ratio and 95 % confidence interval as of 100th generation, assuming fitness less than 10^{-3} as success

Problem	SGX	AGX	LGX	KLX	GPX	GPX90
Keijzer1	0.000 ± 0.000	0.700 ± 0.164	0.067 ± 0.089	0.000 ± 0.000	0.033 ± 0.064	0.033 ± 0.064
Keijzer4	0.000 ± 0.000	0.067 ± 0.089	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Nguyen6	0.000 ± 0.000	0.767 ± 0.151	0.233 ± 0.151	0.433 ± 0.177	0.300 ± 0.164	0.367 ± 0.172
Nguyen7	0.033 ± 0.064	0.900 ± 0.107	0.300 ± 0.164	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Nomic	0.000 ± 0.000	0.033 ± 0.064	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
R1	0.000 ± 0.000	0.100 ± 0.107	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
R2	0.000 ± 0.000	0.167 ± 0.133	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
R3	0.000 ± 0.000	0.600 ± 0.175	0.100 ± 0.107	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
Septic	0.000 ± 0.000	0.200 ± 0.143	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
CMP6	1.000 ± 0.000	0.800 ± 0.143	0.733 ± 0.158	0.533 ± 0.179	0.400 ± 0.175	0.433 ± 0.177
CMP8	0.933 ± 0.089	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.033 ± 0.064	0.000 ± 0.000
MAJ7	1.000 ± 0.000	0.533 ± 0.179	0.300 ± 0.164	0.900 ± 0.107	0.567 ± 0.177	0.600 ± 0.175
MAJ8	1.000 ± 0.000	0.800 ± 0.143	0.867 ± 0.122	1.000 ± 0.000	0.967 ± 0.064	1.000 ± 0.000
MUX6	0.100 ± 0.107	0.000 ± 0.000	0.200 ± 0.143	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
MUX11	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
PAR5	1.000 ± 0.000	0.033 ± 0.064	0.167 ± 0.133	0.133 ± 0.122	0.000 ± 0.000	0.033 ± 0.064
PAR6	1.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000
PAR7	0.900 ± 0.107	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000

Fig. 5 Median and 95 % confidence interval of the test-set error of the best-of-generation individual on the training set

Table 5 Median of the test-set error achieved by the best-of-run individual on the training set as of the 100th generation

Problem	SGX	AGX	LGX	KLX	GPX	GPX90
Keijzer1	0.601	0.015	0.049	0.151	0.152	0.127
Keijzer4	1.515	0.309	0.478	0.585	0.588	0.438
Nguyen6	5.911	0.001	0.002	0.009	0.017	0.012
Nguyen7	5.460	0.001	0.003	0.041	0.020	0.013
Nonic	11.996	0.061	0.081	0.172	0.149	0.168
R1	15.298	0.030	0.008	0.156	0.127	0.143
R2	3.137	0.011	0.038	0.133	0.275	0.164
R3	0.607	0.009	0.008	0.027	0.021	0.029
Septic	14.549	0.032	0.026	0.271	0.257	0.224

set error rates quickly and typically maintaining them throughout the runs. At the ends of runs, AGX may slightly overfit, which causes it to be overtaken by LGX at the end of three out of nine benchmarks. However, the deterioration remains in the range of confidence interval for all the benchmarks.

The results for KLX, GPX and GPX90 are very similar. Among them, KLX achieves its best test-set error a bit quicker than GPX. It is not conclusive whether mutation added to the GPX90 setup helped evolution to generalize better: in six out of nine benchmarks the results of GPX90 are a bit better than the results of bare GPX, however in other three benchmarks GPX is better.

5.3 Statistical significance

In this section, we distill the quantitative results presented so far into qualitative, statistically sound conclusions. Since the aggregate measures used in the assessments on training and test sets are different (average and median), we decided to make use of the non-parametric Friedman's test for multiple achievements of multiple subjects to answer this question [17]. We carried it out for both domains (Boolean, symbolic regression) and each data set (training, testing) separately, and once for all benchmarks together. The outcomes are presented in Table 6 (note that there is no test set in the Boolean domain). The p values resulting from the tests are shown in the headings of particular table insets. Assuming the critical value of $\alpha = 0.05$, all tests are conclusive, i.e., there is at least one significant difference between the fitness/error of best-of-run programs evolved using particular operators. To find the pairs of operators with statistically different performance, we conducted post-hoc analysis using the symmetry test, the results of which are shown in Table 6. Figure 6 shows the outranking graphs built from the significant p values, assuming $\alpha = 0.05$.

AGX fares the best in symbolic regression (Fig. 6a), outranking four other operators on the training sets and three operators on the test sets. The only operator on a par with it is LGX, which however outperforms significantly only two other

Table 6 The outcomes of Friedman's test and post-hoc analysis conducted on the data from Table 3

Symbolic regression – training set Friedman's p-value = 2.54×10^{-5}							Boolean domain – training set Friedman's p-value = 3.70×10^{-4}						
AGX	GPX	GPX90	KLX	LGX	SGX		AGX	GPX	GPX90	KLX	LGX	SGX	
AGX	.013	.020	.000	.807	.000		AGX				.478		
GPX			.807	.657			GPX	.949			.949		
GPX90	1.000		.736	.573			GPX90	.647	.988		1.000		
KLX				1.000			KLX						
LGX	.333	.408	.013	.006			LGX	.081	.478	.863	.949		
SGX							SGX	.000	.012	.081	.150	.647	
Symbolic regression – test set Friedman's p-value = 1.25×10^{-6}							All domains – training set Friedman's p-value = 3.71×10^{-2}						
AGX	GPX	GPX90	KLX	LGX	SGX		AGX	GPX	GPX90	KLX	LGX	SGX	
AGX	.030	.118	.009	.996	.000		AGX				.360		
GPX			.207				GPX	.531			1.000		
GPX90	.996		.951	.062			GPX90	.854	.995		.967		
KLX	.999			.408			KLX						
LGX	.118	.333	.044	.000			LGX	.923	.078	.262	.037		
SGX							SGX	.998	.810	.980	.650	.707	

Every p value in each table inset is the probability of incorrectly judging the respective operator corresponding to a row as outranking the operator corresponding to a column. Significant p values ($\alpha = 0.05$) are marked in bold

Fig. 6 The outranking graphs built from the significant p values of post-hoc analysis of Friedman's test in Table 6. *Solid arcs* denote outranking on both sets, the *dashed* ones on the training set only

operators on both datasets (SGX and KLX). AGX and LGX are also the only non-outranked operators in the symbolic regression domain. The most outranked operators are SGX and KLX, both being outranked by AGX and LGX.

The situation is quite different in the Boolean domain (Fig. 6b), where SGX is the best operator by significantly outranking AGX and GPX. The relative order of the best performing methods is therefore reversed here compared to the symbolic regression domain. This explains why the joint statistics of both domains have only one significant difference between LGX and KLX (Fig. 6c). However, by comparing table insets in Table 6, we can hypothesize that LGX maintains good overall performance across problem domains, while the performance of the other operators substantially varies across the domains.

5.4 Computational costs

From the practical point of view, the actual computational costs behind each operator are of crucial importance. From the range of available measures, we decided to report the CPU time, as it captures not only the costs explicitly visible in an evolutionary workflow (e.g., evaluation of candidate programs), but also the hidden overheads (e.g., searching the library of subprograms in LGX and AGX).

Figure 7 presents the average and 95 % confidence interval of the best-so-far fitness over time, obtained using our experimental platform, which was Core i7-950 processor and 6GB of DDR3 RAM running in the 64-bit mode under the control of Linux and Java 1.7. Note the log timescale. SGX is clearly the fastest operator; if it finds the optimum, it does so in less than two seconds, which is an outstanding result compared to the timing of the other operators. On the other hand, AGX seems to be the slowest operator, with a long convergence time, which is mostly due to searching the library of programs in every crossover act. However, given enough time, AGX finally takes the preeminence over the other operators on the symbolic regression benchmarks. In the Boolean domain, AGX is expectedly worse, since the previous sections demonstrated that it does not cope well with the Boolean problems. LGX operates significantly faster than AGX in the first stage of evolution, and given enough time it achieves similar results on the symbolic regression benchmarks and a lot better results than AGX on the Boolean benchmarks. The methods from the GPX family are indisputably faster than both AGX and LGX, however in the end they often yield to AGX and/or LGX in the symbolic regression domain.

Fig. 7 Average and 95 % confidence interval of best-so-far fitness over time (note the log timescale)

5.5 Bloat

The computational cost of a given GP method strongly depends on the size of programs in a population, which are known to have the unfortunate tendency to grow with evolution time. In Fig. 8 we report the average and 95 % confidence interval of the number of nodes in the best-of-generation individual. We also conducted Friedman's test for multiple achievements of multiple subjects [17] on the number of nodes in the best-of-run individuals to assess the significance of differences between the sizes of programs produced by each operator. Since Friedman's p value is 1.19×10^{-10} , the test is conclusive for $\alpha = 0.05$, and therefore we carried out the post-hoc analysis, reported in Table 7.

The plots in Fig. 8 suggest that the overall dynamics and final characteristic of the program size are largely problem-independent. The most bloating operator is SGX, which is not surprising given the exponential growth captured in Eq. (9). Although our implementation employs a set of basic simplification rules and thus the number of nodes reported here is smaller than predicted by Eq. (9), SGX-evolved programs still grow exponentially with the number of generations. As a result, the programs generated by SGX in 10–20 generations have limited applicability without additional, more thorough simplification.

The other operators cause significantly less bloat than SGX. The second most bloating operator is AGX, however the programs it generates are only insignificantly larger than the trees produced by the third operator, LGX. The least bloating operators are *ex aequo* KLX, GPX, and GPX90, which produce significantly smaller trees than SGX, AGX and LGX.

The general conclusion is that the geometric semantic crossover operators breed significantly larger programs than the non-geometric ones. The only exception is KLX, which, however, ranks very low with respect to fitness and error (cf. Tables 3, 5).

5.6 Degree of geometricity

In the group of considered geometric operators, SGX and AGX proved superior to the other geometric operators, but their relative performances turned out to be substantially different across the symbolic regression and Boolean domains. To explain these phenomena, we examine the geometric properties of the offspring produced by particular crossover operators.

Technically, we monitor each crossover act in the experiment presented in Sect. 5.1 and classify it independently with respect to two aspects:

1. Whether the offspring is:
 - (a) *geometric* (cf. Def. 5) and *effective*,³ i.e., has a geometric semantics that is different from the semantics of both parents,

³ The concept of effectiveness is related to the notion of *neutrality* in GP, meant as modifications of genotype that are not reflected in a phenotype [18] or fitness change [9]. However, these early studies use different representations for genotype (bitstring [18] or integer vector [9]) and phenotype (program syntax [9, 18]). This issue is also addressed in a more recent work on semantic GP [39], however at the level of subtrees, and not of entire programs.

Fig. 8 Average and 95 % confidence interval of the number of nodes in the best-of-generation individual

Table 7 Post-hoc analysis of Friedman's test using the symmetry test conducted on the average number of nodes in the best-of-run individuals

	AGX	GPX	GPX90	KLX	LGX	SGX
<i>Friedman's p value</i> = 1.19×10^{-10}						
AGX						0.596
GPX	0.000		1.000	0.998	0.038	0.000
GPX90	0.000			0.998	0.039	0.000
KLX	0.000				0.009	0.000
LGX	0.537					0.013
SGX

Every p value is the probability of incorrectly judging an operator corresponding to a row as less bloating than an operator corresponding to a column. Significant p values ($\alpha = 0.05$) are in bold and visualized as arcs in the outranking graph

- (b) *geometric* but *ineffective*, i.e., the offspring's semantics is equal to the semantics of one of its parents, or
 - (c) *not geometric* at all;
2. Whether the offspring's fitness is *better* than the fitnesses of parents, *in between* them, or *worse* than both of them.

In total, we consider $3 \times 3 = 9$ categories of crossover outcomes: three due to the former aspect and three due to the latter.

To verify whether an offspring is geometric, we use an L_1 segment connecting the parents' semantics, since in general an offspring is more likely to be produced in such a segment than in an L_2 segment.⁴ Note that an L_1 segment may contain offspring worse than both parents (cf. Sect. 3).⁵ We discard the crossover acts in which any of the parents or offspring includes *NaN* or $\pm\infty$ in its semantics, since in that case the geometric property cannot be determined. In the symbolic regression domain, the elements of semantics are compared with a 10^{-6} tolerance to account for the floating point errors.

Table 8a–e present the frequencies of crossover acts falling into the nine categories defined above, i.e., the count of crossover acts normalized by the total number of them (over 10^7). The 'Better' and the 'Worse' rows of the 'Ineffective' column should in principle contain zeros, since an offspring cannot be strictly better or worse than a parent while having the same semantics (cf. Def. 4). However this may happen, albeit rarely, in the regression domain due to roundoff errors.

The highest fraction of offspring that are simultaneously effective and geometric is produced by SGX in both problem domains, which is not surprising given the exact nature of this operator. In the symbolic regression domain, this fraction is

⁴ The preliminary analysis showed that, except for SGX, the probability of producing a strictly geometric offspring in the L_2 sense is virtually zero.

⁵ This actually happens most of the times, unless an L_1 segment happens to be identical with an L_2 segment.

Table 8 The co-occurrence of the geometric offspring and offspring's fitness (compared to the parents' fitnesses) and Cramer's V [7] for strength of association between these two variables

		Symbolic regression				Boolean domain			
		Geometric offspring				Geometric offspring			
		Not	Ineffective	Effective	Sum	Not	Ineffective	Effective	Sum
<i>(a) SGX</i>									
Fitness	Better	0.000	0.011	0.087	0.098	0.000	0.000	0.064	0.064
	Between	0.000	0.657	0.168	0.825	0.000	0.719	0.152	0.871
	Worse	0.000	0.011	0.066	0.077	0.000	0.000	0.064	0.064
	Sum	0.000	0.679	0.321	1.000	0.000	0.719	0.281	1.000
<i>(b) AGX</i>									
Fitness	Better	0.185	0.001	0.005	0.190	0.002	0.000	0.010	0.012
	Between	0.263	0.129	0.009	0.401	0.006	0.935	0.028	0.969
	Worse	0.405	0.001	0.002	0.408	0.006	0.000	0.014	0.020
	Sum	0.853	0.131	0.016	1.000	0.014	0.935	0.052	1.000
<i>(c) LGX</i>									
Fitness	Better	0.078	0.001	0.003	0.082	0.004	0.000	0.003	0.007
	Between	0.149	0.066	0.006	0.221	0.029	0.724	0.009	0.762
	Worse	0.695	0.001	0.001	0.697	0.225	0.000	0.006	0.231
	Sum	0.923	0.067	0.010	1.000	0.258	0.724	0.018	1.000
<i>(d) KLX</i>									
Fitness	Better	0.019	0.000	0.002	0.021	0.002	0.000	0.009	0.012
	Between	0.033	0.866	0.004	0.903	0.014	0.895	0.034	0.943
	Worse	0.075	0.000	0.000	0.076	0.025	0.000	0.020	0.045
	Sum	0.128	0.866	0.006	1.000	0.042	0.895	0.063	1.000
<i>(e) GPX</i>									
Fitness	Better	0.024	0.000	0.001	0.021	0.002	0.000	0.002	0.004
	Between	0.049	0.102	0.002	0.161	0.024	0.518	0.007	0.549
	Worse	0.821	0.000	0.000	0.818	0.442	0.000	0.005	0.446
	Sum	0.894	0.103	0.004	1.000	0.468	0.518	0.014	1.000
Operator	Symbolic regression				Boolean domain				
	With ‘ineffective’		Without ‘ineffective’		With ‘ineffective’		Without ‘ineffective’		
<i>(f) Cramer’s V</i>									
SGX	0.39		n/a		0.61		n/a		
AGX	0.33		0.10		0.49		0.12		
LGX	0.36		0.16		0.67		0.36		
KLX	0.60		0.23		0.54		0.29		
GPX	0.57		0.19		0.69		0.41		

Better means that the offspring's fitness is strictly better than the fitnesses of *both* parents, *Between* that it is between the parents' fitnesses, *Worse* that it is worse than *both* parents. The χ^2 test for association between variables is conclusive ($p \ll 0.05$) for Table (a)–(e), including and excluding the 'Ineffective' columns

roughly $\frac{1}{3}$, while in the Boolean domain $\frac{1}{4}$. These figures are undoubtedly impressive: still, almost $\frac{2}{3}$ and $\frac{3}{4}$ of offspring, respectively, are geometric but ineffective. As a matter of fact, all operators in the Boolean domain, even the purely syntactic GPX, result in geometric but ineffective offspring in more than 50 % of acts. Nevertheless, this statistics is noticeably higher for the geometric crossover operators than for the semantically unaware GPX. On the other hand, in the symbolic regression domain only SGX and KLX are ineffective more than half of the time. In conclusion, the majority of crossover acts do not advance the search, and this seems to be the common problem for all the operators considered here.

The arguably most interesting question is whether producing a geometric offspring is related to fitness improvement. To answer it, we conducted a separate χ^2 test for every table in Table 8a–e. One should be, however, aware of a technical problem here: as indicated above, a geometric but ineffective offspring, by being semantically equivalent to one of its parents, cannot be better or worse than this parent. The zero or close-to-zero frequencies resulting from that constraint cause an apparent strong relationship between the considered aspects and can bias the results. Therefore, we conducted two analyses: one involving all table columns, and the other one excluding the ‘Ineffective’ column.

The χ^2 test resulted in p values $\ll 0.05$ for all configurations. Thus all tests were conclusive for $\alpha = 0.05$.⁶

In Table 8f we present Cramer’s V [7], the measure of strength of association in the range $[0, 1]$ (0 for no and 1 for perfect association). Cramer’s V can be easily calculated from the χ^2 statistics, but is more informative by factoring in the sample size. With the geometric but ineffective offspring included, the strength of association varies from low to high depending on operator and problem domain. However, with the geometric but ineffective offspring excluded, the strength is very low to medium, and for all operators it reaches substantially higher levels for the Boolean domain. Therefore, the capability of producing geometric offspring has an observable impact on the likelihood of the improvement of offspring’s fitness with respect to its parents.

5.7 Impact of operator characteristics on search performance

The previous section considered the impact of geometric recombination on the change of fitness in the context of a single crossover act. Here, we examine the impact of operators’ characteristics on the actual course of evolutionary search. To that aim, we calculated, independently for both domains, an aggregated ranking of the operators with respect to the average fitness of the best-of-run individuals presented in Table 3. In Table 9, we juxtapose them with the rankings built from the probabilities of producing geometric effective and geometric but ineffective offspring by each operator (calculated from Tables 8a–e). The last but one column of the table presents Spearman’s rank correlation coefficient [36] of the fitness-based rankings and the probability-based ranking. The last column cites the p values of the t test for significance of correlation.

⁶ The test was not applicable to SGX with the ‘Ineffective’ column removed, since in absence of non-geometric offspring only one column remains.

Table 9 Aggregated ranks of the best-of-run fitness (cf. Table 3) and the ranks of probabilities of producing geometric effective Pr(geometric) and geometric ineffective Pr(ineffective) offspring (cf. Table 8a–e), factored by problem domain

	SGX	AGX	LGX	KLX	GPX	Correlation	<i>p</i> value
<i>Symbolic regression</i>							
<i>rank</i> (Fitness)	4.44	1.00	2.11	4.11	3.33		
<i>rank</i> (Pr(geometric))	1.00	2.00	3.00	4.00	5.00	0.10	0.44
<i>rank</i> (Pr(ineffective))	2.00	3.00	5.00	1.00	4.00	−0.56	0.16
<i>Boolean domain</i>							
<i>rank</i> (Fitness)	1.28	4.33	2.56	3.17	3.67		
<i>rank</i> (Pr(geometric))	1.00	3.00	4.00	2.00	5.00	0.57	0.16
<i>rank</i> (Pr(ineffective))	3.00	1.00	4.00	2.00	5.00	−0.26	0.33

The last two columns show Spearman's rank correlation of the fitness-based ranking and the probability-based rankings and the corresponding *p* values of *t* test for significance of correlation

In the symbolic regression domain, we observe medium negative correlation between the best-of-run fitness and the probability of producing geometric but ineffective offspring, and a very low positive correlation between the fitness and producing a geometric effective offspring. For the Boolean domain, the correlation of the fitness and the probability of producing geometric effective offspring is substantially stronger, and the correlation of the fitness and the probability of producing geometric but ineffective offspring is weaker. None of the presented correlation coefficients is statistically significant at $\alpha = 0.05$. We attribute this primarily to the small size of the sample (equal to 5, i.e., the number of operators). Nevertheless, the observed correlations incline us to claim that the use of geometric recombination may have an observable positive impact on the final result of evolutionary search. This impact seems to be low in the continuous semantic space but much stronger in the discrete semantic space. Moreover, its presence shows that a high fraction of ineffective crossover acts is a factor that significantly reduces search performance. Equipping geometric recombination operators with techniques that prevent the ineffective crossovers is thus desirable and worth considering in the further research.

5.8 Diversity analysis

In the previous sections, we hypothesized that some operators may underperform due to the premature convergence of candidate programs. To verify this hypothesis, we carried out a diversity analysis by tracking the average number of semantically unique individuals in every generation of evolutionary runs. Figure 9 presents this statistics, averaged over all 30 runs. The maximum diversity is bound by 1,024, i.e., the population size (cf. Table 1). The runs that terminate are excluded from the analysis in the successive generations—once all runs terminate, the curve of semantic diversity is discontinued (which can be verified by juxtaposing Fig. 9 with Fig. 4). Contrary to the previous experiments, we do not use a tolerance threshold

Fig. 9 Average number of unique individuals in a population over generations

when testing semantics for equivalence in the symbolic regression domain. The rationale for that choice is that tolerance threshold causes the semantic equivalence relation to become intransitive, in which case the calculated number of semantically unique individuals depends on the ordering of individuals in the population (which is random and thus irrelevant).

In Fig. 9, we observe two different patterns in the respective problem domains. In the symbolic regression benchmarks, LGX maintains the highest semantic diversity for the most part of evolutionary runs. AGX provides similar, albeit slightly lower diversity, while the SGX-evolved population is substantially less diverse in the first halves of the runs, but increases towards the end of the runs. KLX usually quickly reduces diversity to 200 or less unique semantics in about 30 generations (except for MAJ8). This characteristics, together with the high fraction of ineffective offspring (cf. Table 8d), explains its poor performance. Both GPX-based setups perform similarly: their populations gradually lose diversity, though with time that degradation typically slows down, and at some point the diversity levels off and remains roughly the same till the end of the run. The mutation featured by GPX90 only slightly improves diversity.

In the Boolean domain, the diversity maintained by most operators is noticeably lower. The exception is SGX, which in the first stage of evolution often manages to make almost every population member semantically unique. Our working explanation is that SGX can in principle⁷ generate *any* semantics in the Hamming segment between the parents. Given two semantically distant parents, such a segment is likely to cover a great part of the entire semantic space (as opposed to the regression problems where the semantic space is infinite). This allows SGX to quickly locate the optimum in most benchmarks (Fig. 4). The exceptions are the MUX problems: here, from the beginning of the runs, SGX gradually decreases the semantic diversity and converges to a suboptimal solution, where it stagnates. It is interesting to observe that no such *gradual* convergence takes place for the other Boolean benchmarks, which SGX manages to solve.

A similar dynamics of diversity can be observed for KLX and LGX, albeit it is less prominent there. After SGX, the second most diverse operator in the first phase of evolution is KLX, however after 30 generations it is overtaken by GPX90 and GPX, which behave almost identically.

The overall conclusion is that all geometric crossover operators are more likely to hamper diversity in the Boolean domain. Another common feature is that, on almost all Boolean benchmarks, semantic diversity seems to increase for a short period of time (typically less than 10 generations), before starting to deteriorate. For the non-semantic operators, this phenomenon is barely observable. This suggests that geometric semantic search in the Boolean spaces spontaneously proceeds in two quite clearly delineated phases of exploration and exploitation.

⁷ In principle, because some semantics in the segment may be not expressible given the available instruction set.

6 Discussion of results

In this section, we review the major observations resulting from the performed experiments, sketch the characteristics of particular operators, and propose explanations for some of the observed phenomena.

SGX obtains the best results in the Boolean domain but performs much worse in the symbolic regression domain, where it quickly stagnates. Our working explanation of this difference is underfitting. More formally, SGX is by definition unable to produce an offspring outside the convex hull of semantics of individuals in the population. As a result, if the convex hull spanning the semantics of programs in the initial population does not contain the target, SGX has no means to reach it. This limitation also applies to the Hamming space of the Boolean domain, but seems to be less crippling there. The convex hull may be apparently more likely to include the target in the finite Hamming space than in an infinite Euclidean space. The practical upshot is that SGX alone may be inherently incapable of composing a good solution from the genetic material available in the initial population only. To do so, it needs the diversifying support from another search operator, e.g., the semantic geometric mutation [29].

Bloat remains a challenge for the exact geometric operators. Sharing parents' code [5] may lessen this problem when *evolving* the programs, but the resulting best-of-run program can be still large and hard to simplify.

AGX attains the best training set performance and test set results for the symbolic regression domain, and is in that respect statistically better than SGX, KLX, and GPX. We hypothesize that this is due to its ability to explore the semantic space beyond the convex hull spanning the semantics of programs in the current population. There are two technical reasons for this. Firstly, AGX relies on a library of subprograms which, even though collected from the current population only, is semantically more diverse than the working population of programs (while SGX relies only on the latter). Secondly, AGX is inherently approximate in its geometric character, so the offspring it produces can be located outside the segment connecting the parents' semantics, and consequently beyond the convex hull.

The performance of AGX in the Boolean domain is much worse and it clearly suffers from the premature convergence there. It is also the second worse when it comes to bloat, and it has very high computational cost. Overall, it may be thus seen as an alternative to SGX for continuous domains with unrestricted semantic space (like symbolic regression), and in applications where good generalization beyond the training set is essential.

LGX fares well on the symbolic regression problems and is quite resistant to overfitting. It is statistically better than SGX and KLX on these problems. In general, it produces smaller programs and is a bit faster than AGX. Similarly to AGX, the main cost of running LGX is searching the library of subprograms, but LGX employs a much simpler technique for matching the subprograms to the desired semantics [25, 33], which may explain the observed difference in computational costs. All in all, LGX can be considered a viable alternative to AGX, especially when it comes to computation time requirements and the size of evolved programs.

The results obtained by KLX are similar to those obtained by the canonical tree-swapping crossover (GPX). Most of the offspring produced by KLX are geometric but ineffective (85–90 %), therefore the diversity in population drops very fast. It seems that KLX has no meaningful advantages in comparison with the other geometric operators.

7 Conclusions

The overall conclusion of this study is that there is no ultimate winner among the considered geometric semantic crossover operators. SGX is undoubtedly attractive in its potential of finding a perfect solution in a short time, but at the price of producing extremely large programs that do not necessarily generalize well. LGX and AGX, on the other hand, yield smaller programs that may still perform reasonably well and have an appealing capability of generalization.

Interestingly, the reason for which SGX does not generalize well may be its core ability of attaining, in the limit, the target semantics. The reasoning behind this hypothesis is as follows. SGX persistently converges to the target,⁸ while neglecting all the other program characteristics (e.g., program size). The other operators, by not being able to fully exploit the geometric properties of the semantic space, may yield programs that commit non-zero errors on some fitness cases. As demonstrated in the above experiments, this characteristic results in performance lower than SGX in the Boolean domain, but higher in the symbolic regression domain.

This consideration clearly relates to the well-known *bias-variance dilemma* known in machine learning [11]. In those terms, SGX has essentially zero bias (in being able to bring the fitting error arbitrarily low), and is thus deemed to pay for that with high variance. The other geometric operators considered here are more biased because they struggle to reach the target, but in exchange for that can enjoy a reasonably low variance.

With this study, we hope not only to share the exciting theoretical aspects of semantic GP, but also to provide some guidance for practitioners. Following the evidence resulting from the experiments and from conceptual analysis, as an overall recommendation concerning crossover operators, we suggest using the Approximately Geometric Semantic Crossover (AGX) [33] in real function synthesis tasks, the exact Semantic Geometric Crossover (SGX) [29] for Boolean function synthesis tasks, and Locally Geometric Semantic Crossover (LGX) [25] as a general purpose operator.

8 Summary and future work

In this study, we have theoretically shown that geometric crossover is able to produce offspring not worse than the best of its parents, given that fitness function is

⁸ A necessary condition is that the target is in the convex hull spanning the semantics of the initial population.

L_1 or L_2 . We also demonstrated that for L_2 such an offspring is provably not worse than the worse parent. We conducted a thorough experimental examination of all known geometric semantic operators. This included examination of the speed of convergence, the likelihood of producing optimal solutions, capability of generalization, bloat characteristic and time complexity. We also showed the positive correlation between the operators' degree of geometricity and performance. Additionally, we scrutinized an important and so far largely neglected factor which highly influences the performance of geometric crossover—the number of ineffective offspring with respect to their parents.

Semantic GP is still in its infancy, so there are all too many possible followups of this study. Because generalization capability is critical in many domains and applications, an investigation of the bias-variance trade-off for the geometric semantic operators is particularly urgent. It seems to be also worthy to thoroughly verify the relationship between the likelihood of breeding of ineffective offspring and the best-of-run fitness; a large scale experiment would be necessary to prove a statistically sound relationship between these features. For similar reasons, more work is needed on designing crossover operators that lower the likelihood of breeding ineffective offspring while still keeping them geometric (both these goals were clearly behind the design of the oldest operator considered here, i.e., KLX). A similar review and analysis as presented here remains to be conducted for the geometric semantic *mutation* operators. Finally, many of these aspects clearly have a direct impact on algorithm runtime, which is yet another future research topic to pursue.

Acknowledgments Work supported by Polish National Science Centre, T. Pawlak is supported by Grant No. DEC-2012/07/N/ST6/03066, B. Wieloch and K. Krawiec are supported by Grant No. DEC-2011/01/B/ST6/07318.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. L. Altenberg, Genome growth and the evolution of the genotype-phenotype map, in *Evolution as a Computational Process* (Springer, Berlin, 1995), pp. 205–259
2. L. Beadle, C. Johnson, Semantically driven crossover in genetic programming, in *IEEE WCCI 2008. IEEE Computational Intelligence Society* (IEEE Press, Hong Kong, 2008), pp. 111–116
3. L. Beadle, C.G. Johnson, Semantic analysis of program initialisation in genetic programming. *Genet. Program Evolvable Mach.* **10**(3), 307–337 (2009)
4. L. Beadle, C.G. Johnson, Semantically driven mutation in genetic programming, in *IEEE CEC 2009. IEEE Computational Intelligence Society* (IEEE Press, Trondheim, 2009), pp. 1336–1342
5. M. Castelli, D. Castaldi, I. Giordani, S. Silva, L. Vanneschi, F. Archetti, D. Maccagnola, An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics, in *EPIA 2013, vol. 8154, LNCS* (Springer, Angra do Heroismo, Azores, Portugal, 2013), pp. 78–89
6. M. Castelli, L. Vanneschi, S. Silva, Semantic search-based genetic programming and the effect of intron deletion. *IEEE Trans. Cybern.* **44**(1), 103–113 (2014)

7. H. Cramér, *Mathematical Methods of Statistics* (Princeton University Press/Almqvist & Wiksells Akademiska Handböcker, Princeton/Uppsala, 1946)
8. N. Dershowitz, J.P. Jouannaud, Rewrite systems, in *Handbook of Theoretical Computer Science*, vol. B (MIT Press, Cambridge, MA, 1990), pp. 243–320
9. C. Ferreira, Genetic representation and genetic neutrality in gene expression programming. *Adv. Complex Syst.* **5**(4), 389–408 (2002)
10. E. Galvan-Lopez, B. Cody-Kenny, L. Trujillo, A. Kattan, Using semantics in the selection mechanism in genetic programming: a simple method for promoting semantic diversity, in *IEEE CEC 2013*, vol. 1 (Cancun, Mexico, 2013), pp. 2972–2979
11. S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma. *Neural Comput.* **4**(1), 1–58 (1992)
12. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st edn. (Addison-Wesley Longman Publishing Co., Inc., Boston, 1989)
13. A. Hara, Y. Ueno, T. Takahama, New crossover operator based on semantic distance between subtrees in genetic programming, in *IEEE SMC 2012* (Seoul, Korea, 2012), pp. 721–726
14. G. Harik, Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. Ph.D. thesis, University of Illinois at Urbana-Champaign (1997)
15. J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (Bradford Books/ MIT Press, Cambridge, MA, 1992)
16. D. Jackson, Phenotypic diversity in initial genetic programming populations, in *EuroGP 2010*, LNCS, vol. 6021 (Springer, Istanbul, 2010), pp. 98–109. doi:[10.1007/978-3-642-12148-7_9](https://doi.org/10.1007/978-3-642-12148-7_9)
17. G. Kanji, *100 Statistical Tests* (SAGE Publications, Thousand Oaks, 1999)
18. R.E. Keller, W. Banzhaf, Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes, in *GP'96* (MIT Press, 1996), pp. 116–122
19. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
20. K. Krawiec, Medial crossovers for genetic programming, in *EuroGP 2012*, LNCS, vol. 7244 (Springer, Malaga, 2012), pp. 61–72
21. K. Krawiec, P. Lichocki, Approximating geometric crossover in semantic space, in *GECCO '09* (ACM, Montreal, 2009), pp. 987–994
22. K. Krawiec, T. Pawlak, Locally geometric semantic crossover, in *GECCO Companion '12* (ACM, Philadelphia, 2012), pp. 1487–1488
23. K. Krawiec, T. Pawlak, Quantitative analysis of locally geometric semantic crossover, in *PPSN XII*, vol. 7491, LNCS (Springer, Berlin, 2012), pp. 397–406
24. K. Krawiec, T. Pawlak, Approximating geometric crossover by semantic backpropagation, in *GECCO '13* (ACM, Amsterdam, 2013), pp. 941–948
25. K. Krawiec, T. Pawlak, 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)
26. S. Luke, L. Panait, Is the perfect the enemy of the good?, in *GECCO 2002* (Morgan Kaufmann Publishers, New York, 2002), pp. 820–828
27. J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, U.M. O'Reilly, Genetic programming needs better benchmarks, in *GECCO '12* (ACM, Philadelphia, 2012), pp. 791–798
28. P. Merz, B. Freisleben, Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *Trans. Evol. Comp.* **4**(4), 337–352 (2000)
29. A. Moraglio, K. Krawiec, C.G. Johnson, Geometric semantic genetic programming, in *PPSN XII*, vol. 7491, LNCS (Springer, Taormina 2012), pp. 21–31
30. A. Moraglio, D. Sudholt, Runtime analysis of convex evolutionary search, in *GECCO '12* (ACM, New York, 2012), pp. 649–656
31. Q.U. Nguyen, X.H. Nguyen, M. O'Neill, Semantics based mutation in genetic programming: the case for real-valued symbolic regression, in *Mendel'09* (Brno, Czech Republic, 2009), pp. 73–91
32. T.P. Pawlak, Combining semantically-effective and geometric crossover operators for genetic programming, in *PPSN XIII* (Ljubljana, Slovenia, 2014)
33. T.P. Pawlak, B. Wieloch, K. Krawiec, Semantic backpropagation for designing search operators in genetic programming. *IEEE Trans. Evolut. Comput.* (2014). doi:[10.1109/TEVC.2014.2321259](https://doi.org/10.1109/TEVC.2014.2321259)
34. R. Poli, W.B. Langdon, Schema theory for genetic programming with one-point crossover and point mutation. *Evol. Comput.* **6**(3), 231–252 (1998)

35. F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, 2nd edn. (Springer, New York, 2006)
36. C. Spearman, The proof and measurement of association between two things. *Am. J. Psychol.* **15**, 88–103 (1904)
37. W.A. Tackett, A. Carmi, The unique implications of brood selection for genetic programming, in *IEEE WCCI 1994*, vol. 1 (IEEE Press, Orlando, 1994), pp. 160–165
38. N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, E. Galvan-Lopez, Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program Evolvable Mach.* **12**(2), 91–119 (2011)
39. N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, D.N. Phong, On the roles of semantic locality of crossover in genetic programming. *Inf. Sci.* **235**, 195–213 (2013)
40. L. Vanneschi, M. Castelli, S. Silva, A survey of semantic methods in genetic programming. *Genet. Program Evolvable Mach.* **15**(2), 195–214 (2014)
41. J.A. Walker, J.F. Miller, Investigating the performance of module acquisition in cartesian genetic programming, in *GECCO 2005*, vol. 2 (ACM Press, Washington, DC, 2005), pp. 1649–1656
42. S. Wallis, Binomial confidence intervals and contingency tests: mathematical fundamentals and the evaluation of alternative methods. *J. Quant. Ling.* **20**(3), 178–208 (2013)
43. R. Watson, Compositional evolution: Interdisciplinary investigations in evolvability, modularity, and symbiosis. Ph.D. thesis, Brandeis University (2002)
44. B. Wieloch, K. Krawiec, Running programs backwards: instruction inversion for effective search in semantic spaces, in *GECCO '13* (ACM, Amsterdam, 2013), pp. 1013–1020
45. S. Wright, The roles of mutation, inbreeding, crossbreeding and selection in evolution, in *Proceedings of the 6th International Congress of Genetics*, vol. 1 (1932), pp. 356–366