Competent Geometric Semantic Genetic Programming for Symbolic Regression and Boolean Function Synthesis

Tomasz P. Pawlak tpawlak@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Krzysztof Krawiec krawiec@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Abstract

Program semantics is a promising recent research thread in Genetic Programming (GP). Over a dozen of semantic-aware search, selection, and initialization operators for GP have been proposed to date. Some of those operators are designed to exploit the geometric properties of semantic space, while some others focus on making offspring *effective*, i.e., semantically different from their parents. Only a small fraction of previous works aimed at addressing both these features simultaneously. In this paper, we propose a suite of *competent operators* that combine effectiveness with geometry for population initialization, mate selection, mutation and crossover. We present a theoretical rationale behind these operators and compare them experimentally to operators known from literature on symbolic regression and Boolean function synthesis benchmarks. We analyze each operator in isolation as well as verify how they fare together in an evolutionary run, concluding that the competent operators are superior on a wide range of performance indicators, including best-of-run fitness, test-set fitness, and program size.

Keywords

semantics, metrics, geometry, effectiveness, theory, experiment

1 Introduction

Recent research in Semantic Genetic Programming (SGP) (Vanneschi et al., 2014) pushed the envelope of EC-based program synthesis by shifting the focus from program syntax to program behavior. An important thread in this area is Geometric Semantic GP (GSGP) (Moraglio et al., 2012), where program semantics is interpreted as a point representing its output on the training set in a multidimensional space, dubbed *semantic space*.

Research in GSGP focuses on designing search operators that have well-defined impact on the geometric relationships between programs in the semantic space (Section 2). However, practice shows that they often produce offspring solutions that, though formally geometric, fail to be useful. For instance, an offspring program that is semantically equivalent to its parent does not advance search and is a waste of computing resources. The empirical evidence we brought in previous study (Pawlak et al., 2015a) shows that this happens frequently. In other words, conventional GSGP *constrains* the distribution of offspring semantics, but is in principle not concerned with *shaping* the resulting distribution. In this paper, we scrutinize that distribution with respect to *effectiveness*, meant as the prospective likelihood of search progress. To control and improve effectiveness, we propose a suite of algorithmic components that, apart from

providing for geometric properties (in an exact or approximate sense), are also *effective*, i.e., produce candidate programs that are likely to advance search. We achieve this goal with two means: by using improved variants of effective search operators (mutation and crossover) proposed in our earlier works (Pawlak et al., 2015b; Krawiec and Pawlak, 2013a; Wieloch and Krawiec, 2013), and by augmenting the conventional GSGP workflow with semantic-aware and geometric components: effective geometric initialization and effective geometric mate selection (Sect. 4, Pawlak (2015)). These contributions together form a coherent approach dubbed *competent* GSGP.

We start with a brief formalization in Sect. 2, and in Sect. 3 review the related work. In the experimental part, we analyze each of the proposed components in isolation (most of Sect. 5), and then assess the joint performance of the competent suite (Sect. 5.6) and discuss the results and prospects (Sects. 6 and 7).

2 Semantics in GP

For the sake of this study, we assume that programs are functions, i.e., feature no side effects, nor persistent state between applications to input data. Let $\mathcal{I}$ denote a set of all program inputs, and $\mathcal{O}$ be a set of all program outputs. A program p has thus the signature $p : \mathcal{I} \to O$, and we write $o = p(in)$ to state that p applied to input $in \in \mathcal{I}$ returns an output $o \in \mathcal{O}$. The set of all programs is denoted by $\mathcal{P}$.

Given an n-tuple $I \in \mathcal{I}^n$ of inputs, $I = [in_1, \ldots, in_n]$, the *semantics* $s = [o_1, o_2, ..., o_n]$ is a tuple of corresponding outputs $o_i \in \mathcal{O}$, and $\mathcal{S} \subseteq \mathcal{O}^n$ be a set of all semantics. Note that semantics is *any* combination of elements of $\mathcal{O}$, including the combinations that cannot be generated by any program in $\mathcal{P}$. *The semantics of a program* p is the tuple of outputs it produces for the elements of I, i.e., $s(p) = [p(in_1), \ldots, p(in_n)]$. By definition $s(p) \in \mathcal{S}$. Though $s(p)$ depends on the inputs in I, we write $s(p)$ for brevity.

A *program induction problem* is defined by a program space $\mathcal{P}$, a tuple of n inputs $I \in \mathcal{I}^n$, a *target* semantics $t \in \mathcal{S}$, and a minimized fitness function $f : \mathcal{S} \to \mathbb{R}_{\geq 0}$. A pair of corresponding elements from I and t, i.e., (in_i, t_i), $in_i \in I$, $i = 1, \ldots, n$, forms a *fitness case*. We assume that $f(s) = d(s, t)$ for target t and some metric d. A program p^* such that $s(p^*) = t$ is an *optimal solution* to such program induction problem, and minimizes f, i.e., $f(s(p^*)) = 0$.

In the following, we define properties for each operator involved in GP search: population initialization, mate selection and search operators.

2.1 Geometric search operators and convex hulls

Moraglio et al. (2012) formalized geometric mutation and crossover operators for semantic GP. We adapt those definitions to the notation used in this paper.

Definition 1. A mutation operator is a (random) function $\mathcal{P} \to \mathcal{P}$. A mutation operator is *r-geometric* iff the semantics $s(p')$ of the produced offspring p' belongs to a ball of radius r centered in parent's semantics p, i.e., $\|s(p'), s(p)\| \leq r$.

Moraglio (2011) showed that the convex hull of offspring population is a (proper or not) subset of the convex hull of the parent population. Starting from that observation, we extend the conventional two-parent geometric crossover to an arbitrary number of parents. Let $C(X)$ denote the convex hull of X, i.e., the smallest convex set that includes X. For brevity, we abuse the notation by writing $C(P)$ to denote the convex hull in $\mathcal{S}$ spanning the semantics of programs from P and refer to it as a convex hull of *programs*.

Definition 2. A k-ary crossover ($k \geq 2$) is a (random) function $\mathcal{P}^k \to \mathcal{P}$. A k-ary crossover is *geometric* iff the semantics $s(p)$ of the offspring p belongs to the convex hull

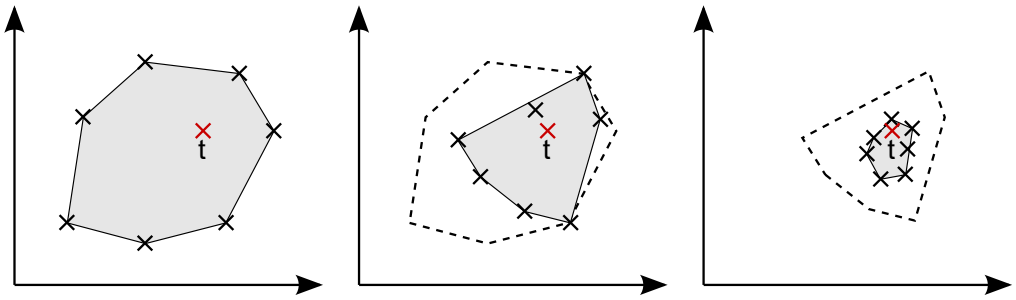


Figure 1: The effects of geometric crossover in $n = 2$-dimensional semantic space $\mathbb{R}^2$ under the L_2 metric. Left: initial population, center: second generation, right: third generation. Dashed polygons are convex hulls of previous generations.

of semantics of its parents $p_i, i = 1..k$, i.e., $s(p) \in C(\{p_1, p_2, ..., p_k\})$.

For $k = 2$, a convex hull is equivalent to a segment, and Def. 2 becomes identical to that presented by Moraglio et al. (2012). Convex hulls embrace different sets of semantics for different metrics. For more details and rationale see (Pawlak and Krawiec, 2016a; Pawlak, 2015).

Following the proof by Moraglio (2011), the relevance of convex hulls for geometric crossover is illustrated for $n = 2$ fitness cases in Fig. 1. The black points mark the semantics of seven programs. An application of a k-ary geometric crossover to any k programs from that set (left inset) can produce offspring in the convex hull of that set only (the shaded area). If geometric crossover is the only search operator employed, all resulting offspring lie in this convex hull (central inset). With each generation, the convex hull shrinks (or at most remains unchanged). In the limit, the convex hull collapses to a single point (semantics).

What follows from this example is that the necessary condition for a geometric crossover to produce an offspring with semantics t is that t must be included in the convex hull of the population. We formalize this observation as follows.

Theorem 3. *A geometric $k \geq 2$-ary crossover applied to a set of programs P may produce a program p such that $s(p) = t$ iff $t \in C(P)$.*

Proof. The proof follows from Def. 2, i.e., a geometric k-ary crossover must produce only offspring in the convex hull of the parents, hence if t lies outside this convex hull, the operator is unable to reach t. □

As already noted by Moraglio et al. (2012), geometric semantic mutation does not require a similar assumption, i.e., it ultimately converges to the target regardless whether $t \in C(P)$ holds or not. However, that convergence may require many iterations, which depends, among others, on the radius of the ball in Def. 1. Geometric crossover can make long 'leaps' in $\mathcal{S}$, reducing the number of required iterations.

2.2 Target-geometric initialization and mate selection

Whether the target remains within the convex hull of population depends on (i) the initial population and (ii) the choice of the parents to mate when producing consecutive populations. We define the desirable properties of these objects below. The first property originated in our previous work (Pawlak and Krawiec, 2016b).

Definition 4. An initialization operator is a (random) function that produces a multiset P of elements from $\mathcal{P}$. An initialization operator is *t-geometric* iff the convex hull of programs in P includes t, i.e., $t \in C(P)$.

Definition 5. A k-mate selection operator is a (random) function $\mathcal{P} \rightarrow \mathcal{P}^k$. A k-mate selection operator is *t-geometric* iff the convex hull of the selected programs $p_i, i = 1..k$, includes t, i.e., $t \in C(\{p_1, p_2, ..., p_k\})$.

The practical upshot of the above is that GP equipped with a target-geometric initialization, a target-geometric mate selection, and geometric search operators is likely to maintain its target t in population's convex hull in consecutive iterations[1] (assuming it was present in the initial population). The usefulness of these components depends on their effectiveness, which we cover in the next subsection, and on the technical possibility of constructing geometric and effective operators for a given semantic space and metric, which we investigate in Sect. 4.

2.3 Effective operators

We consider an operator *effective* if it avoids producing programs that are semantically equal to already considered programs. The meaning of the phrase 'already considered programs' depends on the context, so we define effectiveness separately for initialization, selection, and search operators.

Definition 6. An initialization operator is *effective* iff each initial population P it produces does not contain semantically equal programs, i.e., $\forall_{i \neq j} s(p_i) \neq s(p_j), p_i, p_j \in P$.

Definition 7. A k-mate selection operator $sel : 2^{\mathcal{P}} \rightarrow \mathcal{P}^k$ is *effective* iff, for any $P \subseteq \mathcal{P}$, no two programs in $sel(P)$ have the same semantics, i.e., $\forall_{i \neq j} s(p_i) \neq s(p_j), p_i, p_j \in sel(P)$.

Definition 8. A k-ary search operator $h : \mathcal{P}^k \rightarrow \mathcal{P}$ is *effective* iff, for any $P \in \mathcal{P}^k$, each offspring produced by h from P is semantically distinct from all its parents, i.e., $\forall_{p' \in h(P), p \in P} s(p') \neq s(p)$.

Effective operators lower the risk of revisiting the same solutions and are beneficial for diversification of population. We expect them to increase the odds of solving program induction problems, and verify this hypothesis in Sect. 5.

3 Previous works

This paper involves initialization, mate selection, and search operators, so we structure the review around these categories.

3.1 Semantic-aware search operators

Design of **geometric** search operators is arguably the main research area within GSGP. The key reference methods are here Semantic Geometric Crossover (SGX) and Mutation (SGM) proposed by Moraglio et al. (2012). SGX works by linearly combining parent programs. SGM builds an offspring from a parent program and a carefully designed random expression so that in effect the semantics of the offspring is only slightly disturbed (up to a controlled degree) with respect to parent's semantics (see Pawlak (2016) for proof).

SGM was thoroughly studied by Moraglio et al. (2013); Moraglio and Mambrini (2013), in the Boolean and symbolic regression domains, respectively. Pawlak and Krawiec (2016a) analyzed bounds on fitness change in a single application of SGX and SGM

[1] This likelihood would be turn into certainty if the search operator *guaranteed* that the resulting offspring (i.e., the next population) forms a convex hull that includes t. However, each application of a typical search operator is independent and returns only one offspring, which precludes ensuring this property.

under different metrics and fitness functions. Moraglio et al. (2014) adapted SGX to grammatical evolution. SGX and SGM are geometric, but not effective in the sense of Def. 8.

To our knowledge, Krawiec and Lichocki (2009) proposed the first *approximately* geometric crossover for GP. This operator involves brood selection: a set of offspring candidates is produced, and the one that minimizes a custom measure of divergence from geometry is returned. This operator is characterized by high probability of producing offspring semantically equal to one of their parents (see Pawlak et al. (2015a)). This probability was reduced to zero by Pawlak (2014), making so, this operator effective w.r.t. of Def. 8.

Krawiec and Pawlak (2013b) proposed Locally Geometric Crossover (LGX) that approximates geometric crossover by replacing parent subprograms using library of known programs. LGX does not guarantee that an offspring is semantically different from its parents, thus is not effective.

Pawlak et al. (2015b); Krawiec and Pawlak (2013a); Wieloch and Krawiec (2013) developed approximately geometric, but not effective, mutation (RDO) and crossover (AGX) that are based on the idea of inverse execution of parent program(s). The competent mutation and crossover proposed in Sects. 4.3 and 4.4 are respective variants of RDO and AGX extended for effectiveness.

Nguyen et al. (2016) proposed Subtree Semantic Geometric Crossover (SSGX) that, given parent programs, picks from each of them one subprogram that is semantically most similar to the parent program and linearly combines them in the same manner as SGX (Moraglio et al., 2012). SSGX produces programs smaller by few orders of magnitude than SGX, while maintaining similar fitness-based characteristics. SSGX is approximately geometric, but not effective.

Concerning semantic-aware operators that aim at being **effective**, Beadle and Johnson (2008, 2009b) proposed, respectively, Semantically Driven Crossover (SDX) and Mutation (SDM) for the Boolean and artificial ant domains. SDX and SDM repeat in a loop, respectively, the conventional Tree Crossover (TX) and Tree Mutation (TM) (Koza, 1992), until the produced offspring candidate is semantically different from its parents. SDX and SDM are thus effective w.r.t. Def. 8.

Jackson (2010b) also experimented with an effective crossover based on TX (Koza, 1992), which admits an offspring only if its is semantically different from both its parents.

The series of publications by the same authors (Nguyen et al., 2009; Uy et al., 2011, 2013) proposes three crossover and one mutation operator. The common feature of all these operators is that they breed offspring by replacing subprograms in parents with semantically different subprograms. These operators are not effective in sense of Def. 8, because other parts of offspring program may still cause it to be semantically equal to one of its parents.

3.2 Semantic-aware population initialization

Most works on initialization in SGP focus on providing semantic diversity in the initial population. In one of the earliest attempts of that type, Looks (2007) proposed Semantic Sampling (SS) – a population initialization heuristic that defines bins for programs of particular sizes and fills them up to assumed capacity by semantically distinct programs.

Semantically Driven Initialization (SDI) presented by Beadle and Johnson (2009a) for the Boolean and artificial ant domains is effective in sense of Def. 6. SDI starts with seeding a population with single node-programs. Then, it iteratively picks a random instruction and combines it with programs drawn from the population. The resulting program is added to the population if no other program in there has equal semantics.

Jackson (2010a) proposed Behavioral Initialization (BI) that acts as a wrapper on

the Ramped Half and Half method (RHH, Koza (1992)). BI invokes RHH in a loop and admits an initialized program to the population if it is semantically different from all programs already present in the population. Thus, BI is effective, but not geometric.

The recent Semantic Geometric Initialization (SGI) (Pawlak and Krawiec, 2016c) is geometric and effective in the sense of Defs. 4 and 6. SGI surrounds a target by a set of semantics that form a convex hull that includes the target. Then, it applies domain-specific exact algorithms that explicitly construct programs for the abovementioned semantics. SGI turns out to significantly improve the performance of GSGP.

3.3 Semantic-aware selection operators

The works on semantic-aware selection, including mate selection, are few and far between. Galván-López et al. (2013) proposed Semantic Tournament Selection (STS), a 2-mate selection operator. STS selects the first parent using Tournament Selection (TS, Miller and Goldberg (1995)). The second parent is the best one from a set of candidates drawn from the population, where the candidates that are semantically equal to the first parent are discarded. If all candidates in this set are worse than the first parent, STS by definition allows the same program to be selected twice and act as both parents. Hence, STS is not effective w.r.t. Def. 7.

For a comprehensive survey of semantic methods in GP, the reader is referred to Vanneschi et al. (2014); Pawlak et al. (2015a).

From all the above-mentioned initialization operators, only the one by Pawlak and Krawiec (2016b) is t-geometric. However, programs initialized by that operator by design memorize the correct outputs for particular inputs (by storing them explicitly in the program), and as such are unable to generalize well for unseen inputs. In contrast, the competent initialization (CI) proposed in this paper searches for semantically unique programs that expand population's convex hull while not explicitly storing correct outputs in program code. As it will become clear in the following section, the competent tournament selection (CTS) is significantly different from past work too: it takes into account the relative locations of the target and mates' semantics, which none of the former semantic-aware selection methods did. The semantic-aware mutation and crossover operators that complement our competent framework are effective counterparts of their ancestors presented in (Pawlak et al., 2015b; Krawiec and Pawlak, 2013a; Wieloch and Krawiec, 2013).

4 Proposed Competent Operators

The definitions presented in the previous section determine only the desirable properties of GSGP operators, but do not propose concrete algorithms. In this section we present a suite of *competent operators* for GP, i.e., algorithms that are effective (Defs. 6–8) and *designed to approximate* the geometric characteristics (Defs. 1–5). The latter characteristics are achieved only approximately or in the limit by some of the proposed components, for the reasons that we detail in the following. This however does not prevent them from performing significantly better than the original GSGP, as we demonstrate in Sect. 5.

All GSGP concepts depend on the choice of a metric, and so do the algorithms proposed here. In the following, we assume the Minkowski metric of order z:

$$L_z(a,b) = \left(\sum_i |a_i - b_i|^z \right)^{\frac{1}{z}}. \tag{1}$$

L_1 is the city-block metric, and L_2 is the Euclidean metric. Note that the Minkowski met-

Algorithm 1 Competent Initialization algorithm. $\Phi_\bullet$ is a set of terminals, $\Phi_\circ$ is a set of non-terminals, RANDOM(X) picks random element from set X, ARITY(p') returns arity of node p'.

```
1: function CI(Φ•, Φ∘)
2:     P ← Φ•                                   ▷ Add terminals to population
3:     while |P| < popsize do
4:         p' ← RANDOM(Φ∘)
5:         for i = 1..ARITY(p') do
6:             p'_i ← RANDOM(P)                 ▷ Assign child node
7:         if ∀p ∈ P : s(p) ≠ s(p') ∧ s(p') ∉ C(P) then
8:             P ← P ∪ {p'}
9:     return P
```

ric can be applied to any domain with numeric elements, including symbolic regression and the Boolean domain with $0/1$ encoding.

4.1 Competent Initialization (CI)

Algorithm 1 presents *Competent Initialization* (CI) for tree-based programs. Like the conventional GP search operators, CI constructs new programs from the programs already present in the population.

The algorithm begins with filling the working initial population P with single-node programs made of individual terminal (zero-argument) instructions. Then, it iteratively builds new program candidates bottom-up from the programs already present in P, and accepts a candidate p only if it meets two conditions (line 7): P contains no program with the same semantics as p, and p expands the convex hull of the programs in P. Verifying the latter condition is equivalent to checking whether $s(p) \in C(P)$, which for the L_1-convex hull is true iff:

$$\forall_i \min_{p \in P} s(p)_i \leq s(p')_i \leq \max_{p \in P} s(p)_i \tag{2}$$

To verify the condition for the L_2-convex hull, we arrange the semantics of the programs in P into a matrix, where element $s_{i,j}$ is the jth component of the semantics of the ith program in P. Then, $s(p) \in C(P)$ iff the system of equations

$$\begin{bmatrix} s_{1,1} & \cdots & s_{|P|,1} \\ \vdots & \ddots & \vdots \\ s_{1,n} & \cdots & s_{|P|,n} \\ 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{|P|} \end{bmatrix} = \begin{bmatrix} s(p)_1 \\ \vdots \\ s(p)_n \\ 1 \end{bmatrix} \tag{3}$$

has solution(s) $\forall x_i \in [0, 1]$. The existence of such solutions can be verified by solving this system using, e.g., Singular Value Decomposition (Burden and Faires, 2010), which we do in the experimental part.

An implementation of Algorithm 1 requires additional safeguards that warrant termination of the loop in lines 3–8, because CI may be unable to produce a program that expands the convex hull of the population, for instance when the available instructions are not sufficient to express certain semantics, or the convex hull already incorporates the entire semantic space, i.e., $C(P) = \mathcal{S}$. Thus, if CI cannot fulfill the condition in line 7 in an assumed number of attempts, it supplements the population with a random program and starts the loop over. Moreover, CI discards the candidate programs that

Algorithm 2 Competent Tournament Selection algorithm. p is mate parent, μ is tournament size, and RANDOM(P, μ) draws without replacement μ elements from P.

1: **function** CTS(P)
2: $p_1 = \text{TOURNAMENTSELECTION}(P)$
3: $\Gamma \leftarrow \text{RANDOM}(P \backslash \{p_1\}, \mu)$
4: $p_2 \leftarrow \arg\min_{p' \in \Gamma} \underbrace{\frac{\overbrace{L_z(s(p'), t)}^{\text{distance to target}}}{L_z(s(p'), s(p_1))}}_{\text{distance to mate program}} \times \underbrace{(1 + |L_z(s(p'), t) - L_z(s(p_1), t)|)}_{\text{penalty for unequal distances of programs to target}}$
5: **return** $\{p_1, p_2\}$

return the same value for all fitness cases, because in most problems such programs are of little use.

CI is effective in the sense of Def. 6, because it explicitly forbids adding to the population P a program that is semantically equal to any program already in P. CI does not require the knowledge of the target (note the absence of t among CI's arguments in Algorithm 1), and can be thus applied to problems with unknown t . By the same token CI is not target-geometric (Def. 4): the resulting convex hull $C(P)$ is in general not guaranteed to include t. Achieving such a guarantee would require a domain-specific algorithm, which we did not want to involve in this study. Also, even a target-geometric initialization operator would fail to produce a convex hull that encloses t if population size was small compared to the dimensionality of semantic space n; for instance, it may be hard to find two programs that form a convex hull enclosing t even in two-dimensional semantic space. Nevertheless, we show in Sect. 5.2 that in practice CI is likely to produce an initial population that includes t.

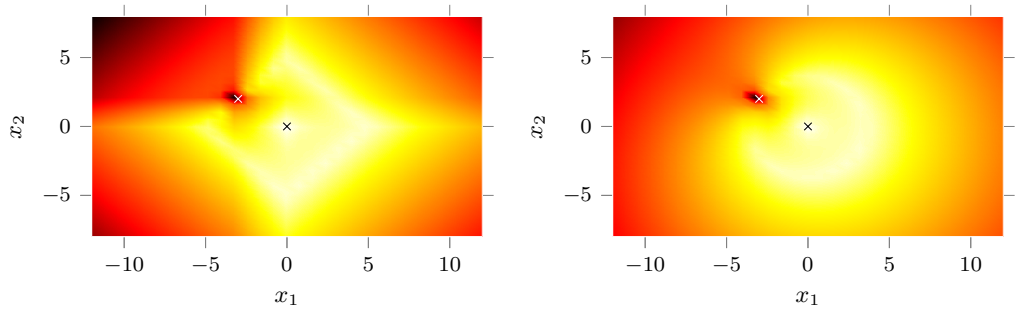
4.2 Competent Tournament Selection (CTS)

The intent behind the t-geometric k-mate selection (Def. 5) is to select from a population k parent programs that form a convex hull that includes the target t. This makes it possible for the subsequent k-ary geometric crossover to generate an offspring in the proximity of t. Such k programs are guaranteed to exist in P if $t \in C(P)$ and $k \geq n + 1$, where n is the dimensionality of the semantic space (Carathéodory, 1911). It would be tempting thus to consider such k-ary selection operators, *if only* $t \in C(P)$. However, our competent framework, though strongly motivated by geometric relationships between programs, does not guarantee that the target is within the convex hull of the population throughout the run. For instance already CI, which supplies the population with initial programs, does not provide such a guarantee. In this context, there is limited motivation for designing k-ary, $k > 2$ selection operators, and the *Competent Tournament Selection* (CTS) that we propose below is designed to approximate the t-geometric 2-mate selection.

CTS (Algorithm 2) selects the first parent p_1 using the tournament selection (Miller and Goldberg, 1995). Then, it draws uniformly without replacement μ candidate programs and selects the one that minimizes the expression in line 5, which is a heuristic measure of desirability of a candidate mate p' for p_1 with respect to three aspects:

1. The distance between $s(p')$ and the target t – to be minimized in order to to prefer better programs,

2. The distance between $s(p')$ and $s(p_1)$ – to be maximized to penalize the candidates



(a) L_1 (b) L_2

Figure 2: Spatial characteristics of selection expression in CTS under L_1 and L_2 in $\mathbb{R}^2$, for $t = (0, 0)$ (black $\times$) and $s(p_1) = (-3, 2)$ (white $\times$). White color marks the minimum of the expression, dark red marks the maxima (in the considered range), black is ∞.

that are semantically too similar to p_1 (and so prefer distinct programs) and to make effective selection more likely (cf. Def. 7),

3. The difference between the distances of $s(p')$ and $s(p_1)$ from the target t – to be minimized to promote the close-to-equidistant candidates p'.

Terms (2) and (3) make it likely for the center of the segment connecting $s(p_1)$ and $s(p')$ to be close to t. Figure 2 presents the visualizations of the expression for L_1 and L_2 metrics in the $\mathbb{R}^2$ semantic space ($n = 2$). The minimal value of 0 is attained for the candidates having target semantics, i.e., $s(p') = t$, as mandated by the positive bias toward the good candidates (term 1). For the points close to $s(p_1)$, the expression approaches infinity, so such candidates are discouraged (term 2). Crucially, for the candidates located 'on the opposite side' of t and at the same distance to t as p_1, i.e., $L_z(s(p_1), s(p')) = 2L_z(s(p_1), t)$, the expression attains the relatively low value of $1/2$ (term 3).

CTS is effective (Def. 7) because the drawing procedure in line 2 prevents repeated selection of the same candidate (provided P contains at most μ semantic duplicates). CTS is not t-geometric, since it does not guarantee selecting $\{p_1, p_2\}$ such that t belongs to the convex hull $C(\{p_1, p_2\})$. Note that for $k = 2$, that convex hull is a segment connecting $s(p_1)$ and $s(p_2)$, so picking a pair of programs with the above property from a finite population P is particularly unlikely. Nevertheless, as we demonstrate later in Sect. 5.3, simultaneous optimization of the three above terms allows CTS to select the parents that are likely to result in geometric and well-performing offspring.

Contrary to what Algorithm 2 may suggest, CTS does not require access to the target t. Note that the terms that t appears in (line 5) are actually *fitness values*: $L_z(s(p'), t)$ is nothing else than the fitness of p', while $L_z(s(p_1), t)$ is the fitness of p_1 (assuming fitness uses on L_z). Therefore, even though CTS attempts to keep t within the convex hull of the parents (or otherwise expand it towards t), the explicit knowledge of its location is not essential, as the necessary information is conveyed by the fitness value. Thus, similarly to CI, CTS can be applied to problems with an unknown t.

4.3 Competent Mutation (CM)

Competent Mutation (CM) relies on the concept of *semantic backpropagation* (SB) we introduced in Random Desired Operator (RDO) (Pawlak et al., 2015b; Wieloch and Krawiec,

Algorithm 3 Competent Mutation algorithm. t is target, p is parent. LIBSEARCH$(D, D_\varnothing)$ returns a program that matches D and does not match $D_\varnothing$, RANDOMNODE(p) picks random node from program p, REPLACE(p, a, p') replaces a in p with p', SB comes from Algorithm 5 in Appendix A.

```
1: function CM(t, p)
2:     a ← RANDOMNODE(p)
3:     D ← SB(t, p, a)
4:     D_∅ ← SB(s(p), p, a)
5:     p' ← LIBSEARCH(D, D_∅)
6:     return REPLACE(p, a, p')
```

2013; Wieloch, 2013). The SB algorithm, given a semantics s, a program p, and a subprogram a in p (i.e., its location in p), computes the *combinatorial semantics*. Combinatorial semantics is a tuple of sets D_i, each corresponding to a single fitness case. D_i stores the values such that, when substituted for a in p, cause the resulting program to return the ith element of the target (t_i) when evaluated on ith fitness case. Because program execution on each fitness case is independent, any combination of values from D_is causes the program to have semantics equal to s. In this way, combinatorial semantics captures – in general exponentially many – semantics of subprograms that, when substituted for a, would make the resulting program attain semantics s. Because SB involves domain-specific inversion of instructions, we detail it in Appendix A to maintain smooth narration here.

Given a parent p and a target t, CM (Algorithm 3) starts with drawing a node a from p. Then, it calls SB(t, p, a) to obtain the *desired semantics* D, i.e., the combinatorial semantics of subprograms that would change p's semantics to t when substituted for a. Next, CM calls SB$(s(p), p, a)$, i.e., backpropagates through the parent program p *its own semantics* $s(p)$. The resulting combinatorial semantics $D_\varnothing$ represents the *forbidden semantics* – the subprograms that, when substituted for a, do not change the semantics of p. Then, a library of subprograms is searched for a subprogram p' that matches D as close as possible and does not match $D_\varnothing$. An efficient algorithm for performing that search is presented in Appendix B. Finally, CM replaces a in p with p' to produce an offspring.

By aiming at desired semantics while simultaneously avoiding forbidden semantics, CM attempts to modify the parent so that the offspring's semantics becomes equal to t. When that succeeds, CM solves a program induction problem in a single step; this is however not frequent in practice, due to possible absence of an adequate subprogram in the library.

Because the distance of the parent from the offspring can be arbitrary, CM is not geometric in the sense of Def. 1.

Because library search must not return a subprogram that matches $D_\varnothing$, the offspring is guaranteed to be semantically different from the parent. Thus, CM is effective w.r.t. Def. 8.

Note that, in contrast to all other operators in our competent suite, CM requires the knowledge on the location of the target t.

4.4 Competent Crossover (CX)

Competent Crossover (CX) operates analogously to Approximately Geometric Semantic Crossover (AGX) (Pawlak et al., 2015b; Krawiec and Pawlak, 2013a) and is augmented with extensions that make it effective. Like CM, CX employs semantic backpropaga-

Algorithm 4 Competent Crossover algorithm. p_1, p_2 are parents. MIDPOINT is defined in Eq. (4), RANDOMNODE, REPLACE and LIBSEARCH come from Algorithm 3, SB from Algorithm 5.

```
1: function CX(p1, p2)
2:     s_m ← MIDPOINT(p1, p2)
3:     a ← RANDOMNODE(p1)
4:     D ← SB(s_m, p1, a)
5:     D^1_∅ ← SB(s(p1), p1, a)
6:     D^2_∅ ← SB(s(p2), p1, a)
7:     p' ← LIBSEARCH(D, D^1_∅, D^2_∅)
8:     return REPLACE(p1, a, p')
```

tion, this time however in order to produce an offspring that is geometric w.r.t. parent programs.

CX (Algorithm 4) starts with calculating the midpoint s_m between the semantics of the parents p_1 and p_2, i.e., s_m that satisfies $L_z(s(p_1), s_m) = L_z(s_m, s(p_2))$. Construction of s_m is domain-dependent and relies on the following formulas:

$$\begin{array}{ll} s_m = \text{MIDPOINT}(p_1, p_2) \equiv (s(p_1) \wedge s_x) \vee (\neg s_x \wedge s(p_2)) & \text{(Boolean domain)} \\ s_m = \text{MIDPOINT}(p_1, p_2) \equiv \frac{1}{2}(s(p_1) + s(p_2)) & \text{(regression domain)} \end{array} \quad (4)$$

where s_x is a random semantics. To place s_m as closely as possible to the actual midpoint between $s(p_1)$ and $s(p_2)$, s_x for the Boolean is drawn so that the numbers of bits inherited from both parents' differ by at most one in s_m. Next, a node a is drawn in p_1, and CX calls SB(s_m, p_1, a) (Appendix A) to backpropagate s_m in p_1 to a, resulting in the desired semantics D. Then, we attempt to attain effectiveness in a similar way as in CM: the semantics of the parents, i.e., $s(p_1)$ and $s(p_2)$ are backpropagated through p_1 to a, resulting in forbidden semantics $D^1_\varnothing$ and $D^2_\varnothing$, respectively. Next, a library is searched for a program p' that matches D and does not match $D^1_\varnothing$ nor $D^2_\varnothing$ (cf. Appendix B). Finally, p' replaces the subprogram rooted at a in p_1. The second offspring can be created by swapping the parents.

The motivation for using the midpoint as as a temporary goal in the above procedure is providing for better exploration. A candidate offspring located near a parent has by definition very similar semantics and as such usually does not advance much the search process. Promoting locations near to the midpoint helps exploring new areas in semantic space.

Note also that CX is the only operator in the competent suite which explicitly refers to domain's specifics (in (4)). This is because CX *constructs* a new semantics s_m, for which a metric alone is not sufficient. Other operators do not require such constructive means, as they depend only on metric's values for semantics of *existing* programs.

If the library contains a subprogram that matches D, CX is guaranteed to produce a geometric offspring and is thus geometric w.r.t. Def. 2. However, that is not guaranteed for the finite libraries used in practice, so in absence of match, the semantically closest subprogram from the library is returned and planted into the parent program. In such cases CX can be considered approximately geometric, as it produces a good approximation of the midpoint between parents' semantics, given the programs available in the library.

CX is effective, because library search cannot return a subprogram with semantics in $D^1_\varnothing$ or $D^2_\varnothing$, i.e., such that would result in an offspring semantically equal to either parent.

5 Experiments

5.1 Setup

We compare the operators from the suite presented in Sect. 4 (Competent Initialization (CI), Tournament Selection (CTS), Mutation (CM) and Crossover (CX)) to the reference operators in Table 1, i.e., canonical GP operators and the operators that are known to be semantic but not simultaneously effective and geometric. In the first four experiments, we characterize individual operators in isolation from other operators and outside evolutionary runs. The characteristics in question include fitness distribution, effectiveness, geometry, and program size. The fifth experiment (Sect. 5.6) concerns joint performance of operators and verifies whether the competent operators lead to better performance than the reference operators.

We use five univariate and four bivariate symbolic regression problems, and nine Boolean program synthesis problems from Table 3. When choosing the problems, we aimed on one hand at using a diversified suite of problems (different domains, varying number of input variables, various difficulty) taken from a range of sources (the GP-Benchmarks repository and previous works), while on the other hand trying to keep the number of them within a reasonable limit. In univariate regression problems, 20 Chebyshev nodes (Burden and Faires, 2010) in the given range form the training set; the test set comprises 20 points drawn uniformly from the same range. For bivariate problems, 10 values of each variable are analogously selected (Chebyshev nodes for training, uniform drawing for testing), and their Cartesian products form the respective data sets. In the Boolean domain, the training set incorporates all inputs and there is no test set.

Evolutionary runs are set up according to Table 2. Parameters not presented there are set to ECJ's defaults (Luke, 2010). Methods' parameters are not fine-tuned to benchmarks, because in this experiment we aim at side-by-side comparison rather than maximization of performance. The only exception is the mutation step of SGM in symbolic regression domain, because we found out that the default value (1.0) led to very bad performance. In symbolic regression, two floating-point numbers in semantics are considered equal if they differ by less than $2^{-52} \approx 2.22 \times 10^{-16}$ (the difference between 1 and the closest IEEE754 double-precision number). The experimental software is available online at www.cs.put.poznan.pl/tpawlak/link/?ECJCompetent.

5.2 Analysis of initialization

Our hypothesis is that the effective and geometric characteristics of CI are superior to those of RHH and SDI. To verify this, we run these operators 30 times to produce a population of size 1000 for each problem and analyze the resulting initial populations.

For brevity, in this and following experiments we present only aggregate results, while the details and statistical assessment can be found in Appendix C. Unless stated otherwise, the characteristic in question is first averaged over a number of runs for a given operator and problem, then the averages obtained by particular operators are ranked on that problem, and finally the ranks are averaged over the entire problem suite. In Table 4 we present the average ranks of the number of semantically unique programs in initial populations and the ranks of the average number of programs produced until the target t is included in population's L_1-convex hull. Details results are shown in Tables 17 – 18.

The results confirm that both CI and SDI produce populations composed entirely of semantically unique programs. For the semantic-unaware RHH, the fraction of such programs is lower for all problems. This is reflected by the outranking graph in the first

Table 1: Compared operators.

Name		Citation	*Semantic*	*Effective*	*Geometric*
RHH	Ramped Half-and-Half	Koza (1992)			
SDI	Semantically Driven Initialization[a]	Beadle and Johnson (2009a)	✓	✓	
TS	Tournament Selection	Miller and Goldberg (1995)			
STS	Semantic Tournament Selection	Galván-López et al. (2013)	✓		
TM	Tree Mutation	Koza (1992)			
SDM	Semantically Driven Mutation[a]	Beadle and Johnson (2009b)	✓	✓	
SGM	Semantic Geometric Mutation	Moraglio et al. (2012)	✓		✓
TX	Tree Crossover	Koza (1992)			
SDX	Semantically Driven Crossover[a]	Beadle and Johnson (2008)	✓	✓	
SGX	Semantic Geometric Crossover	Moraglio et al. (2012)	✓		✓

[a] SDI, SDM and SDX are designed for semantics represented as binary decision diagrams (BBDs) or sequence of agent moves in the Boolean and controller domains. We augmented them to handle tuple semantics (Sect. 2) by verifying if two tuple semantics are equal, instead of verifying if two BDDs or move sequences are equal.

Table 2: Parameters of evolution.

Generic parameters		Symbolic regression	Boolean domain
Number of runs		30	
Fitness function		L_2 metric	L_1 metric
Population size		1000	
Termination condition		100 generations or find of a program with fitness 0	
Instructions		x_1, x_2, $+$, $-$, $\times$, $/$, sin, cos, exp, log, ERC[a]	$x_1, x_2, ..., x_{11}$,[b] and, or, nand, nor, ERC
Operator-specific parameters			
RHH, SDI, CI	Max tree height	6	
RHH	*Syntactic* duplicate retries	10	
SDI, SDM, SDX	*Semantic* duplicate retries	10	
CI	Convex hull expansion retries	10	
TS, STS, CTS	Tournament size μ	7	
CM, CX	Programs in library	All semantically unique programs built from above instructions (except ERCs) and having height up to	
		4	3
SGM	Random tree source	GROW (Koza, 1992) with tree height in range $[3, 6]$	
	Mutation step	Uniformly drawn from range $[0, 0.02]$	1
SGX	Random tree source	Constant uniformly drawn from range $[0, 1]$	GROW with tree height in range $[3, 6]$

[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 3: Benchmark problems.

Symbolic regression				Boolean domain		
Prob.	Definition (formula)	Range	n	Problem	Instance (bits)	n
R1	$(x_1+1)^3/(x_1^2-x_1+1)$	$[-1,1]$	20	Even parity	Par6	64
R2	$(x_1^5-3x_1^3+1)/(x_1^2+1)$	$[-1,1]$	20		Par7	128
Kj3	$0.3x_1\sin(2\pi x_1)$	$[-3,3]$	20		Par8	256
Kj4	$x_1^3e^{-x_1}\cos(x_1)\sin(x_1)(\sin^2(x_1)\cos(x_1)-1)$	$[0,10]$	20	Multiplexer	Mux11	2048
Kj11	$x_1x_2+\sin((x_1-1)(x_2-1))$	$[-3,3]^2$	100		Mux20	2^{20}
Ng9	$\sin(x_1)+\sin(x_2^2)$	$[0,1]^2$	100	Majority	Maj9	512
Ng12	$x_1^4-x_1^3+\frac{x_2^2}{2}-x_2$	$[0,1]^2$	100		Maj10	1024
Pg1	$1/(1+x_1^{-4})+1/(1+x_2^{-4})$	$[-5,5]^2$	100	Comparator	Cmp8	256
Vl1	$e^{-(x_1-1)^2}/(1.2+(x_2-2.5)^2)$	$[0,6]^2$	100		Cmp10	1024

Table 4: Average ranks over problems and outranking graphs of initialization operators on numbers of semantically unique programs in population P and programs produced until target t is included in population's L_1-convex hull. Details in Tables 17 – 18 in Appendix C.

Operator:	RHH	SDI	CI	Outranking graph
Ranks on number of semantically unique programs in P	3.00	1.50	1.50	CI → SDI, SDI → RHH
Ranks on number of programs produced until $t \in C(P)$	2.06	2.53	1.42	CI → SDI; RHH

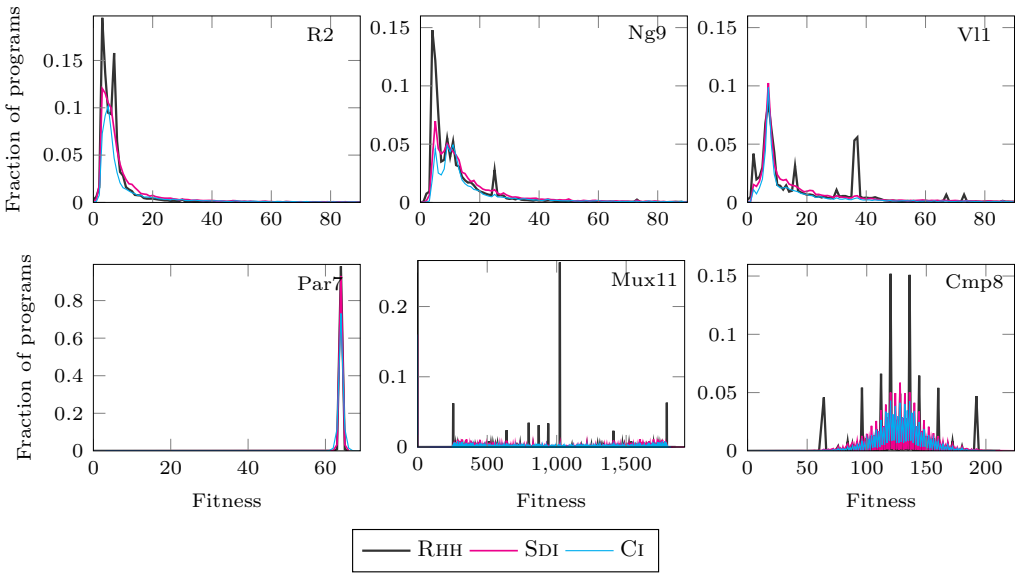


Figure 3: Distribution of fitness in initial population in selected problems.

row of Table 4. Since SDI and CI provide semantic uniqueness by definition, there is no need to statistically assess these differences.

Concerning the number of programs needed to to include the target in an L_1-convex hull of population, the lowest rank is achieved by CI. Table 18 in Appendix C reveals that in 12 out of 18 problems CI needs the lowest number of programs to attain that goal. In the remaining 6 problems, RHH achieves the lowest average and for one problem RHH and CI are on par on this characteristic. The Friedman's test (Kanji, 1999) results in p-value of $p = 2.03 \times 10^{-3}$, suggesting that some pairwise differences are significant. Thus, we conduct a post-hoc analysis using symmetry-test (Hothorn et al., 2015) and present the resulting outranking graph in the second row of Table 4 (the significant differences ($p < 0.05$) are shown as arcs). The graph reveals that CI requires significantly fewer programs to include the target in the convex hull than SDI.

All operators manage to build the desired convex hull with much smaller numbers of programs than the population size considered here (1000). None of the 540 runs performed here failed to produce a convex hull. However, the observed differences may become critical for smaller populations. Also, we hypothesize that in semantic spaces of higher dimensionality (greater n), it would be much harder for RHH and SDI to produce a convex hull that includes t.

For additional insight, we collect the histograms of fitness values of generated programs for a representative sample of problems and present them in Fig. 3. The distributions vary heavily across the problems, while having overall similar shapes for individual operators. RHH produces very rugged distributions with few peaks that correspond to most common semantics. SDI features a lower number of such peaks, and they are typically less prominent. The distributions for CI are most smooth and unimodal for most problems. We hypothesize that the further the produced programs are from the target on average (i.e., the greater the average fitness), the more programs CI has to produce to include the target in the convex hull. This hypothesis is supported by Pearson's correlation coefficient of these two factors, which amounts to 0.49, while the t-test for significance of correlation results in the p-value of 0.01.

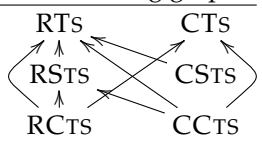
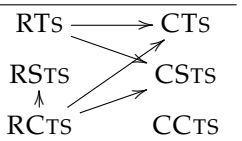
5.3 Analysis of selection

In this experiment we verify whether CTS (Sect. 4.2) is effective and more probable to be t-geometric than STS or TS. To this aim, we apply these operators to populations initialized by RHH and CI. There are thus three setups for RHH's populations: RTS, RSTS, RCTS, and three for CI's populations: CTS, CSTS, CCTS. There are 30 runs for each problem and setup, and each run involves producing an initial population of size 1000 by a given initialization operator, followed by 500 acts of selection of two parents from that population.

Table 5 presents the average ranks over problems on the empirical probability that an application of a given selection operator is effective (first row) and t-geometric under L_1 (second row); see Tables 19 – 22 for detailed results. CTS achieves the best average ranks; its likelihood of selecting programs effectively is close to 1.0 for all problems, and for populations initialized by both algorithms. CTS returned two semantically equal programs only eight times out of over quarter a million selection acts performed in the experiment. STS is second-best, and TS ranks last for both initialization methods. The probabilities for STS and TS are higher for the populations initialized by CI. Friedman's test ($p = 1.16 \times 10^{-12}$) indicates that the only setup not surpassed by both RCTS and CCTS is CSTS.

Concerning the characteristic of being t-geometric selection, recall that for $k = 2$-

Table 5: Average ranks over problems and outranking graphs on probabilities that selection is effective, and t-geometric under L_1. Details in Tables 19 – 22 in Appendix C.

Setup:	RTS	RSTS	RCTS	CTS	CSTS	CCTS	Outranking graph
Ranks on probability that selection is effective	5.94	4.17	1.36	4.78	3.11	1.64	
Ranks on probability that selection is t-geometric under L_1	3.00	3.56	1.67	4.86	4.72	3.19	

ary selection operators, convex hull (Def. 5) is a segment connecting the semantics of selected programs. For L_2 metric, the empirical probability of such an L_2-segment to include t is very low (< 0.001 in preliminary analysis). Thus, in this experiment we use L_1. An L_1-segment (a hyperrectangle) may have significant chance of including any t in low-dimensional spaces (low n), but that becomes much less likely for higher n (and $n \geq 20$ for all problems considered here).

All selection operators achieve higher probabilities of including t in parents' convex hull for the populations initialized by RHH, however conclusive Friedman's test ($p = 1.08 \times 10^{-6}$) shows that the difference between the initialization methods is significant only for TS. In 13 out of 18 problems, RCTS achieves the highest probability, which makes RCTS significantly more t-geometric than STS and CTS. CCTS achieves the highest probability for three problems. Within the groups of setups that use a given initialization method CTS is the most geometric operator.

5.4 Analysis of mutation

We compare the mutation operators from Table 1, i.e., TM, SDM, SGM, and CM, with respect to effectiveness, geometry and impact on program size, by applying them to programs drawn uniformly from populations initialized by RHH and CI. There are four setups operating on RHH-initialized population: RTM, RSDM, RSGM, RCM, and four operating on CI-initialized population: CTM, CSDM, CSGM, CCM. In each of 30 runs per problem, we first initialize the population of size 1000 programs, and then apply the mutation operator to each of them.

Table 6 presents the aggregated ranks on the empirical probability that a mutation is effective (top row), and on the offspring-to-parent L_1 distance (bottom row). SDM features the highest probabilities of effectiveness and ranks best for both initialization methods. CM is the runner-up and TM is the least effective operator for both initialization methods. Friedman's test ($p = 2.13 \times 10^{-10}$) confirms significance of these findings: SDM outranks all other operators on effectiveness.

The definition of geometric mutation is parameterized by the radius r (Def. 1). Rather than setting this parameter in advance and counting how many times a given mutation act is r-geometric, we abstract from r and record the L_1-distance of offspring to parent. In Table 6, we present the ranks on the median of that distance. The setups that use CI produce offspring closer to parents than the corresponding setups with RHH, except for SGM. CTM achieves the overall lowest rank with CCM as the runner-up. According to Friedman's test ($p = 2.87 \times 10^{-9}$) these two setups are indiscernible and

Table 6: Average ranks and outranking graphs on probability that application of mutation is effective and on the median offspring to parent L_1-distance. Details in Tables 23 – 25 in Appendix C.

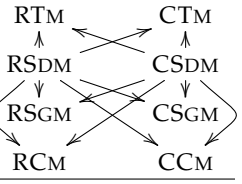
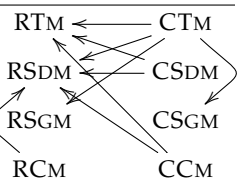
Setup:	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM	Outranking graph
Ranks on probability of effective mutation	6.11	1.50	5.56	4.56	6.89	1.78	4.83	4.78	
Ranks on offspring to parent L_1-distance	6.39	7.00	4.89	4.17	1.83	3.50	5.11	3.11	

Table 7: Average and .95-confidence interval of offspring to parent size ratio. Details for all problems are to be found in Table 26 in Appendix C.

Domain	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
Regression	2.40 ±0.04	2.47 ±0.04	5.16 ±0.07	2.59 ±0.04	1.04 ±0.01	**0.97** ±0.01	1.63 ±0.02	1.08 ±0.01
Boolean	0.86 ±0.01	0.79 ±0.01	1.45 ±0.01	1.29 ±0.02	0.91 ±0.00	0.71 ±0.01	1.07 ±0.00	**0.97** ±0.02

produce offspring closer to parent than RTM and RSDM, and for CTM also RSGM and CSGM. For setups involving RHH, RCM is the best by providing the lowest rank. The offspring produced by RSDM are the most distant from the parents. Note that SGM is parameterized with a mutation step (Moraglio et al., 2012) (which we set as reported in Table 2), so the comparisons with this operator should be taken with a grain of salt.

To compare the operators with respect to offspring size, we record the ratio of the number of nodes in an offspring to the number of nodes in its parent. The ratio of 1.0 indicates that an operator produces on average offspring that is as large as parent. This indicator abstracts from the absolute sizes of programs, which may vary heavily already in the initial populations considered here.

Table 7 presents the average and .95-confidence interval of offspring-to-parent size ratio. The setups involving CI lead to ratios closer to 1.0 than the the corresponding setups with RHH; the exception is SDM in the Boolean domain, where the opposite holds. The reason for this deviation may be that CI, required to provide semantic diversity in the initial population, tends to build larger programs than RHH, and single-subtree mutations affect relatively smaller fraction of larger programs than of smaller programs. The ratio closest to 1.0 is provided by CSDM in the regression domain, and CCM in the Boolean domain. The second closest setup in regression domain is CTM and in the Boolean domain CSGM. Note that the size of offspring produced by CM can be controlled by restricting the size of programs in the library.

5.5 Analysis of crossover

We compare the crossover operators from Table 1, i.e., CX TX, SDX and SGX, by applying them to the programs drawn uniformly from populations initialized by RHH and CI. There are four setups using RHH: RTX, RSDX, RSGX, RCX and four using CI: CTX,

Table 8: Average ranks and outranking graphs on probability that an application of crossover is effective, and on probability that an application effective *and* L_1-geometric. Details in Tables 27 – 30 in Appendix C.

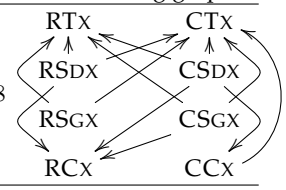
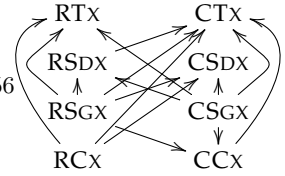
Setup:	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX	Outranking graph
Ranks on probability of effective crossover	7.00	3.06	4.00	5.50	7.44	2.17	2.25	4.58	
Ranks on probability of effective L_1-geometric crossover	6.67	4.78	1.78	3.06	7.72	6.22	1.22	4.56	

Table 9: Average and .95-confidence interval of ratio of offspring's size to average parents' size. Details for all problems are to be found in Table 31 in Appendix C.

Domain	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
Regression	**1.00** ±0.00	1.00 ±0.00	2.35 ±0.00	1.78 ±0.00	0.98 ±0.00	1.00 ±0.00	2.04 ±0.00	0.96 ±0.00
Boolean	**1.00** ±0.00	0.99 ±0.00	2.19 ±0.00	1.01 ±0.00	0.99 ±0.00	0.99 ±0.00	2.03 ±0.00	0.89 ±0.00

CSDX, CSGX, CCX. We run each setup 30 times on each problem, and each run executes 500 applications of crossover.

Table 8 presents the average ranks on the empirical probability that crossover application is effective, and on the empirical probability that it is effective and L_1-geometric (see Tables 27 – 30 in Appendix C for details). Setups that use CI lead to lower ranks than the corresponding RHH setups, except for TX where the opposite holds. CSDX ranks first and CSGX is the close runner-up. For each initialization method TX is least likely to be effective, and CX comes as last-but-one in the ranking. It is however worth to note that the probabilities achieved by CX (Table 27) are still high: over .99 for symbolic regression and in the range .47–.87. for the Boolean domain. Friedman's test confirms significance of these results with $p = 6.50 \times 10^{-10}$.

Concerning the empirical probability that crossover is effective and geometric under L_1 metric, we exclude the neutral applications because they do not advance search, even though they are geometric according to Def. 2. Unsurprisingly, the exact geometric SGX scores the lowest ranks for both initialization methods and thus is the most geometric operator, with probabilities in the range .74–.999 (and 1.0 when including the neutral applications, see Table 29). The conclusive result of Friedman's test ($p = 7.22 \times 10^{-15}$) shows that this result is better than results of all other setups except RCX. The ranks achieved by SGX are lower when a population is initialized by CI, which confirms that geometric initialization helps SGX to produce geometric effective offspring. The runner-up is CX, for which the rank is lower for the population initialized by RHH. The least geometric operator is TX.

Table 9 presents the average and .95-confidence interval of the ratio of the number of nodes in the offspring to the average number of nodes in the parents (for the rationale on ratio-based measures, see Sect. 5.4). RTX diverges the least from the neutral value

of 1.0 in both considered domains. The more divergent are, in order, RSDX, CSDX, CTX. Note that the differences between the ratios of those four setups are minute: all of them range in the interval $[0.98, 1.0]$. CX maintains higher ratios in RHH's populations ($[1.01, 1.78]$) than in CI's populations, ($[0.89, 0.96]$). As in the case of mutation, the size of the offspring produced by CX can be tuned by restricting the size of programs in the library. Because SGX combines parents using an extra code piece, it produces offspring that are on average over twice as large as the parents.

5.6 Joint performance of operators

We assess the joint performance of the operators analyzed in previous sections. We compare four configurations of the generational GP algorithm equipped with the operators from Table 1: (i) the canonical operators: RHH, TS, TX and TM in proportions 90:10 (RTSTXM), (ii) the effective non-geometric operators: SDI, STS, SDX and SDM in proportions 50:50 (SSTSSDXM), (iii) the geometric non-effective operators: RHH, TS, SGX and SGM in proportions 50:50 (RTSSGXM) and (iv) the competent operators: CI, CTS, CX and CM in proportions 50:50 (CCTSCXM). In this experiment, we replace the Mux20 problem by 16-bit Comparator (Cmp16), because calculating the 2^{20}-bit long semantics in Mux20 for every (sub-)program in every generation required over 24 hours per run on high-performance servers, while the 2^{16}-bit long semantics in Cmp16 results in substantially shorter time.

Figure 4 and Table 10 present the average and $.95$-confidence interval of the best-of-generation and the best-of-run fitness, respectively. In symbolic regression, CCTSCXM is the unquestionable winner in terms of best-of-run fitness and speed of convergence. No other setup outperforms CCTSCXM throughout entire runs. In the Boolean domain CCTSCXM achieves the best-of-run fitness in four out of nine problems. In the remaining five problems RTSSGXM is the best. The outcome of Friedman's test ($p = 1.39 \times 10^{-6}$, Table 11) confirms the superiority of CCTSCXM over all other setups except SSTSSDXM.

Table 12 shows the median and $.95$-confidence interval of the test-set fitness of the best-of-run program on the training set. In seven out of nine problems CCTSCXM outperforms all other setups while RTSTXM proves best twice. Friedman's test ($p = 6.91 \times 10^{-4}$, Table 13) shows that RTSSGXM generalizes significantly worse than all setups except the canonical one: RTSTXM. Notice that all setups overfit on the Pg1 problem, with CCTSCXM overfitting particularly strongly. We explain this phenomenon by the use of Chebyshev nodes as training set points (cf. Sect. 5.1). This choice tends to sample the target function more densely at the extremes of training range, while undersampling the center of this range. The values of the Pg1 function (cf. Table 3) vary much stronger in the center of the training range than close to the boundaries of that range. Thus, only a small fraction of training data reflects the most varying (and thus most decisive for the fitness value) part of the sought function.

Table 14 presents the average and $.95$-confidence interval of the number of nodes in the best-of-run program. CCTSCXM produces the smallest programs in all Boolean problems and Ng9, while RTSTXM produces the smallest programs in seven out of nine symbolic regression problems. RTSSGXM produces programs that are significantly larger (Friedman's $p = 4.03 \times 10^{-7}$, Table 15) than the programs produced by the all other setups.

6 Discussion

The consistency of analytical investigations in Sects. 5.2–5.5 with the overall good results produced by the competent suite in Sect. 5.6 demonstrate that effectiveness and

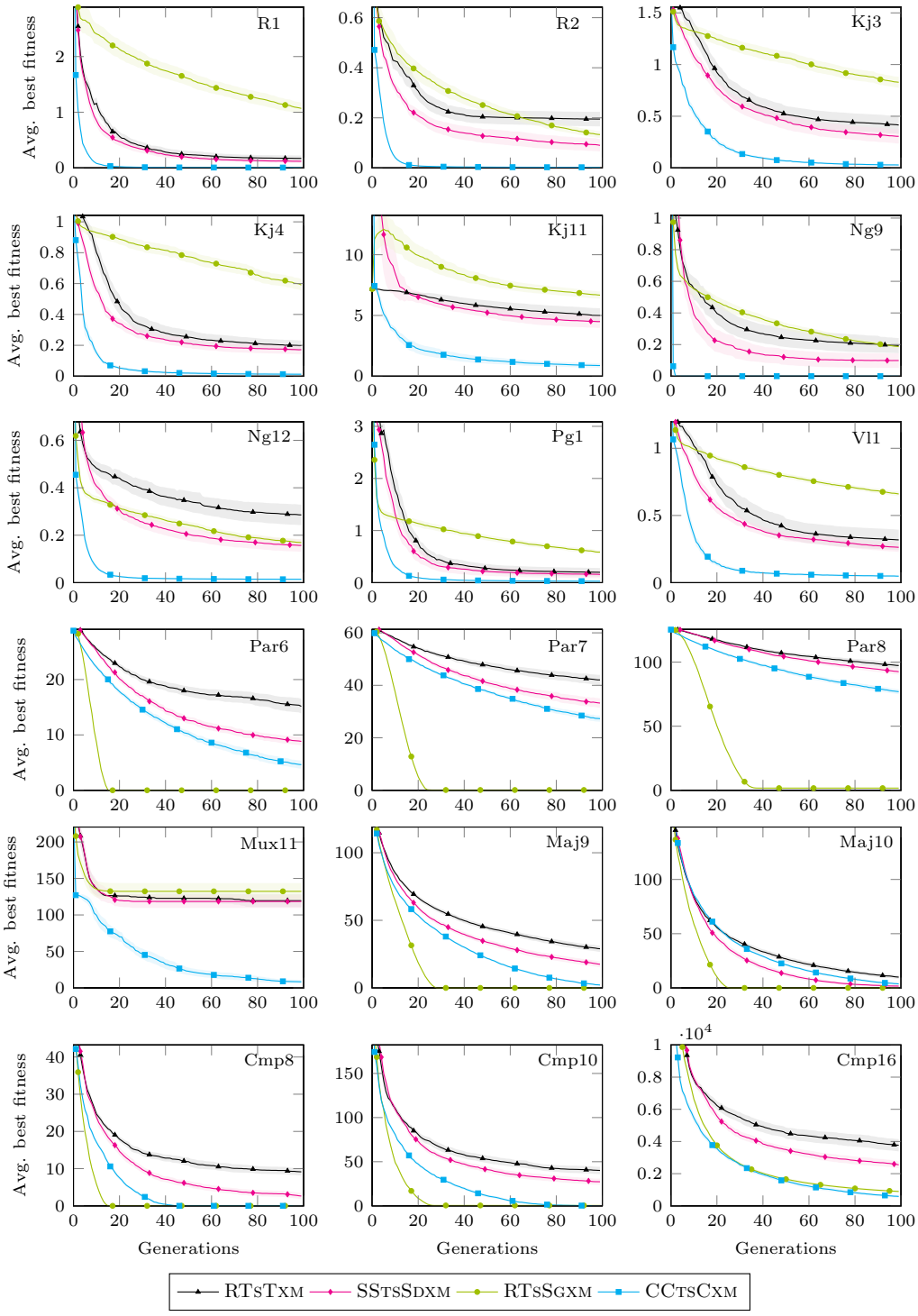


Figure 4: The average and .95-confidence interval of the best-of-generation fitness.

Table 10: The average and .95-confidence interval of the best-of-run fitness.

Problem	RTsTxm		SStsSdxm		RTsSgxm		CCtsCxm	
R1	0.171	±0.052	0.119	±0.032	1.051	±0.102	**0.006**	±0.001
R2	0.195	±0.029	0.089	±0.024	0.131	±0.013	**0.003**	±0.001
Kj3	0.414	±0.085	0.304	±0.069	0.828	±0.053	**0.028**	±0.007
Kj4	0.197	±0.034	0.171	±0.030	0.589	±0.038	**0.012**	±0.010
Kj11	4.995	±0.591	4.492	±0.516	6.658	±0.297	**0.874**	±0.181
Ng9	0.193	±0.048	0.099	±0.046	0.186	±0.014	**0.000**	±0.000
Ng12	0.286	±0.043	0.157	±0.027	0.169	±0.012	**0.014**	±0.003
Pg1	0.196	±0.081	0.158	±0.071	0.586	±0.037	**0.029**	±0.005
Vl1	0.318	±0.076	0.262	±0.033	0.659	±0.017	**0.048**	±0.007
Par6	15.167	±1.307	8.867	±0.726	**0.000**	±0.000	4.567	±0.743
Par7	42.033	±1.926	33.100	±1.823	**0.100**	±0.107	27.033	±1.163
Par8	97.167	±2.490	92.533	±2.265	**1.667**	±0.427	76.733	±1.782
Mux11	119.533	±8.712	118.433	±8.833	132.333	±11.252	**8.167**	±2.574
Maj9	28.767	±2.313	17.433	±2.128	**0.000**	±0.000	2.033	±0.581
Maj10	9.833	±1.653	1.600	±0.809	**0.000**	±0.000	3.567	±1.007
Cmp8	9.033	±0.953	2.633	±0.776	**0.000**	±0.000	**0.000**	±0.000
Cmp10	40.067	±4.159	27.033	±3.355	0.433	±0.239	**0.100**	±0.107
Cmp16	3762.000	±379.835	2541.533	±220.290	887.733	±73.149	**578.381**	±79.870
Rank:	3.611		2.389		2.639		1.361	

Table 11: Post-hoc analysis of Friedman's test ($p = 1.39 \times 10^{-6}$) using symmetry test conducted on Table 10: p-values of incorrectly judging an operator in a row as better than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTsTxm	SStsSdxm	RTsSgxm	CCtsCxm
RTsTxm				
SStsSdxm	**0.023**		0.937	
RTsSgxm	0.106			
CCtsCxm	**0.000**	0.078	**0.015**	

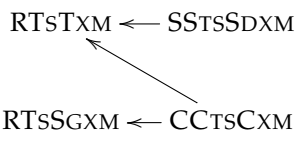


Table 12: The median and .95-confidence interval of test-set fitness of the best of run program on training set.

Problem	RTsTxm		SStsSdxm		RTsSgxm		CCtsCxm	
R1	0.11 ≤	0.15 ≤ 0.30	0.05 ≤	0.09 ≤ 0.20	18.25 ≤	18.30 ≤ 18.40	0.00 ≤	**0.00** ≤ 0.01
R2	0.07 ≤	0.11 ≤ 0.16	0.04 ≤	0.08 ≤ 0.18	3.36 ≤	3.37 ≤ 3.38	0.00 ≤	**0.00** ≤ 0.00
Kj3	1.14 ≤	**1.28** ≤ 1.70	1.11 ≤	1.35 ≤ 1.88	2.18 ≤	2.23 ≤ 2.32	1.11 ≤	1.40 ≤ 1.95
Kj4	0.62 ≤	1.38 ≤ 2.49	1.02 ≤	1.26 ≤ 3.62	2.31 ≤	2.36 ≤ 2.43	0.05 ≤	**0.12** ≤ 0.17
Kj11	6.85 ≤	10.46 ≤ 26.19	6.81 ≤	7.16 ≤ 9.95	49.57 ≤	49.69 ≤ 49.89	1.76 ≤	**3.66** ≤ 11.35
Ng9	0.07 ≤	0.12 ≤ 0.19	0.00 ≤	0.01 ≤ 0.10	6.05 ≤	6.06 ≤ 6.07	0.00 ≤	**0.00** ≤ 0.00
Ng12	0.10 ≤	0.13 ≤ 0.21	0.05 ≤	0.07 ≤ 0.10	2.17 ≤	2.17 ≤ 2.18	0.01 ≤	**0.01** ≤ 0.02
Pg1	3.30 ≤	**3.89** ≤ 5.10	3.54 ≤	4.25 ≤ 4.94	7.56 ≤	7.60 ≤ 7.63	16.91 ≤	38.04 ≤ 157.62
Vl1	0.41 ≤	0.80 ≤ 1.45	0.49 ≤	0.76 ≤ 1.55	3.50 ≤	3.52 ≤ 3.57	0.11 ≤	**0.15** ≤ 0.23
Rank:	2.56		2.00		3.89		1.56	

Table 13: Post-hoc analysis of Friedman's test ($p = 6.91 \times 10^{-4}$) using symmetry test conducted on Table 12: p-values of incorrectly judging an operator in a row as generalizing better than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTsTxm	SStsSDxm	RTsSGxm	CCtsCxm
RTsTxm			0.126	
SStsSDxm	0.798		**0.010**	
RTsSGxm				
CCtsCxm	0.354	0.885	**0.001**	

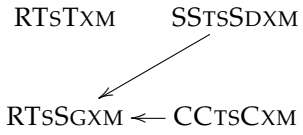
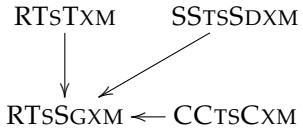


Table 14: The average and .95-confidence interval of number of nodes in the best of run program. Values $\geq 10^4$ are rounded to an order of magnitude.

Problem	RTsTxm		SStsSDxm		RTsSGxm		CCtsCxm	
R1	**121.10**	±16.82	137.23	±22.37	10^{17}	$\pm 10^{16}$	432.17	±27.26
R2	**86.77**	±17.07	125.50	±21.11	10^{17}	$\pm 10^{17}$	364.20	±31.29
Kj3	**150.30**	±23.82	182.23	±27.05	10^{17}	$\pm 10^{16}$	343.53	±29.78
Kj4	**169.00**	±21.14	190.70	±22.39	10^{17}	$\pm 10^{18}$	277.83	±43.63
Kj11	**187.13**	±44.50	235.67	±29.34	10^{17}	$\pm 10^{16}$	353.21	±35.82
Ng9	86.00	±13.40	63.30	±14.12	10^{17}	$\pm 10^{16}$	**7.20**	±0.28
Ng12	**70.67**	±17.52	118.03	±17.98	10^{17}	$\pm 10^{17}$	466.70	±50.76
Pg1	93.93	±12.81	**91.80**	±15.87	10^{17}	$\pm 10^{16}$	413.27	±30.44
Vl1	**126.27**	±15.10	155.27	±24.06	10^{17}	$\pm 10^{16}$	316.80	±25.02
Par6	451.07	±42.17	997.73	±85.91	10^{6}	$\pm 10^{5}$	**309.20**	±28.09
Par7	495.80	±53.29	1103.47	±128.26	10^{16}	$\pm 10^{16}$	**298.87**	±28.35
Par8	492.80	±38.55	1039.93	±91.49	10^{17}	$\pm 10^{16}$	**331.07**	±30.69
Mux11	179.27	±33.51	156.67	±24.37	10^{17}	$\pm 10^{16}$	**119.07**	±9.56
Maj9	542.20	±39.54	1065.53	±88.65	10^{10}	$\pm 10^{9}$	**313.47**	±6.96
Maj10	589.00	±48.32	1007.20	±60.34	10^{9}	$\pm 10^{8}$	**292.13**	±13.11
Cmp8	352.67	±35.96	575.67	±51.92	10^{5}	$\pm 10^{5}$	**99.87**	±7.59
Cmp10	343.87	±36.88	588.47	±61.11	10^{16}	$\pm 10^{16}$	**155.87**	±13.32
Cmp16	240.47	±28.44	478.13	±46.19	10^{17}	$\pm 10^{17}$	**196.14**	±16.21
Rank:	1.72		2.39		4.00		1.89	

Table 15: Post-hoc analysis of Friedman's test ($p = 4.03 \times 10^{-7}$) using symmetry test conducted on Table 14: p-values of incorrectly judging an operator in a row as producing smaller programs than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTsTxm	SStsSDxm	RTsSGxm	CCtsCxm
RTsTxm		0.408	**0.000**	0.980
SStsSDxm			**0.001**	
RTsSGxm				
CCtsCxm		0.651	**0.000**	



geometric character of semantic GP operators are viable performance predictors. The particular indicators and quantities proposed and reported in Sects. 5.2–5.5 can be thus used as guides for designing initialization, selection, and search operators.

Although the components proposed here do not adhere strictly to the formal definitions of geometric properties (Defs. 2–4), they are designed with geometry and effectiveness in mind, and trade the geometric perfection in exchange for practically acceptable computational overhead. In this sense, we find them a viable alternative to the exact geometric operators proposed by Moraglio et al. (2012). As elegant as they are, the exact operators represent one extreme on the spectrum of the above trade-off, and often pay for good training-set performance with poor generalization (Table 12) and skyrocketing program size (Table 14). Though program size can be to an extent managed by simplification or reuse of subexpressions (Vanneschi et al., 2013), generalization remains an challenge. In machine learning terms, the exact operators as proposed by Moraglio et al. (2012) have no bias and, consequently, exhibit high variance (Geman et al., 1992) when confronted with a test set. This problem may be addressed by *geometric block mutations* (Moraglio et al., 2013) that are biased by modifying program semantics on several fitness cases simultaneously. However these operators have been proposed for Boolean domain only, and assessment of their generalization performance was beyond the scope of that study. The competent operators have also nonzero bias, which apparently helps them generalize well, as verified empirically in this study.

An interesting side-result of the experiment conducted in this work is the worse than expected performance of the exact geometric approach (RTSSGXM) in the regression domain. In contrast to previous works that reported excellent results (including the original Moraglio et al. (2012) paper), we found it hard to optimize RTSSGXM's configuration for performance in the continuous domain. Figure 4 and tables reflect the best configuration obtained by using various mutation steps and size of the random 'mixing' tree in SGM (SGX for L_2 is parameter-free). This tuning made it possible for RTSSGXM to catch up with the remaining configurations in three out of nine benchmarks, but did not help generalization: in terms of test-set fitness, RTSSGXM clearly ranks last in Table 12. Our working explanation for the observed underperformance points to three possible causes. Firstly, the continuous benchmarks in our suite are hard (Table 3): apart from one polynomial (Ng12), all problems involve rational, transcendental or periodic functions. Secondly, the instruction set for regression problems includes the trigonometric, exponential and logarithmic functions and is broader than in most previous studies that use polynomial instructions only (Moraglio et al., 2012; Vanneschi et al., 2013; Zhu et al., 2013; Castelli et al., 2012, 2013a,b, 2014, 2015). The latter characteristic may make it harder to provide a desirable magnitude of variation necessary for SGM to safely converge to the target.

7 Conclusions

We demonstrated that the capabilities of GSGP can be substantially boosted by extending geometric considerations beyond the basic framework of search operators to population initialization and selection, and by equipping those components with mechanisms that make them effective. The empirical results clearly show that these extensions prove successful in practice for the Boolean and symbolic regression domains (Sect. 5.6), while the experiments in Sects. 5.2–5.5 revealed the causes of these observations.

The proposed suite of competent operators represents a 'holistic' approach to semantic GP, where *all* key components of evolutionary search infrastructure are designed with semantic and geometric aspects in mind. As the experimental results demonstrate,

this is beneficial for the overall search performance for the two domains considered in this paper, which proves that there is more to GSGP than just search operators. On the other hand, it is clear that the particular components proposed here form just one of many conceivable suites, and the quest for other, possibly even better performing toolkits should continue. Designing population initialization operators and selection operators that guarantee – in deterministic or stochastic sense – the convex hull of population to include the target seems particularly appealing. Another interesting challenge for future research is defining competent operators tailored to other domains, and their empirical assessment.

Acknowledgments

This work was supported by grant no. 2014/15/B/ST6/05205, funded by the National Science Centre, Poland.

A Combinatorial Semantics and Semantic Backpropagation

Semantic backpropagation (SB, Pawlak et al. (2015b); Krawiec and Pawlak (2013a); Wieloch and Krawiec (2013)) is an algorithm employed by CM and CX (cf. Sects. 4.3 and 4.4) that, given a semantics s, a program p, and subprogram a in p (i.e., its location in p), computes the *combinatorial semantics*, a formal object that captures the semantics which, when substituted for a in p, cause the resulting program to have semantics equal to s. A combinatorial semantics is a tuple $D = (D_1, D_2, ..., D_n)$ of n sets $D_i \subseteq \mathcal{O}$, where D_i corresponds to the ith fitness case (i.e., the ith dimension of the considered semantic space). D_i stores the values that, when plugged at location a into p applied to ith fitness case, cause p to return the ith element of the semantics s. Because an application of p to every fitness case is independent, D implicitly represents the set of semantics resulting from the Cartesian product of D_is, i.e., $\Pi_{i=1}^{n} D_i$. This allows us to capture an exponential number of semantics within a compact formal object. For instance, for $n = 2$, $D = (D_1, D_2) = (\{4, 5\}, \{2, 3\})$ is equivalent to the set of four semantics $\{(4, 2), (4, 3), (5, 2), (5, 3)\}$. The transformations between semantics and combinatorial semantics will be assumed implicit in the following. A program p matches the desired semantics D if semantics $s(p)$ is in the set of semantics represented by D, i.e., $s(p) \in \Pi_{i=1}^{n} D_i$.

SB (Algorithm 5) calculates the combinatorial semantics by inverting program execution for the considered semantics. For each component D_i of D, SB traverses the path of instructions from the program root node to a designated node a (where the subtree to calculate combinatorial semantics for is rooted). In each node r and its desired output o, SB inverts the execution of r w.r.t. the output of the next node c_k (the kth child) on the path toward a by calling INVERT(r, k, o). INVERT executes an inverted instruction r^{-1} for which r outputs o, i.e., $o = r(c_1, c_2, ..., c_k, ..., c_n) \Rightarrow c_k = r^{-1}(o, c_1, c_2, ...c_n)$, where c_is are the outputs of r's children nodes. The resulting c_k becomes the desired output for the next node on the path. SB terminates when a is reached. The time complexity of SB is $O(nm)$, where n is the dimensionality of semantic space and m is the number of nodes in program p.

Realization of INVERT is domain-dependent. Table 16 defines it for the common instructions from the regression and Boolean domains. Depending on the properties of the instruction r that is subject to inversion, four cases can be delineated when calling INVERT(r, k, o).

Table 16: Definition of $\text{INVERT}(r, k, o)$ for regression and Boolean domains. For an instruction r with arguments c_i the formulas determine the value for the first ($k = 1$, center column) and the second ($k = 2$, right column) argument, given the output value o for r. Set symbols are omitted for clarity where there is only a one value.

	Subprogram r	$\text{INVERT}(r, 1, o)$	$\text{INVERT}(r, 2, o)$
Regression domain	$c_1 + c_2$	$o - c_2$	$o - c_1$
	$c_1 - c_2$	$o + c_2$	$c_1 - o$
	$c_1 \times c_2$	$\begin{cases} o/c_2 & c_2 \neq 0 \\ \mathbb{R} & c_2 = 0 \wedge o = 0 \\ \emptyset & c_2 = 0 \wedge o \neq 0 \end{cases}$	$\begin{cases} o/c_1 & c_1 \neq 0 \\ \mathbb{R} & c_1 = 0 \wedge o = 0 \\ \emptyset & c_1 = 0 \wedge o \neq 0 \end{cases}$
	c_1/c_2	$\begin{cases} o \times c_2 & c_2 \neq \pm\infty \\ \mathbb{R} & c_2 = \pm\infty \wedge o = 0 \\ \emptyset & c_2 = \pm\infty \wedge o \neq 0 \end{cases}$	$\begin{cases} c_1/o & c_1 \neq 0 \\ \mathbb{R} & c_1 = 0 \wedge o = 0 \\ \emptyset & c_1 = 0 \wedge o \neq 0 \end{cases}$
	$\exp(c_1)$	$\begin{cases} \log o & o \geq 0 \\ \emptyset & \text{otherwise} \end{cases}$	—
	$\log \lvert c_1 \rvert$	$\{-e^o, e^o\}$	—
	$\sin c_1$	$\begin{cases} \{\arcsin o - 2\pi, \arcsin o\} & \lvert o \rvert \leq 1 \\ \emptyset & \text{otherwise} \end{cases}$	—
	$\cos c_1$	$\begin{cases} \{\arccos o - 2\pi, \arccos o\} & \lvert o \rvert \leq 1 \\ \emptyset & \text{otherwise} \end{cases}$	—
Boolean domain	not c_1	not o	—
	c_1 and c_2	$\begin{cases} o & c_2 \\ \mathbb{B} & \text{not } c_2 \text{ and not } o \\ \emptyset & \text{not } c_2 \text{ and } o \end{cases}$	$\begin{cases} o & c_1 \\ \mathbb{B} & \text{not } c_1 \text{ and not } o \\ \emptyset & \text{not } c_1 \text{ and } o \end{cases}$
	c_1 or c_2	$\begin{cases} o & \text{not } c_2 \\ \mathbb{B} & c_2 \text{ and } o \\ \emptyset & c_2 \text{ and not } o \end{cases}$	$\begin{cases} o & \text{not } c_1 \\ \mathbb{B} & c_1 \text{ and } o \\ \emptyset & c_1 \text{ and not } o \end{cases}$
	c_1 nand c_2	$\begin{cases} \text{not } o & c_2 \\ \mathbb{B} & \text{not } c_2 \text{ and } o \\ \emptyset & \text{not } c_2 \text{ and not } o \end{cases}$	$\begin{cases} \text{not } o & c_1 \\ \mathbb{B} & \text{not } c_1 \text{ and } o \\ \emptyset & \text{not } c_1 \text{ and not } o \end{cases}$
	c_1 nor c_2	$\begin{cases} \text{not } o & \text{not } c_2 \\ \mathbb{B} & c_2 \text{ and not } o \\ \emptyset & c_2 \text{ and } o \end{cases}$	$\begin{cases} \text{not } o & \text{not } c_1 \\ \mathbb{B} & c_1 \text{ and not } o \\ \emptyset & c_1 \text{ and } o \end{cases}$
	c_1 xor c_2	c_2 xor o	c_1 xor o

Algorithm 5 Semantic Backpropagation algorithm. D is the combinatorial semantics that is to be propagated to a designated node a in program p to calculate the combinatorial semantics for a. CHILD(r, a) is a child of node r on the path from r to a, and POS(r, a) is a's position in the list of children of r. INVERT is defined in Table 16. $\mathbb{D}$ is the entire domain, e.g., $\mathbb{R}$ or $\mathbb{B} = \{0, 1\}$. Note that if SB is supplied with semantics s instead of combinatorial semantics D, the type cast would be implicit.

```
1:  function SB(D, p, a)
2:      for all D_i ∈ D do                                   ▷ For each input
3:          r ← p
4:          while r ≠ a ∧ D_i ≠ ∅ ∧ D_i ≠ 𝔻 do
5:              k ← POS(r, a)
6:              D'_i ← ∅
7:              for all o ∈ D_i do
8:                  D'_i ← D'_i ∪ INVERT(r, k, o)
9:              D_i ← D'_i
10:             r ← CHILD(r, a)
11:     return D
```

- If r is a bijection[2], it is fully invertible and INVERT returns a single value of the argument in question, calculated from the output and the remaining arguments. Examples are addition, subtraction, negation and xor.
- If r is a non-injective instruction, i.e., maps multiple argument values to the same output, there are many (possibly infinitely many) inversions and INVERT outputs a set. Examples include the absolute value $|\cdot|$ and $\sin(x)$.
- If the considered argument has no impact on r's output and the actual output of r is equal to the given one, INVERT returns the entire domain $\mathbb{D}$, because whatever value from $\mathbb{D}$ is fed into the considered argument, r's output remains correct. Examples are multiplication by 0 and Boolean conjunction with $false$.
- Finally, if r is a non-surjective instruction and the requested output value o is not in its image and thus cannot be returned by r, INVERT returns $\emptyset$.

Note that all these four cases may co-occur for the same instruction for different values of the requested output o, i.e., for different fitness cases.

B Efficient library search

The LIBSEARCH algorithm searches a library of programs for a subprogram that matches a given desired combinatorial semantics D as close as possible and simultaneusly does not match (one or more) forbidden combinatorial semantics $D_\varnothing$. The core of the algorithm can be briefly expressed as

$$\text{LIBSEARCH}(D, D^1_\varnothing, D^2_\varnothing, ...) = \arg\min_{p \in L'} \min_{s \in \Pi_i D_i} \|s(p), s\| \quad (5)$$

$$L' = \{p : p \in L \cup \mathbb{D}, \forall_{D^j_\varnothing} \forall_{s_\varnothing \in \Pi_i D^j_{\varnothing i}} s(p) \neq s_\varnothing\} \quad (6)$$

[2]For multi-argument instructions r, we consider bijection w.r.t. the argument that is to be found by inversion, assuming other arguments are fixed. For instance, $x + y \to z$ is bijective w.r.t. x assuming fixed y.

where D is a desired semantics, and $D^1_{\varnothing}, D^2_{\varnothing}, ...$ are forbidden semantics (cf. Sects. 4.3–4.4).

The set of programs L' searched by LIBSEARCH is a is union of a set of programs L that come from an arbitrary source (e.g., are created in advance) and a set of all single-node constant programs $\mathbb{D}$. Programs that match any forbidden semantics $D^j_{\varnothing}$ are excluded from L' (see definition of matching in Appendix A). Eq. (5) calculates the minimum distance of a program in L' to each semantics that comes from the Cartesian product of components of D (Appendix A).

Below we present how to efficiently calculate Eq. (5) for Minkowski metric. Let us substitute a metric with Minkowski metric:

$$\text{LIBSEARCH}(D, D^1_{\varnothing}, D^2_{\varnothing}, ...) = \arg\min_{p \in L'} \min_{s \in \Pi_i D_i} \left(\sum_{j=1}^{n} |s_j(p) - s_j|^z \right)^{\frac{1}{z}} \tag{7}$$

This can be transformed to (details in Pawlak (2015, Ch7)):

$$\text{LIBSEARCH}(D, D^1_{\varnothing}, D^2_{\varnothing}, ...) = \arg\min_{p \in L'} \sum_{j=1}^{n} \min_{s_k \in D_j} |s_j(p) - s_k|^z \tag{8}$$

This formula reveals that the distance to the semantics represented by D can be minimized separately for each component of D, eliminating so the need of computing the Cartesian product. Eq. (8) can be calculated in $O(n|L'| \log |D_i|)$ time, provided that the elements of D_is are sorted.

To limit the cardinality of L', we constrain the set of constants in $\mathbb{D}$. We observe that each D_j in Eq. (8) defines its own marginal distance. Such a distance has minima where $|c - s_k|^z = 0$, i.e., $c = s_k$. Therefore, the sum of marginal distances cannot have minima for constants beyond the following set:

$$\mathbb{D}' \quad = \quad \{c : c \in \mathbb{D}, \min \bigcup_{D_i \in D} D_i \leq c \leq \max \bigcup_{D_i \in D} D_i\} \tag{9}$$

For L_1 metric ($z = 1$), the term under summation in Eq. (8) reduces to piecewise linear function with roots in D_i and no plateaus. Since the sum of piecewise linear functions is a piecewise linear function too and has minima in the same points where the summed functions, i.e., $\bigcup_{D_i \in D} D_i$, we may narrow the set of candidate minima $\mathbb{D}'$ to:

$$\mathbb{D}'' = \bigcup_{D_i \in D} D_i \tag{10}$$

In this way, the overall set of constants is greatly reduced.For $z \neq 1$, the term under $\min$ in Eq. (8) is non-linear and we cannot reduce $\mathbb{D}'$ this way.

For domains where $\mathbb{D}'$ has low cardinality, e.g., the Boolean one, it is relatively cheap to search the entire set $\mathbb{D}'$. For the regression domain, $\mathbb{D}'$ may be still infinite, and it may be necessary to calculate the value that minimizes Eq. (8) using e.g., a numerical optimization technique or approximate the result by searching the reduced set $\mathbb{D}''$ like for L_1 (what we do in the experiment) .

References

Beadle, L. and Johnson, C. (2008). Semantically driven crossover in genetic programming. In *IEEE CEC'08*, pages 111–116, Hong Kong. IEEE Computational Intelligence Society, IEEE Press.

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

Beadle, L. and Johnson, C. G. (2009b). Semantically driven mutation in genetic programming. In *IEEE CEC'09*, pages 1336–1342, Trondheim, Norway. IEEE Computational Intelligence Society, IEEE Press.

Burden, R. and Faires, J. (2010). *Numerical Analysis*. Cengage Learning.

Carathéodory, C. (1911). über den variabilitätsbereich der fourierschen konstanten von positiven harmonischen funktionen. *Rendiconti del Circolo Matematico di Palermo*, 32:193–217.

Castelli, M., Castaldi, D., Giordani, I., Silva, S., Vanneschi, L., Archetti, F., and Maccagnola, D. (2013a). An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics. In Correia, L., Reis, L. P., and Cascalho, J., editors, *Proceedings of the 16th Portuguese Conference on Artificial Intelligence, EPIA 2013*, volume 8154 of *Lecture Notes in Computer Science*, pages 78–89, Angra do Heroismo, Azores, Portugal. Springer.

Castelli, M., Manzoni, L., and Vanneschi, L. (2012). An efficient genetic programming system with geometric semantic operators and its application to human oral bioavailability prediction. arXiv.

Castelli, M., Vanneschi, L., and Popovic, A. (2015). Predicting burned areas of forest fires: an artificial intelligence approach. *Fire Ecology*, 11(1):106–118.

Castelli, M., Vanneschi, L., and Silva, S. (2013b). Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. *Expert Systems with Applications*, 40(17):6856–6862.

Castelli, M., Vanneschi, L., and Silva, S. (2014). Prediction of the unified parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. *Expert Systems with Applications*, 41(10):4608–4616.

Galván-López, E., Cody-Kenny, B., Trujillo, L., and Kattan, A. (2013). Using semantics in the selection mechanism in genetic programming: a simple method for promoting semantic diversity. In *IEEE CEC'13*, volume 1, pages 2972–2979, Cancun, Mexico.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58.

Hothorn, T., Hornik, K., van de Wiel, M. A., and Zeileis, A. (2015). Package 'coin': Conditional inference procedures in a permutation test framework.

Jackson, D. (2010a). Phenotypic diversity in initial genetic programming populations. In *EuroGP'10*, volume 6021 of *LNCS*, pages 98–109, Istanbul. Springer.

Jackson, D. (2010b). Promoting phenotypic diversity in genetic programming. In *PPSN'10*, volume 6239 of *LNCS*, pages 472–481, Krakow, Poland. Springer.

Kanji, G. (1999). *100 Statistical Tests*. SAGE Publications.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Krawiec, K. and Lichocki, P. (2009). Approximating geometric crossover in semantic space. In *GECCO'09*, pages 987–994, Montreal. ACM.

Krawiec, K. and Pawlak, T. (2013a). Approximating geometric crossover by semantic backpropagation. In *GECCO'13*, pages 941–948, Amsterdam, The Netherlands. ACM.

Krawiec, K. and Pawlak, T. (2013b). Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, 14(1):31–63.

Looks, M. (2007). On the behavioral diversity of random programs. In *GECCO'07*, volume 2, pages 1636–1642, London. ACM Press.

Luke, S. (2010). *The ECJ Owner's Manual – A User Manual for the ECJ Evolutionary Computation Library*, zeroth edition, online version 0.2 edition.

Miller, B. L. and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212.

Moraglio, A. (2011). Abstract convex evolutionary search. In Beyer, H.-G. and Langdon, W. B., editors, *Foundations of Genetic Algorithms*, pages 151–162, Schwarzenberg, Austria. ACM.

Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). Geometric semantic genetic programming. In *PPSN XII*, volume 7491 of *LNCS*, pages 21–31, Taormina, Italy. Springer.

Moraglio, A. and Mambrini, A. (2013). Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In *GECCO'13*, pages 989–996, Amsterdam, The Netherlands. ACM.

Moraglio, A., Mambrini, A., and Manzoni, L. (2013). Runtime analysis of mutation-based geometric semantic genetic programming on boolean functions. In *Foundations of Genetic Algorithms*, pages 119–132, Adelaide, Australia. ACM.

Moraglio, A., McDermott, J., and O'Neill, M. (2014). Geometric semantic grammatical evolution. In *Semantic Methods in Genetic Programming*, Ljubljana, Slovenia. Workshop at Parallel Problem Solving from Nature 2014 conference.

Nguyen, Q. U., Nguyen, X. H., and O'Neill, M. (2009). Semantics based mutation in genetic programming: The case for real-valued symbolic regression. In *Mendel'09*, pages 73–91, Brno, Czech Republic.

Nguyen, Q. U., Pham, T. A., Nguyen, X. H., and McDermott, J. (2016). Subtree semantic geometric crossover for genetic programming. *Genetic Programming and Evolvable Machines*. Online first.

Pawlak, T. (2014). Combining semantically-effective and geometric crossover operators for genetic programming. In *PPSN XIII*, volume 8672 of *LNCS*, pages 454–464, Ljubljana, Slovenia. Springer.

Pawlak, T. P. (2015). *Competent Algorithms for Geometric Semantic Genetic Programming*. PhD thesis, Poznan University of Technology, Poznan, Poland.

Pawlak, T. P. (2016). Geometric semantic genetic programming is overkill. In *EuroGP'16*, Lecture Notes in Computer Science. Springer.

Pawlak, T. P. and Krawiec, K. (2016a). Progress properties and fitness bounds for geometric semantic search operators. *Genetic Programming and Evolvable Machines*, 17(1):5–23.

Pawlak, T. P. and Krawiec, K. (2016b). Semantic geometric initialization. In Heywood, M. I., McDermott, J., Castelli, M., Costa, E., and Sim, K., editors, *EuroGP 2016: Proceedings of the 19th European Conference on Genetic Programming*, volume 9594 of *LNCS*, pages 261–277, Porto, Portugal. Springer Verlag.

Pawlak, T. P. and Krawiec, K. (2016c). Semantic geometric initialization. In *EuroGP'16*, Lecture Notes in Computer Science. Springer.

Pawlak, T. P., Wieloch, B., and Krawiec, K. (2015a). Review and comparative analysis of geometric semantic crossovers. *Genetic Programming and Evolvable Machines*, 16(3):351–386.

Pawlak, T. P., Wieloch, B., and Krawiec, K. (2015b). Semantic backpropagation for designing search operators in genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(3):326–340.

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., and Galvan-Lopez, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119.

Uy, N. Q., Hoai, N. X., O'Neill, M., McKay, R. I., and Phong, D. N. (2013). On the roles of semantic locality of crossover in genetic programming. *Information Sciences*, 235:195–213.

Vanneschi, L., Castelli, M., Manzoni, L., and Silva, S. (2013). A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In Krawiec, K., Moraglio, A., Hu, T., Uyar, A. S., and Hu, B., editors, *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 205–216, Vienna, Austria. Springer Verlag.

Vanneschi, L., Castelli, M., and Silva, S. (2014). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214.

Wieloch, B. (2013). *Semantic Extensions for Genetic Programming*. PhD thesis, Poznan University of Technology.

Wieloch, B. and Krawiec, K. (2013). Running programs backwards: instruction inversion for effective search in semantic spaces. In *GECCO'13*, pages 1013–1020, Amsterdam, The Netherlands. ACM.

Zhu, Z., Nandi, A. K., and Aslam, M. W. (2013). Adapted geometric semantic genetic programming for diabetes and breast cancer classification. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2013)*.

Competent Geometric Semantic Genetic Programming for Symbolic Regression and Boolean Function Synthesis: Appendix C: Detailed experimental results

Tomasz P. Pawlak tpawlak@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poznań, Poland
Krzysztof Krawiec krawiec@cs.put.poznan.pl
Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Tables 17 – 30 present the detailed results of the experiments described in Sections 5.2 – 5.5 of [Pawlak and Krawiec, 2017].

References

[Pawlak and Krawiec, 2017] Pawlak, T. P. and Krawiec, K. (2017). Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. *Evolutionary Computation*. Early access.

Table 17: (a) The average and .95-confidence interval of the number of semantically unique programs in population. (b) Outranking graph of operators.

(a)

Problem	RHH		SDI		CI	
R1	979.00	±1.43	**1000.00**	±0.00	**1000.00**	±0.00
R2	979.00	±1.43	**1000.00**	±0.00	**1000.00**	±0.00
Kj3	976.07	±1.52	**1000.00**	±0.00	**1000.00**	±0.00
Kj4	979.73	±1.35	**1000.00**	±0.00	**1000.00**	±0.00
Kj11	976.30	±1.46	**1000.00**	±0.00	**1000.00**	±0.00
Ng9	977.50	±1.52	**1000.00**	±0.00	**1000.00**	±0.00
Ng12	977.50	±1.52	**1000.00**	±0.00	**1000.00**	±0.00
Pg1	978.20	±1.38	**1000.00**	±0.00	**1000.00**	±0.00
Vl1	979.23	±1.43	**1000.00**	±0.00	**1000.00**	±0.00
Par6	558.73	±5.08	**1000.00**	±0.00	**1000.00**	±0.00
Par7	622.57	±4.35	**1000.00**	±0.00	**1000.00**	±0.00
Par8	676.00	±3.79	**1000.00**	±0.00	**1000.00**	±0.00
Mux11	779.57	±4.55	**1000.00**	±0.00	**1000.00**	±0.00
Mux20	887.90	±4.08	**1000.00**	±0.00	**1000.00**	±0.00
Maj9	717.60	±5.38	**1000.00**	±0.00	**1000.00**	±0.00
Maj10	747.93	±4.22	**1000.00**	±0.00	**1000.00**	±0.00
Cmp8	676.00	±3.79	**1000.00**	±0.00	**1000.00**	±0.00
Cmp10	747.93	±4.22	**1000.00**	±0.00	**1000.00**	±0.00
Rank:	3.00		1.50		1.50	

(b)

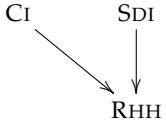


Table 18: (a) The average and .95-confidence interval of the number of programs produced until population's L_1-convex hull includes the target. (b) Post-hoc analysis of Friedman's test ($p = 2.03 \times 10^{-3}$) using the symmetry test: p-values of incorrectly judging an operator in a row as including the target in the convex hull using fewer programs than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

(a)

Problem	RHH		SDI		CI	
R1	72.03	±24.80	54.60	±19.64	**25.93**	±6.80
R2	**11.37**	±2.31	17.70	±3.94	12.63	±2.49
Kj3	**7.97**	±1.32	13.90	±3.31	12.23	±2.55
Kj4	**8.07**	±1.90	13.40	±3.19	11.60	±2.21
Kj11	61.80	±10.19	86.83	±19.87	**60.70**	±13.43
Ng9	9.03	±1.95	11.03	±5.11	**7.50**	±2.16
Ng12	**10.70**	±2.80	14.97	±4.44	11.33	±2.62
Pg1	16.17	±2.82	20.20	±4.28	**15.97**	±3.36
Vl1	**7.90**	±1.34	**7.90**	±1.44	8.90	±2.09
Par6	5.67	±0.77	5.63	±0.75	**4.87**	±0.52
Par7	6.33	±0.80	5.30	±0.56	**5.07**	±0.57
Par8	7.37	±0.74	5.77	±0.68	**5.43**	±0.58
Mux11	7.33	±0.95	7.43	±0.96	**6.40**	±0.63
Mux20	10.65	±1.58	7.50	±1.30	**7.10**	±1.13
Maj9	6.53	±0.75	5.97	±0.76	**5.37**	±0.60
Maj10	**5.97**	±0.65	7.10	±0.80	6.33	±0.61
Cmp8	7.20	±0.81	5.73	±0.69	**5.33**	±0.53
Cmp10	**6.30**	±0.96	7.17	±0.87	**6.30**	±0.65
Rank:	2.06		2.53		1.42	

(b)

	RHH	SDI	CI
RHH		0.322	
SDI			
CI	0.127	**0.002**	

Table 19: The probability that selection is effective, with .95-confidence interval. The values of 1.000 not in bold are factually smaller than one.

Problem	RTs		RSTs		RCTs		CTs		CSTs		CCTs	
R1	0.996	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	0.999	±0.000	1.000	±0.000
R2	0.995	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	0.999	±0.000	**1.000**	±0.000
Kj3	0.990	±0.001	0.998	±0.001	**1.000**	±0.000	0.996	±0.001	0.999	±0.000	1.000	±0.000
Kj4	0.992	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	**1.000**	±0.000
Kj11	0.992	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	1.000	±0.000
Ng9	0.995	±0.001	0.999	±0.000	**1.000**	±0.000	0.997	±0.001	0.999	±0.000	**1.000**	±0.000
Ng12	0.996	±0.001	1.000	±0.000	**1.000**	±0.000	0.996	±0.001	0.999	±0.000	**1.000**	±0.000
Pg1	0.996	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	1.000	±0.000
Vl1	0.993	±0.001	0.999	±0.000	**1.000**	±0.000	0.997	±0.001	1.000	±0.000	1.000	±0.000
Par6	0.972	±0.002	0.973	±0.002	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	**1.000**	±0.000
Par7	0.974	±0.002	0.974	±0.002	**1.000**	±0.000	0.996	±0.001	0.999	±0.000	**1.000**	±0.000
Par8	0.978	±0.002	0.980	±0.002	**1.000**	±0.000	0.997	±0.001	0.999	±0.000	**1.000**	±0.000
Mux11	0.819	±0.004	0.966	±0.002	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	**1.000**	±0.000
Mux20	0.926	±0.004	0.985	±0.002	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	**1.000**	±0.000
Maj9	0.992	±0.001	0.999	±0.000	**1.000**	±0.000	0.996	±0.001	1.000	±0.000	**1.000**	±0.000
Maj10	0.993	±0.001	0.999	±0.000	**1.000**	±0.000	0.997	±0.001	0.999	±0.000	**1.000**	±0.000
Cmp8	0.985	±0.001	0.998	±0.001	**1.000**	±0.000	0.995	±0.001	0.999	±0.000	**1.000**	±0.000
Cmp10	0.993	±0.001	0.999	±0.000	**1.000**	±0.000	0.997	±0.001	0.999	±0.000	**1.000**	±0.000
Rank:	5.944		4.167		1.361		4.778		3.111		1.639	

Table 20: Post-hoc analysis of Friedman's test ($p = 1.16 \times 10^{-12}$) using the symmetry test conducted on Table 19: p-values of incorrectly judging an operator in a row as more effective than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTs	RSTs	RCTs	CTs	CSTs	CCTs
RTs						
RSTs	**0.046**			0.921		
RCTs	**0.000**	**0.000**		**0.000**	0.052	0.998
CTs	0.408					
CSTs	**0.000**	0.525		0.075		
CCTs	**0.000**	**0.001**		**0.000**	0.161	

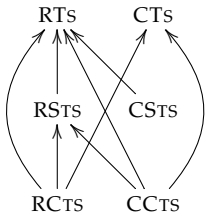


Table 21: Probability and .95-confidence interval that selection is t-geometric under L_1.

Problem	RTs		RSTs		RCTs		CTs		CSTs		CCTs	
R1	**0.001**	±0.000	0.001	±0.000	0.001	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000
R2	0.000	±0.000	0.000	±0.000	**0.001**	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000
Kj3	0.000	±0.000	0.000	±0.000	**0.001**	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000
Kj4	0.000	±0.000	0.000	±0.000	0.001	±0.000	**0.001**	±0.000	0.001	±0.000	0.001	±0.000
Kj11	0.000	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000	**0.000**	±0.000
Ng9	0.000	±0.000	0.000	±0.000	**0.001**	±0.000	0.000	±0.000	0.000	±0.000	0.001	±0.000
Ng12	0.002	±0.001	0.002	±0.000	**0.004**	±0.001	0.000	±0.000	0.000	±0.000	0.001	±0.000
Pg1	**0.001**	±0.000	0.001	±0.000	0.001	±0.000	0.000	±0.000	0.000	±0.000	0.001	±0.000
Vl1	0.001	±0.000	0.001	±0.000	0.002	±0.000	0.001	±0.000	0.001	±0.000	**0.002**	±0.001
Par6	0.036	±0.002	0.031	±0.002	**0.166**	±0.004	0.003	±0.001	0.003	±0.001	0.006	±0.001
Par7	0.028	±0.002	0.027	±0.002	**0.138**	±0.004	0.001	±0.000	0.001	±0.000	0.002	±0.000
Par8	0.022	±0.002	0.019	±0.002	**0.110**	±0.004	0.000	±0.000	0.000	±0.000	0.001	±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
Mux20	**0.000**	±0.000	**0.000**	±0.000	**0.000**	±0.000	**0.000**	±0.000	**0.000**	±0.000	**0.000**	±0.000
Maj9	0.000	±0.000	0.000	±0.000	**0.018**	±0.002	0.000	±0.000	0.000	±0.000	0.000	±0.000
Maj10	0.000	±0.000	0.001	±0.000	**0.001**	±0.000	0.000	±0.000	0.000	±0.000	0.000	±0.000
Cmp8	0.006	±0.001	0.007	±0.001	**0.022**	±0.002	0.000	±0.000	0.000	±0.000	0.002	±0.001
Cmp10	0.003	±0.001	0.003	±0.001	**0.012**	±0.001	0.000	±0.000	0.000	±0.000	0.001	±0.000
Rank:	3.000		3.556		1.667		4.861		4.722		3.194	

Table 22: Post-hoc analysis of Friedman's test ($p = 1.08 \times 10^{-6}$) using the symmetry test conducted on Table 21: p-values of incorrectly judging an operator in a row as more geometric than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTS	RSTS	RCTS	CTS	CSTS	CCTS
RTS		0.939		**0.023**	**0.045**	1.000
RSTS				0.244	0.370	
RCTS	0.223	**0.020**		**0.000**	**0.000**	0.108
CTS						
CSTS				1.000		
CCTS		0.991		0.059	0.108	

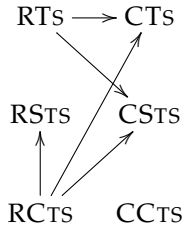


Table 23: Probability and .95-confidence interval that mutation is effective.

Problem	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
R1	0.986 ±0.006	**1.000** ±0.000	0.992 ±0.004	0.995 ±0.004	0.977 ±0.008	**1.000** ±0.000	0.984 ±0.006	**1.000** ±0.000
R2	0.985 ±0.006	**1.000** ±0.000	0.992 ±0.004	0.994 ±0.004	0.977 ±0.007	**1.000** ±0.000	0.984 ±0.006	0.999 ±0.001
Kj3	0.961 ±0.010	**1.000** ±0.000	0.992 ±0.004	0.973 ±0.008	0.982 ±0.007	**1.000** ±0.000	0.984 ±0.006	**1.000** ±0.000
Kj4	0.960 ±0.010	**1.000** ±0.000	0.992 ±0.005	0.969 ±0.009	0.978 ±0.007	**1.000** ±0.000	0.982 ±0.007	**1.000** ±0.000
Kj11	0.972 ±0.008	**1.000** ±0.000	0.980 ±0.007	0.988 ±0.005	0.985 ±0.006	**1.000** ±0.000	0.982 ±0.006	**1.000** ±0.000
Ng9	0.988 ±0.005	**1.000** ±0.000	0.980 ±0.007	0.997 ±0.003	0.988 ±0.005	**1.000** ±0.000	0.983 ±0.006	0.999 ±0.001
Ng12	0.977 ±0.007	**1.000** ±0.000	0.979 ±0.007	0.988 ±0.005	0.988 ±0.005	**1.000** ±0.000	0.983 ±0.006	0.999 ±0.001
Pg1	0.986 ±0.006	**1.000** ±0.000	0.979 ±0.007	0.998 ±0.002	0.986 ±0.006	**1.000** ±0.000	0.982 ±0.007	**1.000** ±0.000
Vl1	0.971 ±0.008	**1.000** ±0.000	0.979 ±0.007	0.982 ±0.007	0.982 ±0.007	**1.000** ±0.000	0.981 ±0.007	0.999 ±0.002
Par6	0.713 ±0.022	**0.996** ±0.003	0.754 ±0.021	0.753 ±0.021	0.377 ±0.024	0.992 ±0.005	0.871 ±0.017	0.493 ±0.025
Par7	0.756 ±0.021	**0.994** ±0.004	0.788 ±0.020	0.755 ±0.021	0.489 ±0.024	0.993 ±0.004	0.870 ±0.017	0.557 ±0.024
Par8	0.779 ±0.020	**0.999** ±0.002	0.801 ±0.020	0.784 ±0.020	0.477 ±0.024	0.997 ±0.003	0.894 ±0.015	0.569 ±0.025
Mux11	0.696 ±0.023	0.995 ±0.004	0.702 ±0.022	0.749 ±0.022	0.545 ±0.025	**0.999** ±0.002	0.884 ±0.016	0.598 ±0.025
Mux20	0.813 ±0.024	0.999 ±0.002	0.811 ±0.024	0.716 ±0.027	0.748 ±0.027	**0.999** ±0.002	0.940 ±0.015	0.686 ±0.028
Maj9	0.880 ±0.016	**1.000** ±0.000	0.900 ±0.015	0.942 ±0.011	0.591 ±0.024	0.999 ±0.001	0.921 ±0.013	0.653 ±0.023
Maj10	0.911 ±0.014	**1.000** ±0.000	0.902 ±0.015	0.946 ±0.011	0.616 ±0.023	0.995 ±0.003	0.926 ±0.013	0.690 ±0.023
Cmp8	0.853 ±0.017	**0.999** ±0.002	0.851 ±0.018	0.912 ±0.014	0.564 ±0.024	0.996 ±0.003	0.900 ±0.015	0.605 ±0.024
Cmp10	0.903 ±0.015	**1.000** ±0.000	0.894 ±0.015	0.939 ±0.012	0.622 ±0.023	0.999 ±0.002	0.928 ±0.013	0.677 ±0.023
Rank:	6.111	1.500	5.556	4.556	6.889	1.778	4.833	4.778

Table 24: Post-hoc analysis of Friedman's test ($p = 2.13 \times 10^{-10}$) using the symmetry test conducted on Table 23: p-values of incorrectly judging an operator in a row as more effective than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
RTM					0.987			
RSDM	**0.000**		**0.000**	**0.004**	**0.000**	1.000	**0.001**	**0.001**
RSGM	0.996				0.743			
RCM	0.512		0.922		0.084		1.000	1.000
CTM								
CSDM	**0.000**		**0.000**	**0.014**	**0.000**		**0.004**	**0.005**
CSGM	0.743		0.987		0.193			
CCM	0.700		0.980		0.166		1.000	

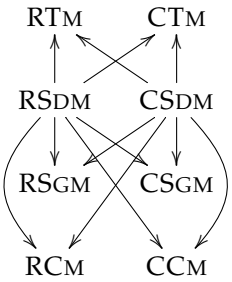


Table 25: (a) Median and .95-confidence interval of offspring-to-parent L_1 distance resulting from mutation operators. Values $> 10^4$ are rounded to an order of magnitude. (b) Post-hoc analysis of Friedman's test ($p = 2.27 \times 10^{-9}$) using the symmetry test: p-values of incorrectly judging an operator in a row as producing offspring less distant to the parent than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

(a)

Problem	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
R1	36.52 ≤ 37.81 ≤ 42.67	32.83 ≤ 35.90 ≤ 36.98	0.61 ≤ 0.79 ≤ 0.79	42.71 ≤ 43.02 ≤ 44.00	10.54 ≤ 10.91 ≤ 11.82	12.86 ≤ 13.69 ≤ 14.55	0.58 ≤ **0.65** ≤ 0.91	16.89 ≤ 17.71 ≤ 19.23
R2	22.01 ≤ 22.59 ≤ 23.96	19.14 ≤ 20.48 ≤ 21.79	0.52 ≤ 0.54 ≤ 0.56	11.15 ≤ 11.34 ≤ 11.57	7.77 ≤ 8.24 ≤ 8.59	8.31 ≤ 8.85 ≤ 9.59	0.53 ≤ **0.54** ≤ 0.59	10.12 ≤ 10.48 ≤ 11.00
Kj3	38.94 ≤ 40.25 ≤ 40.30	30.82 ≤ 32.74 ≤ 37.11	3.15 ≤ 8.77 ≤ 13.27	7.14 ≤ **7.16** ≤ 7.35	9.60 ≤ 10.04 ≤ 10.54	10.53 ≤ 10.88 ≤ 11.20	30.02 ≤ 45.50 ≤ 2890.66	10.08 ≤ 10.45 ≤ 10.66
Kj4	206.67 ≤ 296.62 ≤ 698.39	142.34 ≤ 274.55 ≤ 595.11	5923.39 ≤ 10^6 ≤ 10^6	5.92 ≤ **5.99** ≤ 6.34	8.66 ≤ 9.01 ≤ 9.53	9.45 ≤ 9.66 ≤ 10.31	10^5 ≤ 10^5 ≤ 10^5	9.14 ≤ 9.35 ≤ 9.83
Kj11	248.23 ≤ 265.90 ≤ 279.09	220.19 ≤ 235.97 ≤ 247.04	9.25 ≤ **10.41** ≤ 44.64	260.45 ≤ 276.04 ≤ 324.15	54.02 ≤ 57.12 ≤ 58.93	59.80 ≤ 62.77 ≤ 65.39	5.96 ≤ 152.34 ≤ 10^6	67.55 ≤ 70.26 ≤ 76.31
Ng9	100.23 ≤ 113.29 ≤ 135.20	102.84 ≤ 115.77 ≤ 130.72	3.43 ≤ 5.78 ≤ 6.39	38.48 ≤ 39.19 ≤ 39.44	36.57 ≤ 38.82 ≤ 41.55	39.40 ≤ 41.64 ≤ 44.20	3.32 ≤ **4.94** ≤ 5.49	42.98 ≤ 44.58 ≤ 45.79
Ng12	131.72 ≤ 136.36 ≤ 158.66	168.39 ≤ 174.13 ≤ 212.81	3.09 ≤ 5.54 ≤ 6.97	34.32 ≤ 34.44 ≤ 34.75	45.48 ≤ 46.77 ≤ 48.65	49.00 ≤ 50.85 ≤ 52.84	4.01 ≤ **4.18** ≤ 7.18	50.90 ≤ 51.82 ≤ 53.84
Pg1	461.73 ≤ 543.77 ≤ 686.83	425.81 ≤ 446.12 ≤ 488.63	10^8 ≤ 10^{13} ≤ 10^{36}	83.78 ≤ 85.23 ≤ 87.54	47.11 ≤ **49.64** ≤ 52.07	54.33 ≤ 57.53 ≤ 58.89	10^{14} ≤ 10^{37} ≤ 10^{56}	57.77 ≤ 59.15 ≤ 61.68
Vl1	616.87 ≤ 832.11 ≤ 1149.98	776.37 ≤ 816.35 ≤ 1190.39	284.17 ≤ 522.35 ≤ 10^8	34.46 ≤ **34.87** ≤ 36.51	47.55 ≤ 48.89 ≤ 50.52	54.24 ≤ 57.35 ≤ 61.49	10^{13} ≤ 10^{67} ≤ 10^{125}	50.84 ≤ 51.41 ≤ 52.19
Par6	9.25 ≤ 10.50 ≤ 11.11	19.11 ≤ 19.45 ≤ 19.85	15.96 ≤ 16.00 ≤ 16.22	11.40 ≤ 12.00 ≤ 12.50	0.40 ≤ 0.50 ≤ 0.57	7.80 ≤ 8.67 ≤ 9.00	15.85 ≤ 16.26 ≤ 16.62	0.00 ≤ **0.29** ≤ 1.00
Par7	24.57 ≤ 26.18 ≤ 28.00	39.00 ≤ 39.71 ≤ 41.83	31.32 ≤ 32.19 ≤ 32.33	24.00 ≤ 25.71 ≤ 28.89	1.43 ≤ **1.78** ≤ 2.00	14.00 ≤ 15.27 ≤ 17.33	31.20 ≤ 32.08 ≤ 33.24	1.50 ≤ 2.00 ≤ 2.75
Par8	45.71 ≤ 50.62 ≤ 53.65	75.83 ≤ 78.00 ≤ 79.05	61.92 ≤ 64.11 ≤ 64.11	55.56 ≤ 57.85 ≤ 64.00	3.00 ≤ **3.00** ≤ 3.50	25.14 ≤ 28.00 ≤ 28.86	62.17 ≤ 63.00 ≤ 64.79	2.00 ≤ 4.00 ≤ 5.00
Mux11	134.67 ≤ 152.00 ≤ 192.00	513.93 ≤ 524.36 ≤ 569.25	194.55 ≤ 232.20 ≤ 235.00	156.00 ≤ 161.14 ≤ 180.67	5.50 ≤ **8.00** ≤ 12.00	121.20 ≤ 128.00 ≤ 137.00	161.88 ≤ 171.77 ≤ 184.44	16.00 ≤ 20.92 ≤ 30.00
Mux20	10^5 ≤ 10^5 ≤ 10^5	10^5 ≤ 10^5 ≤ 10^5	10^5 ≤ 10^6 ≤ 10^6	10^5 ≤ 10^5 ≤ 10^5	9772.00 ≤ **10^4** ≤ 10^4	10^5 ≤ 10^5 ≤ 10^5	10^5 ≤ 10^5 ≤ 10^6	10^4 ≤ 10^4 ≤ 10^4
Maj9	149.33 ≤ 160.00 ≤ 170.67	153.60 ≤ 159.80 ≤ 165.50	133.95 ≤ 135.00 ≤ 135.68	115.14 ≤ 116.76 ≤ 120.33	6.40 ≤ **8.00** ≤ 12.00	68.00 ≤ 75.71 ≤ 78.29	135.27 ≤ 135.34 ≤ 135.87	17.00 ≤ 18.80 ≤ 21.60
Maj10	325.82 ≤ 339.20 ≤ 356.27	320.00 ≤ 330.00 ≤ 343.20	262.65 ≤ 263.02 ≤ 264.06	244.59 ≤ 257.45 ≤ 267.00	15.00 ≤ **19.00** ≤ 25.00	109.00 ≤ 117.33 ≤ 139.20	265.50 ≤ 268.62 ≤ 271.00	34.44 ≤ 42.00 ≤ 49.50
Cmp8	71.43 ≤ 76.00 ≤ 77.33	80.50 ≤ 83.47 ≤ 84.00	69.38 ≤ 70.45 ≤ 70.68	52.78 ≤ 53.33 ≤ 54.82	3.00 ≤ **4.00** ≤ 5.00	32.60 ≤ 36.75 ≤ 40.67	68.49 ≤ 69.19 ≤ 69.47	5.00 ≤ 7.00 ≤ 9.80
Cmp10	298.67 ≤ 306.00 ≤ 316.00	302.10 ≤ 306.67 ≤ 314.33	268.00 ≤ 268.97 ≤ 269.27	215.58 ≤ 217.88 ≤ 231.47	16.00 ≤ **22.50** ≤ 26.83	125.20 ≤ 140.00 ≤ 158.67	262.47 ≤ 268.71 ≤ 268.90	33.67 ≤ 40.00 ≤ 48.40
Rank:	6.39	7.00	4.89	4.17	1.83	3.50	5.11	3.11

(b)

	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
RTM		0.995						
RSDM								
RSGM	0.595	0.161					1.000	
RCM	0.116	**0.012**	0.987				0.944	
CTM	**0.000**	**0.000**	**0.004**	0.081		0.454	**0.001**	0.771
CSDM	**0.009**	**0.000**	0.687	0.992			0.500	
CSGM	0.771	0.286						
CCM	**0.001**	**0.000**	0.365	0.902		1.000	0.217	

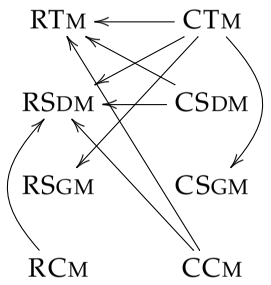


Table 26: The average and .95-confidence interval of offspring-to-parent size ratio resulting from mutation.

Problem	RTM	RSDM	RSGM	RCM	CTM	CSDM	CSGM	CCM
R1	1.95 ±0.09	1.98 ±0.09	4.21 ±0.16	3.00 ±0.12	1.08 ±0.03	**1.01** ±0.04	1.60 ±0.05	1.12 ±0.04
R2	2.54 ±0.14	2.59 ±0.14	5.19 ±0.21	3.18 ±0.12	1.06 ±0.03	**1.00** ±0.04	1.59 ±0.05	1.10 ±0.03
Kj3	2.77 ±0.16	2.82 ±0.16	5.82 ±0.24	1.47 ±0.07	1.07 ±0.03	0.98 ±0.03	1.51 ±0.04	**0.99** ±0.03
Kj4	2.72 ±0.17	2.80 ±0.17	5.71 ±0.25	1.42 ±0.07	**1.04** ±0.03	0.95 ±0.03	1.53 ±0.05	0.95 ±0.03
Kj11	2.17 ±0.13	2.15 ±0.13	4.97 ±0.22	4.38 ±0.21	**1.02** ±0.03	0.97 ±0.03	1.64 ±0.05	1.22 ±0.04
Ng9	2.07 ±0.11	2.15 ±0.12	4.80 ±0.20	2.86 ±0.09	1.04 ±0.03	**0.98** ±0.04	1.73 ±0.05	1.17 ±0.04
Ng12	2.39 ±0.14	2.55 ±0.15	5.43 ±0.23	4.21 ±0.18	**1.01** ±0.03	0.95 ±0.03	1.63 ±0.05	1.17 ±0.04
Pg1	2.09 ±0.12	2.07 ±0.12	4.82 ±0.19	2.53 ±0.12	**1.03** ±0.03	0.95 ±0.04	1.73 ±0.06	1.06 ±0.03
Vl1	2.42 ±0.14	2.47 ±0.14	5.50 ±0.23	1.58 ±0.08	**1.03** ±0.03	0.96 ±0.04	1.68 ±0.05	1.05 ±0.04
Par6	0.87 ±0.03	0.73 ±0.03	1.37 ±0.04	0.82 ±0.02	0.94 ±0.01	0.70 ±0.02	**1.03** ±0.00	0.87 ±0.02
Par7	0.86 ±0.03	0.74 ±0.03	1.36 ±0.03	0.74 ±0.02	0.93 ±0.01	0.70 ±0.02	**1.04** ±0.01	0.85 ±0.02
Par8	0.84 ±0.03	0.74 ±0.03	1.37 ±0.03	0.70 ±0.02	0.92 ±0.01	0.70 ±0.02	**1.05** ±0.01	0.87 ±0.02
Mux11	0.82 ±0.02	0.64 ±0.03	1.23 ±0.03	1.13 ±0.04	0.89 ±0.01	0.70 ±0.02	**1.06** ±0.01	0.88 ±0.02
Mux20	0.76 ±0.02	0.62 ±0.02	1.22 ±0.03	1.90 ±0.22	0.85 ±0.01	0.74 ±0.02	**1.08** ±0.01	2.28 ±0.37
Maj9	0.85 ±0.04	0.88 ±0.04	1.63 ±0.05	1.80 ±0.06	0.90 ±0.01	0.73 ±0.02	1.10 ±0.01	**0.93** ±0.02
Maj10	0.82 ±0.04	0.83 ±0.04	1.63 ±0.04	1.86 ±0.07	0.87 ±0.01	0.74 ±0.02	1.10 ±0.01	**0.96** ±0.02
Cmp8	0.84 ±0.04	0.83 ±0.04	1.51 ±0.04	1.37 ±0.04	0.89 ±0.01	0.70 ±0.02	**1.08** ±0.01	0.89 ±0.02
Cmp10	0.77 ±0.03	0.81 ±0.03	1.54 ±0.04	1.43 ±0.04	0.88 ±0.01	0.72 ±0.02	1.09 ±0.01	**0.92** ±0.02

Table 27: Probability and .95-confidence interval that crossover is effective.

Problem	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
R1	0.841 ±0.004	0.995 ±0.001	0.999 ±0.000	0.990 ±0.001	0.917 ±0.004	0.999 ±0.000	**1.000** ±0.000	1.000 ±0.000
R2	0.841 ±0.004	0.995 ±0.001	0.999 ±0.000	0.990 ±0.001	0.917 ±0.004	0.999 ±0.000	**1.000** ±0.000	1.000 ±0.000
Kj3	0.840 ±0.004	0.995 ±0.001	0.999 ±0.000	0.990 ±0.001	0.919 ±0.004	1.000 ±0.000	**1.000** ±0.000	**1.000** ±0.000
Kj4	0.838 ±0.004	0.995 ±0.001	0.999 ±0.000	0.990 ±0.001	0.928 ±0.004	1.000 ±0.000	**1.000** ±0.000	1.000 ±0.000
Kj11	0.848 ±0.004	0.994 ±0.001	1.000 ±0.000	0.992 ±0.001	0.940 ±0.003	1.000 ±0.000	1.000 ±0.000	**1.000** ±0.000
Ng9	0.848 ±0.004	0.994 ±0.001	1.000 ±0.000	0.992 ±0.001	0.944 ±0.003	1.000 ±0.000	**1.000** ±0.000	1.000 ±0.000
Ng12	0.848 ±0.004	0.994 ±0.001	1.000 ±0.000	0.992 ±0.001	0.944 ±0.003	1.000 ±0.000	**1.000** ±0.000	1.000 ±0.000
Pg1	0.847 ±0.004	0.994 ±0.001	1.000 ±0.000	0.992 ±0.001	0.943 ±0.003	**1.000** ±0.000	1.000 ±0.000	1.000 ±0.000
Vl1	0.847 ±0.004	0.994 ±0.001	0.999 ±0.000	0.992 ±0.001	0.944 ±0.003	1.000 ±0.000	**1.000** ±0.000	1.000 ±0.000
Par6	0.624 ±0.006	**0.925** ±0.003	0.747 ±0.005	0.706 ±0.005	0.422 ±0.006	0.863 ±0.004	0.823 ±0.004	0.468 ±0.006
Par7	0.655 ±0.006	**0.941** ±0.003	0.786 ±0.005	0.737 ±0.005	0.472 ±0.006	0.903 ±0.003	0.855 ±0.004	0.524 ±0.006
Par8	0.678 ±0.005	**0.954** ±0.002	0.821 ±0.004	0.776 ±0.005	0.495 ±0.006	0.910 ±0.003	0.873 ±0.004	0.537 ±0.006
Mux11	0.724 ±0.005	**0.968** ±0.002	0.880 ±0.004	0.827 ±0.004	0.608 ±0.006	0.958 ±0.002	0.899 ±0.004	0.629 ±0.006
Mux20	0.779 ±0.006	0.984 ±0.002	0.937 ±0.003	0.884 ±0.005	0.772 ±0.006	**0.992** ±0.001	0.949 ±0.003	0.758 ±0.006
Maj9	0.698 ±0.005	**0.961** ±0.002	0.845 ±0.004	0.792 ±0.005	0.539 ±0.006	0.935 ±0.003	0.886 ±0.004	0.578 ±0.006
Maj10	0.707 ±0.005	**0.966** ±0.002	0.863 ±0.004	0.809 ±0.005	0.565 ±0.006	0.943 ±0.003	0.896 ±0.004	0.600 ±0.006
Cmp8	0.678 ±0.005	**0.954** ±0.002	0.821 ±0.004	0.776 ±0.005	0.495 ±0.006	0.910 ±0.003	0.873 ±0.004	0.537 ±0.006
Cmp10	0.707 ±0.005	**0.966** ±0.002	0.863 ±0.004	0.809 ±0.005	0.565 ±0.006	0.943 ±0.003	0.896 ±0.004	0.600 ±0.006
Rank:	7.000	3.056	4.000	5.500	7.444	2.167	2.250	4.583

Table 28: Post-hoc analysis of Friedman's test ($p = 6.50 \times 10^{-10}$) using the symmetry test conducted on Table 27: p-values of incorrectly judging an operator in a row as more effective than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
RTX					0.999			
RSDX	**0.000**		0.944	0.055	**0.000**			0.546
RSGX	**0.005**			0.593	**0.001**			0.995
RCX	0.593				0.249			
CTX								
CSDX	**0.000**	0.944	0.284	**0.001**	**0.000**		1.000	**0.045**
CSGX	**0.000**	0.981	0.407	**0.002**	**0.000**			0.080
CCX	0.067			0.959	**0.012**			

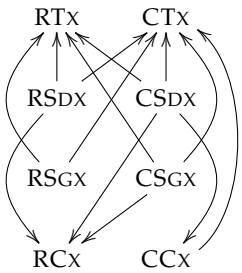


Table 29: Probability and .95-confidence interval that crossover is L_1-geometric and effective.

Problem	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
R1	0.089 ±0.003	0.111 ±0.004	0.999 ±0.000	0.407 ±0.006	0.054 ±0.003	0.058 ±0.003	**0.999** ±0.000	0.088 ±0.004
R2	0.089 ±0.003	0.111 ±0.004	0.999 ±0.000	0.407 ±0.006	0.054 ±0.003	0.058 ±0.003	**0.999** ±0.000	0.088 ±0.004
Kj3	0.061 ±0.003	0.080 ±0.003	0.999 ±0.000	0.282 ±0.005	0.055 ±0.003	0.064 ±0.003	**0.999** ±0.000	0.073 ±0.004
Kj4	0.084 ±0.003	0.113 ±0.004	0.999 ±0.000	0.316 ±0.006	0.068 ±0.004	0.076 ±0.004	**0.999** ±0.001	0.094 ±0.005
Kj11	0.038 ±0.002	0.049 ±0.003	**0.999** ±0.000	0.187 ±0.005	0.056 ±0.003	0.061 ±0.003	0.999 ±0.000	0.063 ±0.003
Ng9	0.104 ±0.004	0.126 ±0.004	**0.999** ±0.000	0.375 ±0.006	0.059 ±0.003	0.061 ±0.003	0.999 ±0.000	0.078 ±0.004
Ng12	0.104 ±0.004	0.126 ±0.004	**0.999** ±0.000	0.375 ±0.006	0.059 ±0.003	0.061 ±0.003	0.999 ±0.000	0.078 ±0.004
Pg1	0.050 ±0.003	0.062 ±0.003	0.999 ±0.000	0.207 ±0.005	0.061 ±0.003	0.065 ±0.003	**0.999** ±0.000	0.072 ±0.004
Vl1	0.058 ±0.003	0.075 ±0.003	**0.999** ±0.000	0.208 ±0.005	0.061 ±0.003	0.059 ±0.003	0.999 ±0.000	0.065 ±0.004
Par6	0.153 ±0.004	0.228 ±0.005	0.747 ±0.005	0.538 ±0.006	0.112 ±0.004	0.223 ±0.005	**0.823** ±0.004	0.315 ±0.005
Par7	0.146 ±0.004	0.229 ±0.005	0.786 ±0.005	0.520 ±0.006	0.121 ±0.004	0.218 ±0.005	**0.855** ±0.004	0.337 ±0.005
Par8	0.145 ±0.004	0.220 ±0.005	0.821 ±0.004	0.507 ±0.006	0.120 ±0.004	0.216 ±0.005	**0.873** ±0.004	0.332 ±0.005
Mux11	0.141 ±0.004	0.202 ±0.005	0.880 ±0.004	0.432 ±0.006	0.111 ±0.004	0.192 ±0.005	**0.899** ±0.004	0.327 ±0.005
Mux20	0.133 ±0.005	0.177 ±0.005	0.937 ±0.003	0.238 ±0.006	0.102 ±0.004	0.153 ±0.005	**0.949** ±0.003	0.280 ±0.007
Maj9	0.144 ±0.004	0.212 ±0.005	0.845 ±0.004	0.483 ±0.006	0.116 ±0.004	0.207 ±0.005	**0.886** ±0.004	0.333 ±0.005
Maj10	0.139 ±0.004	0.211 ±0.005	0.863 ±0.004	0.462 ±0.006	0.116 ±0.004	0.204 ±0.005	**0.896** ±0.004	0.336 ±0.005
Cmp8	0.145 ±0.004	0.220 ±0.005	0.821 ±0.004	0.507 ±0.006	0.120 ±0.004	0.216 ±0.005	**0.873** ±0.004	0.332 ±0.005
Cmp10	0.139 ±0.004	0.211 ±0.005	0.863 ±0.004	0.462 ±0.006	0.116 ±0.004	0.204 ±0.005	**0.896** ±0.004	0.336 ±0.005
Rank:	6.667	4.778	1.778	3.056	7.722	6.222	1.222	4.556

Table 30: Post-hoc analysis of Friedman's test ($p = 7.22 \times 10^{-15}$) using the symmetry test conducted on Table 29: p-values of incorrectly judging an operator in a row as more geometric than the one in a column. The p-values $\leq .05$ are represented by arcs in the graph.

	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
RTX					0.902			
RSDX	0.286				**0.008**	0.641		
RSGX	**0.000**	**0.006**		0.771	**0.000**	**0.000**		**0.015**
RCX	**0.000**	0.408			**0.000**	**0.003**		0.595
CTX								
CSDX	0.999				0.595			
CSGX	**0.000**	**0.000**	0.998	0.324	**0.000**	**0.000**		**0.001**
CCX	0.160	1.000			**0.003**	0.454		

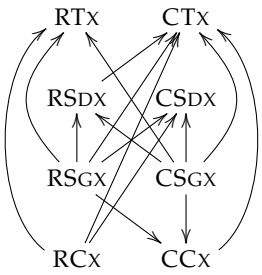


Table 31: The average and .95-confidence interval of offspring-to-parent size ratio resulting from crossover.

Problem	RTX	RSDX	RSGX	RCX	CTX	CSDX	CSGX	CCX
R1	**1.00** ±0.01	1.00 ±0.01	2.36 ±0.00	1.67 ±0.01	0.97 ±0.01	1.00 ±0.01	2.04 ±0.00	0.93 ±0.01
R2	**1.00** ±0.01	1.00 ±0.01	2.36 ±0.00	1.67 ±0.01	0.97 ±0.01	1.00 ±0.01	2.04 ±0.00	0.93 ±0.01
Kj3	**1.00** ±0.01	1.00 ±0.01	2.36 ±0.00	1.71 ±0.01	0.97 ±0.01	1.00 ±0.01	2.04 ±0.00	0.93 ±0.01
Kj4	**1.00** ±0.01	1.00 ±0.01	2.36 ±0.00	1.72 ±0.01	0.98 ±0.01	1.00 ±0.01	2.04 ±0.00	0.94 ±0.01
Kj11	**1.00** ±0.01	1.00 ±0.01	2.34 ±0.00	1.92 ±0.01	0.98 ±0.01	1.00 ±0.01	2.05 ±0.00	0.98 ±0.01
Ng9	**1.00** ±0.01	1.00 ±0.01	2.34 ±0.00	1.84 ±0.01	0.98 ±0.01	1.00 ±0.01	2.05 ±0.00	0.99 ±0.01
Ng12	**1.00** ±0.01	1.00 ±0.01	2.34 ±0.00	1.84 ±0.01	0.98 ±0.01	1.00 ±0.01	2.05 ±0.00	0.99 ±0.01
Pg1	**1.00** ±0.01	1.00 ±0.01	2.34 ±0.00	1.92 ±0.01	0.98 ±0.01	1.00 ±0.01	2.05 ±0.00	0.98 ±0.01
Vl1	1.00 ±0.01	1.00 ±0.01	2.34 ±0.00	1.95 ±0.01	0.98 ±0.01	**1.00** ±0.01	2.05 ±0.00	0.99 ±0.01
Par6	**1.00** ±0.01	0.99 ±0.01	2.18 ±0.00	0.97 ±0.01	0.98 ±0.01	0.98 ±0.01	2.02 ±0.00	0.87 ±0.01
Par7	**1.00** ±0.01	0.99 ±0.01	2.19 ±0.00	0.99 ±0.01	0.99 ±0.01	0.99 ±0.01	2.02 ±0.00	0.87 ±0.01
Par8	**1.00** ±0.01	1.00 ±0.01	2.19 ±0.00	1.01 ±0.01	0.98 ±0.01	0.99 ±0.01	2.02 ±0.00	0.87 ±0.01
Mux11	**1.00** ±0.01	1.00 ±0.01	2.20 ±0.00	1.06 ±0.01	0.99 ±0.01	0.99 ±0.01	2.03 ±0.00	0.87 ±0.01
Mux20	**1.00** ±0.01	1.00 ±0.01	2.20 ±0.00	1.16 ±0.02	0.99 ±0.01	1.00 ±0.01	2.04 ±0.00	1.28 ±0.02
Maj9	**1.00** ±0.01	1.00 ±0.01	2.19 ±0.00	1.03 ±0.01	0.99 ±0.01	0.99 ±0.01	2.03 ±0.00	0.87 ±0.01
Maj10	**1.00** ±0.01	1.00 ±0.01	2.20 ±0.00	1.04 ±0.01	0.99 ±0.01	0.99 ±0.01	2.03 ±0.00	0.87 ±0.01
Cmp8	**1.00** ±0.01	1.00 ±0.01	2.19 ±0.00	1.01 ±0.01	0.98 ±0.01	0.99 ±0.01	2.02 ±0.00	0.87 ±0.01
Cmp10	**1.00** ±0.01	1.00 ±0.01	2.20 ±0.00	1.04 ±0.01	0.99 ±0.01	0.99 ±0.01	2.03 ±0.00	0.87 ±0.01