Quantitative Analysis of Locally Geometric Semantic Crossover

Krzysztof Krawiec and Tomasz Pawlak

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

Abstract. We investigate the properties of locally geometric semantic crossover (LGX), a genetic programming search operator that is approximately semantically geometric on the level of homologous code fragments. For a pair of corresponding loci in the parents, LGX finds a semantically intermediate procedure from a library prepared prior to evolutionary run, and creates an offspring by using such procedure as replacement code. LGX proves superior when compared to standard subtree crossover and other control methods in terms of search convergence, test-set performance, and time required to find a high-quality solution. This paper focuses in particular the impact of homology and program semantic on LGX performance.

Keywords: genetic programming, semantic crossover, homology.

1 Introduction

In a broad sense, a program is a sequence of symbols (instructions), where each symbol has a particular, given a priori, *semantics*. The semantics of an instruction determines its *effect*: how its output should be determined (computed) from a given input. A human programmer familiar with programming language knows that semantics and can use it to anticipate combined behavior of two concatenated instructions or substituting one instruction with another. However, the algorithms considered in genetic programming (GP) have no access to such information. From their perspective, a program is a purely symbolic structure, where opcodes associated with particular instructions have no particular *meaning*.

On one hand, this is consistent with the evolutionary aspect of GP: in the end, natural evolution does not 'know' the phenotypic expression of genes. On the other hand, the knowledge of semantics is definitely one of factors that makes human programming so effective. Therefore, equipping GP algorithms with some semantic extensions can lead to substantial progress in automated programming, and this opportunity attracted notable interest in recent GP research [4,8,11].

In [3] and [5] we proposed methods that make GP alert to certain semantic aspects of programs. The locally geometric semantic crossover (LGX, [5]) finds a semantic approximation of an intermediate ('medial') procedure for a pair of procedures (subtrees) located in parent programs and uses it as a replacement for the parent programs. In this follow-up study, we investigate the properties of this method, in particular the impact of homology on its performance.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 397–406, 2012.

2 The Method

The proposed approach exploits the *compositional* character of programs, by which we mean that not only complete programs, but also parts of programs and program conglomerates have valid interpretation in many programming languages. We also assume that the fitness function captures the divergence between program output and some known desired output. This is consistent with GP standards, where individuals are usually tested on a set of fitness cases, and fitness is some form of error built upon the outcome of these tests. Formally, a *metric* $||\cdot||$ calculating such error is given. The consequence of this assumption is a *convex surface* of fitness landscape, spanned over the space of vectors holding program outputs. Convexity allows designing recombination operators that are likely to yield offspring of good quality [10,3]. This is easy to demonstrate for Euclidean metric: given a point x corresponding to the desired output of a program and a pair of points x_1, x_2 representing parent solutions, any point on the segment between x_1 and x_2 cannot be further from x than $\max(||x_1 - x||, ||x_2 - x||)$.

The proposed method exploits this property *locally*, i.e., on level of program fragments, rather than entire programs. It operates in two major phases. Prior to evolutionary run, it creates a library of short programs, calculates their semantics and builds upon them an index for fast access. Then, during an evolution, the library is used by the crossover operator to modify the fragments of parents' programs in a semantically-aware way.

Building the Library of Procedures. The input to the method is a set of instructions I, each of them being an operator of arbitrary arity. The first step consists in creating from I a library L of short programs, called *procedures* in following discussion. The library is purposed to provide semantically diverse code fragments for the crossover operator. The choice of procedures in L can be done along different criteria, but here, for simplicity, L contains all trees of height at most h.

Next, the *semantics* $s(p)$ of every procedure $p \in L$ is calculated. Throughout this paper, by semantics we mean a vector of d outcomes produced by a program for all d inputs (fitness cases). Any two procedures p_1, p_2 that have the same semantics ($||s(p_1), s(p_2)|| = 0$) do the same thing, which is redundant from the viewpoint of the method. Therefore, we discard from L procedures that duplicate the semantics of other procedures, leaving only the shortest ones.

Indexing the Library. The semantics $s(p)$ of a procedure p is a point in a d-dimensional space. Distances between such points reflect the semantic differences between procedures. Semantically similar procedures are located close to each other, while the very different ones occupy distant positions.

To efficiently search this space for procedures that are as close as possible to an arbitrarily selected point, we employ *spatial index*, a data structure designed for geographic databases. As in this study we limit our interest to symbolic regression, the space of consideration is Euclidean and the semantic distance $||\cdot||$ becomes a norm, which allows us to employ the *R-trees* [1].

Locally Geometric Semantic Crossover. After the library is built and equipped with an R-tree index, a GP run is launched. It proceeds as regular GP,

except for employing a homologous crossover operator, termed *locally geometric crossover* (LGX). Given two parent programs p_1, p_2, LGX first identifies the *structurally common region* for them, which is defined as in one-point crossover by Poli and Langdon [13], i.e., a set of node locations (loci) that occur in both parents. The common region can be considered as an intersection of the parents, where the opcodes are ignored – only the tree structure matters (taking the opcodes into account would often render the common region almost empty). The subtree can embrace at most all locations in both parents, but typically it is smaller.

Next, LGX selects a random location (locus) in the common subtree, intended to serve as crossover point. This choice follows the same rules as in the canonic Koza-style GP [2]: an internal node is selected with probability 0.9 and a leaf with probability of only 0.1, to reduce bloat. Subsequently, LGX identifies the subtrees p'_1 and p'_2 rooted in the selected location in p_1 and p_2, respectively. As they are independent executable programs, their semantics $s(p'_1)$ and $s(p'_2)$, are known (technically: cached during individuals' evaluation), which allows us to determine the midpoint between them in the semantic space:

$$s_m = \frac{s(p'_1) + s(p'_2)}{2} \tag{1}$$

This point represents the semantics of a hypothetical procedure $p : s_m = s(p)$, which, when inserted into parents at the appointed location, would make the resulting offspring programs semantically intermediate at the point of crossover (cf. [3]). However, finding p in general requires solving an inverse problem $p = s^{-1}(s_m)$, which is a separate program induction problem in itself. Also, as s_m is a combination of semantics of two, potentially big trees, it may not be represented by a program available within the assumed program space.

This is where the library comes at help. Instead of looking for a procedure whose semantics is exactly s_m, we find in L the procedure that is semantically *most similar* to s_m, i.e.:

$$p = \arg\min_{p' \in L} ||s(p') - s_m|| \tag{2}$$

Finding p is facilitated using the R-tree index. The procedure p replaces then the subtrees p'_1 and p'_2 in the parent solutions, which so become the two offspring. This step concludes the crossover act.

Properties of the Approach. An important property of the proposed approach is *completeness*. As the library contains representatives of *all* semantic equivalence classes obtainable from given set of procedures, LGX can produce any tree. The semantic search space is not constrained.

The computational overhead compared to the standard GP approach is the sum of the time required to prepare the library (generation of procedures, calculation of semantics, elimination of semantic duplicates, and construction of an R-tree) and the time of querying the R-tree in LGX. According to [12], the worst-case complexity of the latter component is linear w.r.t. the number of objects (here: library size $|L|$), but usually the query time is significantly lower. This

cost depends also on the number of fitness cases and the number of procedures to be stored in the library, which in our case is a function of h. For large h, it can be substantial, thus, to keep the computational cost at bay, we use $h \in \{3, 4\}$.

Related Research. The past contributions that have something common with the approach presented here can be grouped according to two features: the use of a library and the semantically-aware modification of solutions. Concerning the former, LGX can be likened to run transferable libraries [14], which are repositories of program fragments intended to be used across multiple GP runs applied to different problem instances. However, [14] does not involve semantics. Concerning the latter, McPhee *et al.* were probably the first to study the impact of crossover on program semantics and so-called semantic building blocks [8]. In [9], Moraglio *et al.* considered properties of semantic spaces for different metrics and provided guidelines for designing semantically geometric crossovers. The semantically-aware crossover by Quang *et al.* [11] swaps a pair of subtrees in parent solutions that have similar, yet not too similar, semantics.

In the context of these contributions, LGX remains unique in combining three elements: the choice of program fragments w.r.t. their semantic properties, homologous character of crossover, and the use of a library of procedures.

3 The Experiment

The experiment is aimed at verification whether the semantic properties of LGX influence the efficiency of GP search. The experimental framework is symbolic regression, with instructions $\{+, -, \times, /\}$ and a terminal representing independent variable x. Semantics is defined as a vector of values returned by a program for 20 fitness cases distributed equidistantly in the interval $[-1, 1]$.

We consider two libraries, for the maximum procedure height $h \in \{3, 4\}$. For $h = 3$, there are 81 procedures, but only 38 of them are semantically distinct, so $|L| = 38$. For $h = 4$, these figures amount to 21385 and 1697, respectively.

We examine LGX with two types of control setups. The first of them is standard Koza-style GP [2], which involves conventional tree-swapping crossover that uses the same probability distribution as LGX for node selection (0.1 for leafs and 0.9 for internal nodes). Like other considered operators, it never replaces the root node. The latter, called RX (random crossover), is intended to verify if the observed results are due to the geometric character of crossing over performed by LGX. To certain extent, RX operates as LGX (Section 2), however its choice of procedure from L is purely random. Thus, RX is similar to LGX in terms of mode of operation, but it is completely blind to the structure of semantic space.

To sum up, there are 5 setups in total: canonical GP, RX and LGX for $h \in \{3, 4\}$, further referred as GP, RX_3, RX_4, LGX_3 and LGX_4.

We solve 6 univariate symbolic regression problems shown in Table 1: 3 polynomials and 3 rational functions taken from [6]. For each configuration, 150 runs are carried out, each starting from different initial population of size 1024 and lasting for 250 generations. Fitness is minimized and defined as the absolute error of the output produced byof an individual w.r.t. the desired output, summed for

Table 1. Test problems

Problem	*Definition (formula)*	*Problem*	*Definition (formula)*
Sextic	$x^6 - 2x^4 + x^2$	*R1*	$(x+1)^3/(x^2 - x + 1)$
Septic	$x^7 - 2x^6 + x^5 - x^4 + x^3 - 2x^2 + x$	*R2*	$(x^5 - 3x^3 + 1)/(x^2 + 1)$
Nonic	$x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$	*R3*	$(x^6 + x^5)/(x^4 + x^3 + x^2 + x + 1)$

Table 2. The absolute error with 0.95 confidence interval, committed by the best-of-run individuals on *training set* (mean of 150 runs)

	Sextic	*Septic*	*Nonic*	*R1*	*R2*	*R3*
GP	0.002 ±0.001	0.159 ±0.040	0.122 ±0.041	0.441 ±0.102	0.231 ±0.040	0.184 ±0.026
RX_3	0.002 ±0.001	0.130 ±0.039	0.128 ±0.040	0.175 ±0.039	0.130 ±0.028	0.164 ±0.027
LGX_3	0.003 ±0.001	0.109 ±0.034	0.114 ±0.041	0.170 ±0.033	0.086 ±0.020	0.179 ±0.024
NHX_3	0.001 ±0.001	0.138 ±0.047	0.085 ±0.039	0.292 ±0.073	0.145 ±0.033	0.141 ±0.030
RX_4	0.005 ±0.002	0.102 ±0.024	0.130 ±0.035	0.140 ±0.035	0.101 ±0.019	0.076 ±0.014
LGX_4	0.001 ±0.001	0.044 ±0.011	0.043 ±0.009	0.061 ±0.014	0.041 ±0.013	0.028 ±0.007
NHX_4	0.002 ±0.001	0.084 ±0.022	0.063 ±0.014	0.102 ±0.022	0.060 ±0.014	0.045 ±0.010
GP_{time}	0.002 ±0.001	0.102 ±0.031	0.070 ±0.024	0.210 ±0.041	0.137 ±0.028	0.085 ±0.015

the 20 fitness cases. The selection method is tournament of size 7, crossover likelihood is 0.9 and reproduction likelihood is 0.1. There is no mutation involved. Other parameters are set to defaults used in the ECJ package [7], which served as experimental environment.

Search Progress. Figure 1 presents the fitness of best-of-generation individuals averaged over 150 runs, along with 0.95-confidence intervals shown as shading. LGX_4 is an unquestionable winner in terms of speed of convergence, while LGX_3 makes much slower progress. This may be explained by the fact that the library it uses is almost two orders of magnitude smaller than that of LGX_4 (38 vs. 1697 procedures). As a consequence, the semantic diversity of the procedures inserted into offspring (the number of unique semantic) is here much lower, which deteriorates the algorithm's ability to perform effective exploration.

The fact that LGX outperforms RX is the main result of this study. It demonstrates that introducing 'medial' tendency in crossover makes the search process converge faster towards good solutions. It is particularly remarkable when we recall that LGX never affects the root node. Therefore, the effects of geometric-aware changes introduced into deeper tree nodes must propagate to its root, and, on average, improve the fitness of offspring more than for the other methods. This confirms the conclusion of our former study that dealt with a problem of more discrete nature [3].

Last but not least, the confidence intervals for LGX are much narrower than those for the other methods. The behavior of this method is thus much more predictable, and, in convenient circumstances, it should be possible to estimate the expected number of generations required to attain an assumed fitness level.

Importance of Homology. LGX adds two elements to standard subtree crossover: homology and the semantically geometric choice of procedures. Its

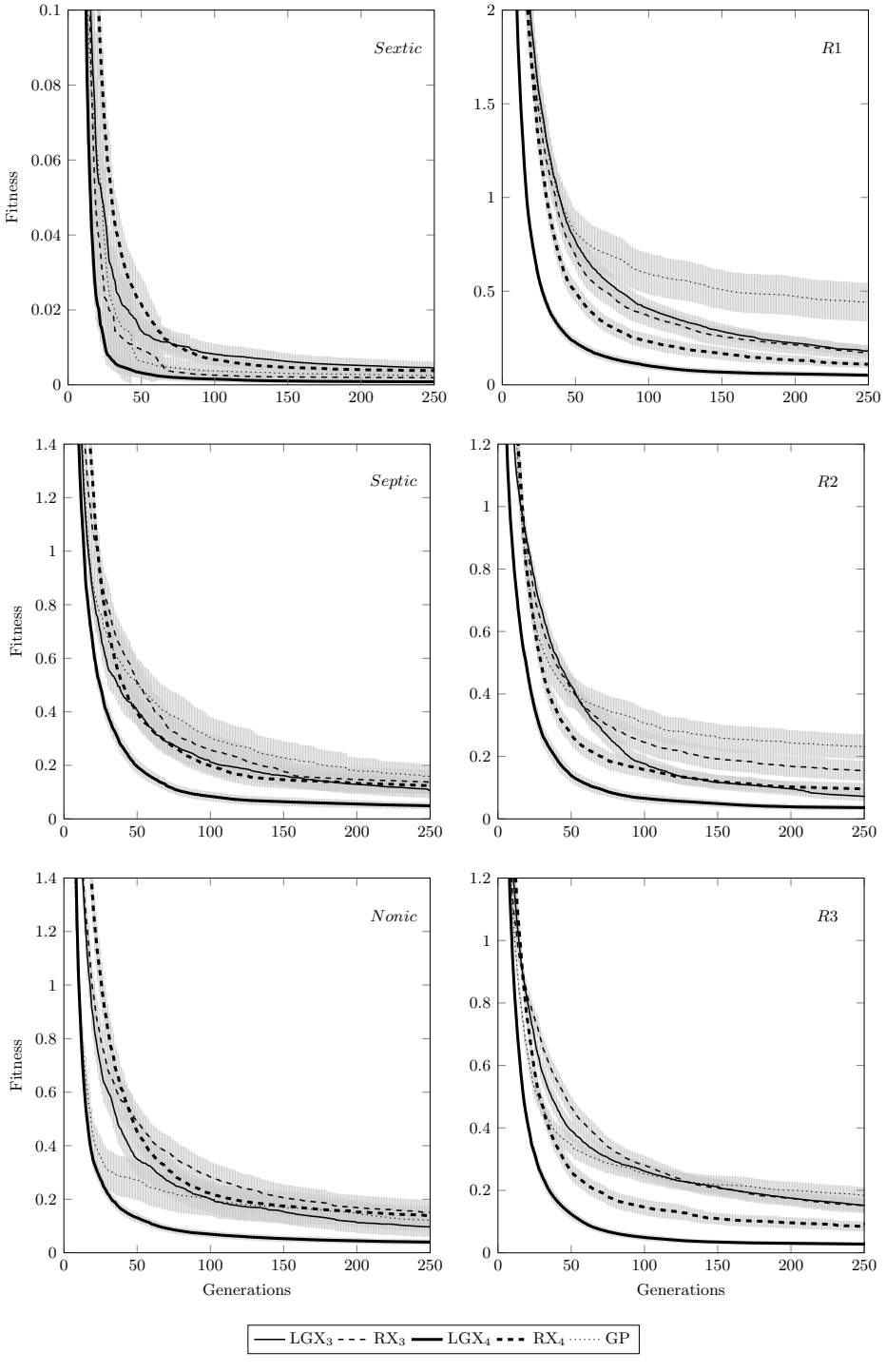

Fig. 1. Best-of-generation fitness graphs averaged over 150 evolutionary runs

superiority of LGX to RX demonstrates that the latter is essential. However, would LGX perform equally well if it was not homologous? To settle this issue, we prepared an additional control setup that uses a non-homologous but locally geometric crossover operator (NHX). NHX mimics LGX except for the choice of loci to be affected, where it works as the standard tree-swap crossover, i.e., selects them in both parents randomly and independently. Then it finds the semantically most medial procedure in the library and inserts it into both parents.

Table 2 compares the final (end-of-run) fitness of best-of run individuals evolved by NHX with the other methods. For $h = 3$ (small library), the duel between NHX and LGX is inconclusive, methods win or lose depending on the problem. However, for $h = 4$ the conclusion is clear: LGX yields lower error rate and not worse variance than NHX. Homology is then an essential component for this operator that significantly contributes to its performance.

Impact on Tree Size. By being homologous, LGX can be expected to affect also tree size. Figure 2 depicts the mean number of nodes per individual, calculated over all individuals in populations and averaged over 150 runs. The results are very similar across all benchmark problems. The methods using large library ($h = 4$) suffer from substantial bloat that is more severe than for the small library ($h = 3$). This can be easily explained. The mean tree depth of procedures in the library is greater for $h = 4$ than for $h = 3$. On average then, every act of crossover brings more genetic material to the population in the former case.

Another observation following from Fig. 2 is that RX suffers from bloat more than LGX. This suggests that the semantically close-to-geometric procedures inserted by LGX are on average shorter than the procedures selected from the library at random by RX. Our explanation for this phenomenon pertains to the relation between lengths of procedures and their location in the semantic space. Typically, short procedures will have semantics of small magnitudes, as it is unlikely to produce large values using arithmetic instructions that operate on numbers form interval $[-1, 1]$ (with obvious exception of the division operator). Such semantics will crowd closely around the origin of semantic space. On the contrary, longer procedures are capable of producing larger output values, which correspond to semantics that are distant from the origin. Also, every long procedure that is semantically equivalent to a shorter procedure is discarded when the library is being built (see Sec. 2). LGX, which looks for procedures that are semantically medial with respect to parents' subtrees (cf. s_m in Eq. (2)), is more likely to generate s_m that is close to the origin of semantic space. As a consequence, it selects shorter procedures more frequently.

Test-Set Performance. To assess the generalization capability of the considered methods, we employed a test set composed of 20 cases drawn randomly from the interval $[-1, 1]$, with uniform distribution. The best-of-run individual for each run is executed on these cases, and its generalization capability is expressed in the same terms as for the training process – the absolute error.

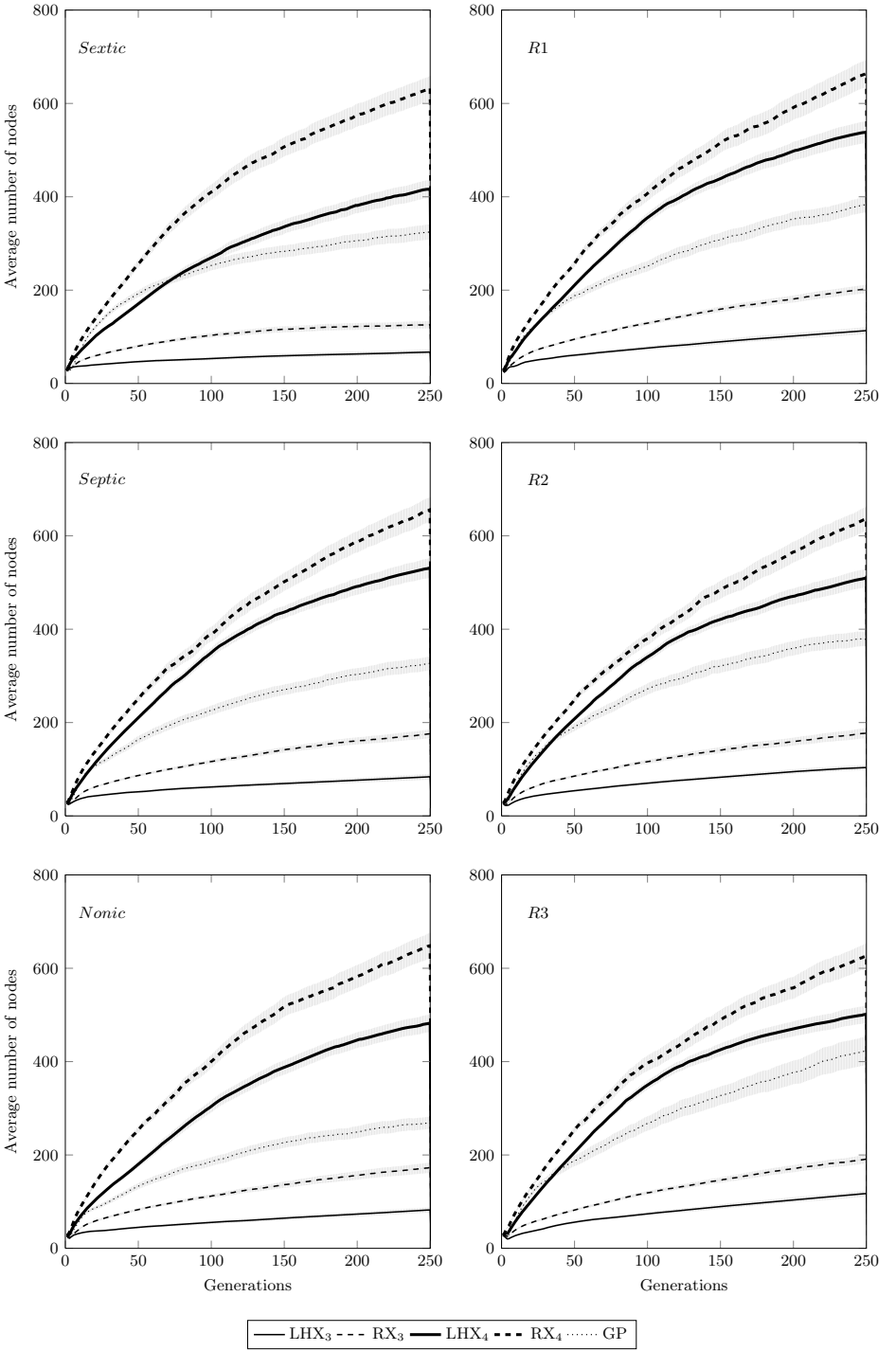

Fig. 2. Number of nodes (population mean) averaged over 150 evolutionary runs

Table 3. The absolute error with 0.95 confidence interval, committed by the best-of-run individuals on *test set* (mean of 150 runs)

	Sextic	*Septic*	*Nonic*	*R1*	*R2*	*R3*
GP	0.009 ±0.007	0.233 ±0.068	0.182 ±0.070	0.483 ±0.120	0.302 ±0.109	0.230 ±0.040
RX_3	0.004 ±0.002	0.140 ±0.040	0.146 ±0.047	0.191 ±0.044	0.129 ±0.028	0.167 ±0.037
LGX_3	0.005 ±0.002	0.122 ±0.038	0.117 ±0.045	0.226 ±0.069	0.086 ±0.020	0.163 ±0.020
RX_4	0.029 ±0.031	23.443 ±42.628	0.262 ±0.102	8.025 ±14.853	0.196 ±0.055	0.316 ±0.149
LGX_4	0.003 ±0.002	0.116 ±0.032	0.099 ±0.026	0.102 ±0.027	0.077 ±0.032	0.103 ±0.029

Table 3 presents the test-set errors of the best-of-run individuals averaged over all runs. Comparison of these figures with fitness values achieved on the training set (Table 2) leads to conclusion that all methods suffer from overfitting. In terms of the ratio of test-set error to training-set error, GP is superior. However, in absolute terms, LGX_4 attains the lowest error on the test set for all problems and has also the lowest variance. This is particularly interesting, because LGX_4 yields substantially bigger trees than GP (Fig. 2).

Time Complexity. The benefits of LGX come at extra computational cost of creating the library and searching for the semantically similar procedures. While the former turns out to be low (1–2 seconds compared to 100–150 seconds of the cost of entire run), the latter cannot be ignored. The roughly $250 \times (1024/2) \times 0.9 = 115,200$ R-tree queries per run make LGX substantially slower. In effect, its overall runtime is on average 2.8 times longer than GP's. This, together with the curves in Fig. 1, urges us to ask the question: will LGX maintain its superiority to GP with same time allocated to both methods?

To verify this possibility, we conducted an additional experiment, which consisted in giving GP the same amount of time as corresponding LGX runs took. By comparing the results of these runs, presented in the last row of Table 2 (GP_{time}) with the final fitness values of LGX, we conclude that GP, despite having more time, cannot catch up LGX_4, although it manages to reach the performance level of LGX_3 for some problems. Therefore, LGX can be considered attractive not only from theoretical viewpoint, but also in practical perspective.

4 Conclusion

The main conclusion of this study is that search operators that are at the same time homologous and semantically medial can improve the efficiency of GP search and cause the evolved programs generalize better. The experimental analysis suggests that both these properties are essential. It can be hypothesized, though remains to be verified that, with time of evolution, LGX causes emergence of a common semantic blueprint in the population, with the subprograms located at particular loci specializing at solving certain subproblems of the original problem. This hypothesis sounds very attractive, as it implies a capability for discovering semantic modules in the structure of the problem, which in turn could provide the possibility of problem decomposition.

LGX has been presented and verified here in the context of symbolic regression, but it has wider applicability. Any domain for which semantics are computable and a semantic metric is available, can be subject to this approach. In particular, if the metric $||\cdot||$ is not a norm, there are alternative ways in which a crossover can be made semantically medial [3].

Acknowledgment. Work supported by grant no. DEC-2011/01/B/ST6/07318.

References

1. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. ACM SIGMOD Conf., p. 47, Boston, MA (June 1984); Reprinted in Stonebraker, M.: Readings in Database Sys. Morgan Kaufmann, San Mateo, CA (1988)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
3. Krawiec, K.: Medial Crossovers for Genetic Programming. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) EuroGP 2012. LNCS, vol. 7244, pp. 61–72. Springer, Heidelberg (2012)
4. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Raidl, G., et al. (eds.) GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, Montreal, July 8-12, pp. 987–994. ACM (2009)
5. Krawiec, K., Pawlak, T.: Locally geometric semantic crossover. In: GECCO 2012: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, Philadelphia, PA, USA. ACM Press (accepted, July 2012)
6. Krawiec, K., Wieloch, B.: Automatic generation and exploitation of related problems in genetic programming. In: IEEE Congress on Evolutionary Computation (CEC 2010), July 18-23. IEEE Press, Barcelona (2010)
7. Luke, S.: The ECJ Owner's Manual – A User Manual for the ECJ Evolutionary Computation Library, zeroth edition, online version 0.2 edition (October 2010)
8. McPhee, N.F., Ohs, B., Hutchison, T.: Semantic Building Blocks in Genetic Programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 134–145. Springer, Heidelberg (2008)
9. Moraglio, A., Krawiec, K., Johnson, C.: Geometric semantic genetic programming. In: Igel, C., et al. (eds.) The 5th Workshop on Theory of Randomized Search Heuristics, ThRaSH 2011, Copenhagen, Denmark, July 8-9 (2011)
10. Moraglio, A., Poli, R.: Topological Interpretation of Crossover. In: Deb, K., Tari, Z. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 1377–1388. Springer, Heidelberg (2004)
11. Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Semantic Aware Crossover for Genetic Programming: The Case for Real-Valued Function Regression. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 292–302. Springer, Heidelberg (2009)
12. Papadopoulos, A., Manolopoulos, Y.: Performance of Nearest Neighbor Queries in R-Trees. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 394–408. Springer, Heidelberg (1996)
13. Poli, R., Langdon, W.B.: Schema theory for genetic programming with one-point crossover and point mutation. Evolutionary Computation 6(3), 231–252 (1998)
14. Ryan, C., Keijzer, M., Cattolico, M.: Favorable biasing of function sets using run transferable libraries. In: O'Reilly, U.-M., et al. (eds.) Genetic Programming Theory and Practice II, May 13-15, ch.7, pp. 103–120. Springer, Ann Arbor (2004)