Genetic Programming and Evolvable Machines (2018) 19:535–563
https://doi.org/10.1007/s10710-018-9325-4

Comparison of semantic-based local search methods for multiobjective genetic programming

Tiantian Dou[1] · Peter Rockett[1]

Received: 19 December 2017 / Revised: 1 June 2018 / Published online: 5 July 2018

Abstract
We report a series of experiments that use semantic-based local search within a multiobjective genetic programming (GP) framework. We compare various ways of selecting target subtrees for local search as well as different methods for performing that search; we have also made comparison with the random desired operator of Pawlak et al. using statistical hypothesis testing. We find that a standard steady state or generational GP followed by a carefully-designed single-objective GP implementing semantic-based local search produces models that are mode accurate and with statistically smaller (or equal) tree size than those generated by the corresponding baseline GP algorithms. The depth fair selection strategy of Ito et al. is found to perform best compared with other subtree selection methods in the model refinement.

Keywords Semantic-based genetic programming · Local search · Multiobjective optimization · Model selection

1 Introduction

It is well established that genetic programming (GP) exhibits good performance on the empirical modeling of complex systems [41]. Nonetheless, traditional GP still has the limitation that since it acts at the syntactic level, a small syntactic modification can produce a dramatic change in program fitness, which can harm search efficiency.

To address these issues, the integration of local search into GP has attracted significant attention [20, 45]. At a wider level, the hybridization of population-based global search with heuristic local search—often termed a memetic algorithm

✉ Peter Rockett
p.rockett@sheffield.ac.uk

Tiantian Dou
tdou1@sheffield.ac.uk

1 Department of Electronic and Electrical Engineering, University of Sheffield, Pitt Street, Sheffield, S1 4ET, UK

[35]—has achieved notable successes [37] although remains comparatively little used in GP. Typically, existing GP memetic algorithms may include hill climbing local search over the coefficients or the model structures of the GP solutions.

In GP, semantics usually refers to the vector of output values a program produces over the training data [50] and has been the subject of much recent research. Experimental results to date suggest that awareness of semantics is a great help both in maintaining population diversity and improving search power. Among these approaches, semantic-based local search methods [13, 50] have exhibited promising performance. Based on semantics, Pawlak et al. [40] have implemented a novel random desired operator (RDO), which decomposes the search task into a series of subtasks. By backpropagating the desired semantics to individual subtrees, the fitness of a solution can be improved by seeking to replace a selected subtree with a subtree better matching the desired semantics. Since replacement subtrees are selected from a (pre-computed) library, it is unclear whether RDO is a local search method or a (global) crossover method as claimed in [40]; clarifying this question is a part of the motivation for the present paper.

Local search inevitably adds to the computational burden and runtime of GP, which seemingly makes many practitioners wary of local search-based approaches. For this reason, we have restricted the present paper to local search—where employed—for tuning the solutions at the end of a conventional GP run. We believe this approach thus closely fits with the conventional memetic algorithm formulation [35] (although some may argue that in a memetic algorithm, the local search would be embedded within the global search). The principal contribution of this paper is an investigation of the performance of GP approaches when supplemented by semantically-aware local search methods. In particular, this paper extends consideration of the effectiveness of local search to a multiobjective (MO) GP framework since this explicitly trades off goodness-of-fit against model complexity, a key requirement in the empirical modeling of data [10, 30]. For the reasons stated in the preceding paragraph, we also make comparison with the RDO approach. We specifically constrain the scope of this work to local search methods that modify the morphologies of the GP trees rather than approaches described in Sect. 2 that fine-tune node functionality. Local search methods that change the tree morphologies are comparatively little explored.

2 Related work

2.1 Semantically-aware methods in GP

The study of semantics in GP has been an active topic since the term was first proposed by McPhee et al. [33]. As an evolutionary method, GP faces the issue that the 'shapes' of initial trees can be rapidly lost within a few generations. Traditionally, a diverse initial population, which plays an important role in a successful GP run, has usually been obtained with the ramped half-and-half method [25]—diversity here has to be interpreted at the syntactic level. Beadle [3] proposed a semantically-driven initialization algorithm to produce a diverse initial population at the phenotypic level. Compared to the ramped half-and-half method, increased semantic diversity seems to have a positive effect on GP search efficiency. Jackson [23], however, pointed out that it is not sufficient to ensure semantic diversity only in the initialization stage since a lack of semantic diversity diminishes the exploratory power of GP over the whole run. This author concluded that measures to promote syntactic diversity produced few gains, but those designed to produce semantic diversity generated a noticeable performance improvement. Based on the evaluation of behavioral changes caused by structural modification as a result of mutation, Beadle and Johnson [4] proposed a semantically-driven mutation operator to prevent the creation of new offspring with equivalent performance to that of their parents.

Locality in GP [15] measures the effect of a genotypical change on the phenotype, which is a crucial prerequisite to prevent evolutionary algorithms from behaving as pure random search. Uy et al. [47] compared the roles of syntactic and semantic localities of crossover in GP, and pointed out that improving syntactic locality reduced tree size and produced a slight improvement in model generalization. In contrast, improving semantic locality was more effective in reducing tree size and improving model generalization. These authors also proposed a number of semantic-based crossover and mutation operators.

Krawiec et al. [28] proposed the approximately geometric crossover (AGC), which combined a geometric crossover operator with semantic backpropagation. The semantics were used for guiding the crossover operation during evolution; these operators were further generalized in [40]. The recently-proposed RDO and approximately geometric semantic crossover (AGX) operator use semantic backpropagation to identify intermediate subtasks during the evolution process, and then solve these using an exhaustive search method. When compared with other semantic-aware operators and standard genetic operators, RDO and AGX were shown to exhibit improved performance on a series of symbolic regression and boolean benchmark problems. Though generating promising performance, a major weak point of these algorithms is that the child solutions are typically larger than their parents, which may lead to unacceptably slow fitness evaluations after a few generations. In this paper, we adopt the semantic backpropagation strategy of [40] for producing the desired output vectors of subtrees to serve as a basis for selecting better-performing replacements.

Geometric Semantic Genetic Programming (GSGP) has aroused the interest of an increasing number of researchers. Moraglio et al. [34] introduced a novel set of semantically-aware genetic operators to search the underlying semantic space directly. GSGP, however, has a major shortcoming in that the size of the individuals grows exponentially during the evolution, which makes it impractical for complex, real-life applications. Vanneschi et al. [49] overcome this limitation by introducing new, efficient geometric semantic operators. Castelli et al. [6] proposed geometric semantic genetic operators that enabled them to solve complex, real-world problems efficiently. Moreover, Ruberto et al. [43] presented a new genetic programming framework by introducing two concepts: optimally aligned, or optimally coplanar, individuals, which outperformed the standard GSGP. Nevertheless, Ruberto et al. omitted problems for which they were unable to find aligned or coplanar individuals and the generalization to unseen data was not clear. Gonçalves et al. [17] addressed these questions by using a geometric semantic hill climber to explore the search space.

This work provided a new insight into the relationship between program syntax and semantics, and allows for the principled, formal design of semantic operators for various problems.

2.2 Local search in GP

The combination of local search and evolutionary global search [16, 18] has been widely studied and shown to be a powerful strategy for improving search efficiency although this is less commonly employed in GP. Local hill climbing has been integrated into GP either for tuning numerical coefficients [1, 20, 27, 45, 54, 56], or fine-tuning the model structure [2, 13, 19, 26, 29, 31, 53, 55]. Interleaved with global search, the parameters of solutions in each generation have been optimized via: relabeling [20], genetic local search [27], gradient descent [45, 56], and linear scaling [1, 2].

Many hill climbing local search methods have been embedded in standard GP for model structure optimization. Harries and Smith [19] proposed a non-evolutionary based GP with several genetic operators to evolve solutions in a hill climbing manner. Later, a co-evolving memetic algorithm [26] was introduced to produce solutions for the comparison of protein structures by integrating co-evolving local searches with GP. Wang et al. [53] optimized decision trees using a splitting operator to divide the whole sample space into subspaces, and then conducted a hill-climbing tuning process. Zhang et al. [55] introduced the new crossover operator, called looseness control crossover, to find good building blocks by continually crossing over selected parents in a hill climbing manner. Looseness values assigned to each link between adjacent nodes prevent disruption of good building blocks in subsequent operations.

As the traditional crossover operator has often been criticized for being less powerful in forming good offspring solutions, Majeed [31] proposed a semantic context-aware crossover operator for breeding better child solutions with high fitness gain. This operator identified the best possible crossover point in each selected subtree by examining all possible contexts in which a subtree can be grafted, finally selecting the site where the highest fitness is attained.

Azad and Ryan [2] proposed a method to tune the internal nodes of trees one-at-a-time by trying all possible nodes with the same arity, and retaining the modification if a change of node improved the fitness. Although the method demonstrated performance improvements, this is an extreme form of local hill climbing that is unable to modify the ‘shape’ of a tree.

Since it is only able to explore syntactic space, canonical GP is deficient at determining the (implicit) parameters of a particular program. In order to address this deficiency, Z-Flores et al. [54] developed a Lamarckian memetic GP incorporating a local search strategy to optimize parameters embedded in the nodes of the GP trees. These authors concluded that incorporating local search improves convergence and performance while reducing code growth. As with the work in [2], the approach of

Z-Flores et al. does not use local search to modify the functional form (‘shape’) of the tree although whether this approach is effective due to also modifying selective pressures within the population is possible but as yet unexplored. The work in [54] was extended in [24] by hybridization with the neuro-evolution of augmenting topologies (NEAT) method.

For combining the exploration ability of semantic genetic programming and the exploitation ability of local search, Castelli et al. [7] integrated semantic mutation operators [34] with a local search method of solving a problem in energy consumption forecasting. This case study resulted in good model accuracy with a speeded-up search process. In order to accelerate convergence, Castelli et al. [8] proposed a hybrid algorithm combining GSGP and the above method. The results show this hybrid method allows the search to converge quickly while also exhibiting a noteworthy ability to limit overfitting.

Inspired by the RDO algorithm, Ffrancon and Schoenauer [13] proposed a local tree improvement (LTI) operator within a standard local search framework to find the best possible semantic match between all subtrees in a parent tree and all programs in a pre-constructed library. This semantic-aware method performed well on several boolean benchmark problems.

La Cava et al. [29] claimed that the performance of stack-based GP can be improved by embedding local search using epigenetic instructions to specify active and silent genes. In contrast to tree-based GP, stack-based GP is “syntax-free” and syntactic validity is guaranteed no matter how the epigenetic instructions change.

Very recently, Trujillo et al. [46] have argued that local search is necessary to allow GP to reach its full potential; these authors also note that local search seems comparatively little utilized by the GP community.

3 Experimental methodology

3.1 Evolutionary framework

In the context of empirical modeling using GP, Le et al. [30] have recently reviewed the use of complexity measures, and point out the critical importance of trading off goodness-of-fit to the training data against model complexity; see also [36, Chap 7]. To explicitly address this trade-off here, we have used a global multiobjective GP formulation in this work with conventional tree-based individuals where the single population was sorted according to Pareto dominance. We have employed both the sorting approach and selection method of Fonseca and Fleming [14]. We have employed both generational and steady-state evolutionary strategies for ‘global’ search followed—optionally—by local search over the final populations; we make detailed comparisons below.

Experimental details of the basic evolutionary algorithm are shown in Table 1. This, we believe, is a fairly standard configuration except we have used the analytic quotient operator [38] instead of protected division to avoid near-singularities in the solutions. We have employed the straightforward complexity measure of tree node count in our multiobjective formulation since this gives a direct measure of the

Table 1 Evolutionary parameters used in this work

Parameter	Value
Population size	100
Initialization method	Ramped half-and-half; maximum tree depth = 6
Number of evolutionary generations	222
Function set	$+, -, \times$, Analytic quotient [38]
Terminal set	Input variables; constants in 0.1, 0.2..., 0.9
Conventional GP Elitism	Top 10 solutions survive
Conventional GP operators	Point crossover + point mutation (tree depth ≤ 4)
RDO-based GP Elitism	None
RDO-based GP search operator	Static library (maximum tree depth = 4)
Subtree selection method	Equal node probability OR
	Equal depth probability OR
	Ito depth-fair selection [22]

computational burden of evaluating a tree. The imposition of evolutionary pressure to reduce node count is also an effective way of controlling tree bloat.

Using the normal definition of semantics as the indexed output vector of tree responses over the training data, the semantics of each node within the tree were estimated recursively and stored when it was evaluated for the first time. The calculation of the desired semantics starts from the root node and propagates along all paths to all leaves. Since the desired output of the root node of a tree is known, the desired semantics of each child node in the tree can be calculated assuming that its siblings have the correct structure. If the backpropagation process yields multiple possible values, one is chosen arbitrarily; if the value is undefined, it is ignored in the subsequent calculations of semantic distances between subtrees. See Pawlak et al. [40] for further details.

We have considered the basic evolutionary GP algorithm followed by one of a number of different local search methods; the aim in each case was to reduce the Euclidean distance between the subtree's actual and desired outputs. We investigated a number of strategies for selecting subtrees for replacement that we detail below. Local search has been restricted to the final population in order to keep the computation times within practical limits.[1] In addition, we also include results from the basic GP without local search as a baseline case.

Since it is a prominent example of semantic-based search, we have also included the RDO operator [40] as a comparator. This method uses a library of semantically-unique programs, and when a subtree in a parent is selected during the evolutionary process, a new offspring is generated by replacing the selected subtree with the library program exhibiting the closest match to the subtree's desired semantics. (This

[1] Additionally, we have observed (unpublished) that, apart from significantly increasing the computation time, applying local search to every generation is ineffective because the conventional evolutionary operators of crossover and mutation are so highly disruptive. These results will be published elsewhere.

strategy has the disadvantage that growth in the overall size of the parent tree is not explicitly constrained.) We have used only a static library of trees up to a predefined size limit, precomputed before the evolutionary process commences since this has been shown to yield superior performance to the alternative of a dynamic library [40]. Further, we have used more modest library sizes compared to the 100,000 used by Pawlak et al. because we are concerned with the practical application of the method, and therefore its runtime; even with a reduced library size of 1000, the runtime of the RDO-based method was typically 30 times longer than that of the baseline GP approach. Static libraries were generated with a maximum tree depth of 4, and an initial library size of 1000 that was then reduced by removing semantic duplicates; typically 5% of the library individuals were removed at this stage. Within RDO, we have explored a range of subtree selection approaches—see Sect. 3.3 for full details. The algorithm settings are shown in Table 1.

3.2 Local search methods

We have applied one of a number of local search approaches to the final population obtained from the baseline GP algorithm. These comprise two key elements: (1) the method for selecting a target subtree upon which local search acts, and (2) the method for generating a (potentially) better subtree. Local search was applied to every individual in the final population generated by the baseline GP algorithm. Note that we have not selected a final, single model for evaluation until *after* local search was applied to the whole population. See Sect. 3.5 below for further details on the numbers of times local search was applied.

3.3 Subtree selection

We have employed three different subtree selection methods in this work.

- Equal Node Probability. Selection where each node in the parent tree is chosen with equal probability to be the root of the target subtree; algorithms using this subtree selection method are denoted with a ‘1’.
- Equal Depth Probability where the selection method first chooses a depth value in the range zero (i.e. the parent’s root node) to the maximum depth of the parent tree, with uniform probability. At this point, one of the nodes at the selected depth is chosen with equal probability. Algorithms using this strategy are denoted with a ‘2’.
- Ito’s Depth-fair Selection. Node selection using the depth-fair selection method of Ito et al. [22]. This method is similar to (2) above except that the probability of selecting a given depth halves for every increase in tree depth (subject to the usual normalization condition that the sum of depth selection probabilities is unity). This approach gives nodes at the higher levels of a tree a greater chance of being selected. Algorithms employing this method are denoted with a ‘3’.

All three methods of subtree selection embody different biases as to how nodes (i.e. target subtrees) are chosen.

3.4 Algorithm comparisons

Clearly a fundamental objective in this work has been to make fair comparisons between some quite different algorithms. To compare the baseline generational GP, steady-state GP and RDO global algorithms is fairly straightforward: we allowed each to run for the same number of local search tree evaluations. This allows each algorithm to make the same number of 'moves' in its search, leading to a reasonable basis for comparison although we restate that the runtime of the generational RDO algorithm with subtree selection method '1' above (GenRDO-1) was typically 30 times longer than for the baseline generational GP (GenGP). Establishing a fair basis for comparison with the various local search algorithms, however, is more problematic. We have addressed this by measuring the process time of the GenRDO-1 algorithm on each benchmark problem, and then limiting the total runtime of one of the local search-based algorithms that uses generational global search followed by generational GP local search with Ito depth-fair selection[2] to this figure. The total number of tree evaluations in this algorithm was noted and used as a limit for all the other local search methods. Local search was continued by cycling over the population, attempting to improve one subtree in every individual per cycle, until the allowed number of local search tree evaluations was exhausted. Thus all algorithms were compared on the basis of being allowed equal amounts of computational 'effort' as gaged by numbers of tree evaluations.

3.5 Subtree generation and replacement

In conjunction with different methods of subtree selection, we have used a number of different methods to generate candidate subtrees to use as replacements. In all cases, the objective was to generate a replacement subtree with semantics more closely matched to the desired (back-propagated) semantics than those of the original selected subtree:

– Generational GP to Generate New Subtrees. A single objective generational GP was used to search for a tree better matching the desired semantics; apart from the objective function and restricting the local search GP to 100 generations, the evolutionary parameters were as detailed in Table 1. A hard limit was placed on the number of tree nodes in the local search GP. This limit on replacement subtree sizes was set equal to the node count of the original target tree to be replaced in order to prevent code growth in the parent. Candidate replacement subtrees were thus, at worst, the same size as the originals they sought to replace. (This is in quite deliberate contrast to the RDO operator [40], which is ambivalent about

[2] Designated as algorithm 'GenGP-GenGP-3' below.

code growth.) If an evolved subtree had a smaller mean squared error (MSE) over the semantic target, it was used to replace the original subtree; otherwise, the parent tree remained unaltered.

- Steady-state GP to Generate New Subtrees. Similarly, a single-objective steady-state GP with hard limit on replacement subtree sizes was applied for tuning subtrees so as to better approximate the desired semantics.
- Random Generation of New Subtrees. Randomly generating replacement subtrees of the same or smaller node count than the original target subtree; again, this size restriction was designed to prevent growth of the parent tree. For a given parent tree, one cycle of local search comprised first selecting a target subtree, and then randomly generating a sequence of candidate replacement subtrees with randomly-generated node counts less than or equal to the node count of the target subtree. If a candidate subtree produced a closer semantic match than the original subtree, it was immediately used for replacement and the random subtree generation sequence terminated. The number of attempts at replacing a given subtree was limited to a maximum of 100, and if no suitable replacement was generated, the subtree was left unchanged. This search procedure was continued by cycling over the population, attempting to improve a single selected subtree in each individual, until the limit on the number of tree evaluations was reached.
- Using RDO as local search operator to Generate New Subtrees. We have also investigated using RDO as a local search method to improve the final population generated by the baseline global search algorithms—essentially, replacing the local search by random subtree generation with selection of replacements from an RDO-style static library. The RDO operator selects a program that exhibits the closest match to the desired semantics of a selected subtree. We have observed, however, that, when using RDO as a local search method, search over the static library does not necessarily yield a candidate replacement subtree with *better* semantics than the original target subtree. Consequently, we have employed two different criteria for accepting tree modification by a subtree identified from the static library: firstly, we always accept a best-matching candidate subtree ("Best matching subtree"). Second, we only accept a candidate subtree if it both has better-matching semantics to the selected target subtree, *and* the modified tree Pareto-dominates the original parent tree, i.e. it achieves a lower MSE and/or lower node count ("Better matching subtree"). As above, local search cycled over the population attempting to improve one subtree at each pass.

In what follows, we adopt the naming convention for describing a particular experimental configuration of:

- Global multiobjective search paradigm either generational('Gen') or steady-state ('SS').
- The global search method, either GP, or RDO.
- Local single-objective search method: generational('Gen') GP, steady-state ('SS') GP, random tree generation ('Ran'), or RDO ('RDO').
- The method for selecting the subtree for replacement: equal node probability ('1'), equal depth probability ('2'), or Ito's depth fair selection ('3').

Table 2 Test functions

Problem	Function	Domain
F1:Automatic French curve [52]	$y = 4.26(\exp^{-x} - 4\exp^{-2x} + 3\exp^{-3x})$	$[0 \cdots 3.25]$
F2:Sextic polynomial [48]	$y = x^6 + x^5 + x^4 + x^3 + x^2 + x$	$[-1 \cdots +1]$
F3:Uy5 [48]	$y = \sin x^2 \times \cos x + 1$	$[-1 \cdots +1]$
F4:Uy6 [48]	$y = \sin x + \sin(x + x^2)$	$[-1 \cdots +1]$
F5:Vladislavleva [51]	$y = 8\exp^{-x} x^3 \cos x \sin x(\cos x \sin^2 x - 1)$	$[0 \cdots +10]$
F6:Chebyshev polynomial [39]	$y = 3\cos(3\cos^{-1} x)$	$[-1 \cdots +1]$
F7:Scaled sinc function [39]	$y = 5\sin x/x$	$(0 \cdots +10]$
F8:Cubic polynomial [48]	$y = x^3 + x^2 + x$	$[-1 \cdots +1]$
F9:Quartic polynomial [48]	$y = x^4 + x^3 + x^2 + x$	$[-1 \cdots +1]$
F10:Quintic polynomial [48]	$y = x^5 + x^4 + x^3 + x^2 + x$	$[-1 \cdots +1]$
F11:Uy7 [48]	$y = \log(x + 1) + \log(1 + x^2)$	$[0 \cdots +2]$
F12:Uy8 [48]	$y = \sqrt{x}$	$[0 \cdots +4]$
F13:Seventh order polynomial [39]	$y = 23.7(x + 0.9)(x - 0.9)(x - 0.6)(x - 0.6)$ $(x + 0.8)(x + 0.4)(x + 0.3)$	$[-1 \cdots +1]$

Thus, "SSGP-GenGP-2" indicates a steady-state global GP followed by generational GP local search using equal depth probability method of subtree selection. "GenGP" and "SSGP" refer to the baseline global searches with no local search. In addition, for the reasons explained above, we have included two different acceptance strategies when using RDO as a local search operator: "Best matching subtree" and "Better matching subtree". These lead to additional variants, labeled '4', '5' and '6' only for global GP followed by RDO-based local search.

Summaries of the experiments conducted are shown in Table 3 for the methods employing generational global search, and in Table 4 for methods using steady-state global search.

3.6 Test functions

Although the subject of regression test functions for GP has received detailed consideration [32], we have employed a series of commonly-used benchmark univariate symbolic regression problems—see Table 2-previously used in the GP literature. For each function, we generated 250 independent training sets each containing 20 data uniformly sampled over the domain; the independent test set for each function comprised 10,000 data. The best test mean squared error (MSE) obtained from the final population (after any local search procedures) was taken as a measure of generalization performance, this being equivalent to the general procedure in single-objective GP.

3.7 Statistical testing

We have made detailed statistical comparisons of the results obtained. Since we cannot make any distributional assumptions about the results, we have used the

Table 3 Summary of experimental protocols used: generational global search

	Genetic search operator		Subtree selection method			Local search method				
	Standard xover + mutation	RDO static library	Equal node (1)	Equal depth (2)	Ito depth-fair (3)	GenGP	SSGP	Random generation	RDO	
									Better matching subtree	Best matching subtree
GenGP	✓									
GenRDO-1		✓		✓						
GenRDO-2		✓			✓					
GenRDO-3		✓				✓				
GenGP-GenGP-1	✓		✓			✓				
GenGP-GenGP-2	✓			✓		✓				
GenGP-GenGP-3	✓				✓	✓				
GenGP-SSGP-1	✓		✓				✓			
GenGP-SSGP-2	✓			✓			✓			
GenGP-SSGP-3	✓				✓		✓			
GenGP-Ran-1	✓		✓					✓		
GenGP-Ran-2	✓			✓				✓		
GenGP-Ran-3	✓				✓			✓		
GenGP-RDO-1	✓		✓						✓	
GenGP-RDO-2	✓			✓					✓	
GenGP-RDO-3	✓				✓				✓	
GenGP-RDO-4	✓		✓							✓
GenGP-RDO-5	✓			✓						✓
GenGP-RDO-6	✓				✓					✓

Table 4 Summary of experimental protocols used: steady-state global search

	Genetic search operator		Subtree selection method			Local search method				
	Standard xover + mutation	RDO static library	Equal node (1)	Equal depth (2)	Ito depth-fair (3)	GenGP	SSGP	Random generation	RDO	
									Better matching subtree	Best matching subtree
SSGP	✓									
SSRDO-1		✓		✓						
SSRDO-2		✓			✓					
SSRDO-3		✓				✓				
SSGP-GenGP-1	✓		✓			✓				
SSGP-GenGP-2	✓			✓		✓				
SSGP-GenGP-3	✓				✓	✓				
SSGP-SSGP-1	✓		✓				✓			
SSGP-SSGP-2	✓			✓			✓			
SSGP-SSGP-3	✓				✓		✓			
SSGP-Ran-1	✓		✓					✓		
SSGP-Ran-2	✓			✓				✓		
SSGP-Ran-3	✓				✓			✓		
SSGP-RDO-1	✓		✓						✓	
SSGP-RDO-2	✓			✓					✓	
SSGP-RDO-3	✓				✓				✓	
SSGP-RDO-4	✓		✓							✓
SSGP-RDO-5	✓			✓						✓
SSGP-RDO-6	✓				✓					✓

nonparametric Friedman test [11] under the null hypothesis that all the ranks of the results are drawn from the same distribution and therefore there is no difference between the varying treatments; we used the significance level of $P \leqslant 0.05$ to reject the null hypothesis. When the null hypothesis of the Friedman test was rejected, we used the Holm–Bonferroni post-hoc correction [11] to the significance level in a Wilcoxon signed ranks test [5, 11] to judge the statistical differences between pairs of results.

4 Results and discussion

Applying all the optimization approaches detailed in Tables 3 and 4 over each of the thirteen benchmark regression problems F1–13 in Table 2, and performing a Friedman test on the ranks of the best MSEs for all algorithms (treatments) and regression problems (subjects) indicated, we reach the conclusion that the null hypothesis—that each of the optimization approaches produces identical results—can be rejected with P values < 0.0001. There is thus strong evidence of differences between the experimental treatments. For obtaining detailed information on which algorithms are statistically significantly different from each other, we have carried out a series of pairwise tests using the Wilcoxon signed ranks test with a Holm–Bonferroni post-hoc analysis to constrain the family-wise error rates for the multiple comparisons [11].

Tables 5 and 6 show the mean ranks of test errors and tree sizes, respectively aggregated over all benchmark problems and treatments.

As a brief introductory overview, from Table 5 it is clear that the best-performing algorithm overall is SSGP-SSGP-3 followed by SSGP-GenGP-3. By contrast, the baseline SSGP algorithm ranks 8th overall, and the baseline GenGP algorithm 14th. GenRDO-1 is ranked third along with a number of other algorithms of various configurations. Regarding the significance of the gray-shaded cells in this table, there are no statistical differences between any of the 9th ranked group SSRDO-1 ...GenGP-Ran-3. On the other hand, there is a difference between SSRDO-1 and the 10th ranked GenGP-RDO-1, but no difference between GenGP-RDO-1 and the group SSGP-RDO-1 ...GenGP-Ran-3. We have highlighted this with the gray shading in the 10th column opposite the group SSGP-RDO-1 ...GenGP-Ran-3.

As regards node counts—rankings are shown in Table 6 where smaller rank denotes smaller trees—there is a broad inverse relationship between the rankings on test MSE and tree size. Algorithms involving steady-state approaches tend to be associated with larger trees, but tend to have smaller test MSEs. Again in this table, gray-shaded cells denote, for example, that there is no difference between any of the group SSGP-RDO-4 ...SSGP-SSGP-2, and SSGP-GenGP-1.

(In the more detailed discussion that follows, we use the shorthand terms “larger” and “smaller” in the sense of *statistically* larger (or smaller) at the 95% confidence level.)

The principal observations that can be drawn from these results are:

Table 5 Ranking of the mean squared test errors (MSEs) by algorithm; algorithms listed in the same column display no statistical difference

Overall ranks of MSE values for all the algorithms															
Rank '1'	Rank '2'	Rank '3'	Rank '4'	Rank '5'	Rank '6'	Rank '7'	Rank '8'	Rank '9'	Rank '10'	Rank '11'	Rank '12'	Rank '13'	Rank '14'	Rank '15'	
SSGP-SSGP-3															5.321
	SSGP-GenGP-3														5.807
		GenGP-SSGP-3													7.389
		GenGP-GenGP-3													7.618
		SSGP-SSGP-2													8.567692
		SSGP-GenGP-2													8.752308
		GenRDO-1													9.851538
			GenGP-SSGP-2												11.08538
			SSGP-Ran-2												11.63615
			SSGP-Ran-1												11.97385
			SSGP-Ran-3												13.22385
				SSGP-GenGP-1											14.37231
				SSGP-SSGP-1											14.62154
					GenGP-Ran-1										16.96154
					SSGP-RDO-5										17.44308
					SSGP-RDO-4										17.46
					SSGP-RDO-6										18.29231
						GenGP-Ran-2									19.11231
							SSGP								19.58538
								SSRDO-1							20.47154
								SSGP-RDO-1	▨						20.49
								GenGP-GenGP-2	▨						20.51615
								GenGP-GenGP-1	▨						21.00615
								GenGP-SSGP-1	▨						21.56
								GenGP-Ran-3	▨						21.62462
									GenGP-RDO-1						22.13923
										GenGP-RDO-4					25.48154
											GenRDO-2				26.14077
												GenGP-RDO-5			26.22538
													SSRDO-2		27.05846
													GenGP-RDO-2		27.47385
													SSGP-RDO-2		27.57308
													GenGP-RDO-6		28.64077
													SSRDO-3		31.67308
													GenGP		31.82
														GenRDO-3	33.98308
														GenGP-RDO-3	34.01308
														SSGP-RDO-3	34.03538

Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that the algorithms shown to their immediate left column have no statistical difference with the GenGP-RDO-1 algorithm in column 10. The rightmost column shows the actual mean rank values

Table 6 Overall ranking of node counts by algorithm; algorithms listed in the same column display no statistical difference

Overall ranks of tree sizes for all algorithms														
Rank '1'	Rank '2'	Rank '3'	Rank '4'	Rank '5'	Rank '6'	Rank '7'	Rank '8'	Rank '9'	Rank '10'	Rank '11'	Rank '12'	Rank '13'	Rank '14'	
SSRDO-3														6.314
	GenGP-Ran3													8.910
	GenRDO-3													8.919
		SSRDO-2												9.394
		GenGP-Ran-2												9.512
		GenGP-RDO-3												9.629
		SSGP-RDO-3												9.695
			GenGP-Ran-1											12.315
				SSRDO-1										14.881
				GenGP-RDO-6										15.365
				GenGP-RDO-5										15.467
					SSGP-Ran-2									15.634
					GenGP-RDO-4									15.989
					SSGP-Ran-1									16.026
						GenRDO-2								16.473
						GenGP-RDO-2								17.292
						SSGP-RDO-2								17.544
							SSGP-Ran-3							18.315
							GenGP							19.803
							GenGP-GenGP-2							20.369
							GenGP-SSGP-1							20.567
							GenGP-GenGP-1							21.477
							GenGP-SSGP-2							21.741
							GenGP-GenGP-3							22.775
							GenGP-SSGP-3							22.935
								SSGP-RDO-5						23.783
								SSGP-RDO-4	▨					24.149
								SSGP-GenGP-2	▨	▨				24.863
								SSGP-SSGP-2	▨	▨				25.014
									SSGP-GenGP-1	▨				25.172
										SSGP-RDO-6				25.234
										SSGP-GenGP-3				25.363
										SSGP-SSGP-3				25.463
										SSGP-SSGP-1				25.677
											SSGP			28.699
												GenRDO-1		31.23692
													GenGP-RDO-1	34.411
													SSGP-RDO-1	34.595

Conversely, a statistically significant difference is detected between algorithms in different columns. The gray-shaded cells denote that algorithms to their immediate left show no statistical difference to the algorithms in that column. The rightmost column shows the actual mean rank values

4.1 Comparison of generational and steady-state global strategies without local search

In the absence of any local search, the global steady-state (SSGP) strategy clearly produces smaller test errors than the corresponding generational strategy (GenGP), with mean ranks of 19.585 and 31.820, respectively. The generally superior performance of the steady-state strategy has previously been observed in the context of multiobjective genetic algorithms by Durillo et al. [12]. The average tree size of the models created by SSGP, however, is larger than the average tree size for GenGP strategy with mean ranks of 28.698 and 19.803, respectively. Since we are generally interested in models with smaller test errors and superior generalization, the results here suggest that, in the absence of local search, the steady-state strategy is better than the much more widely used generational strategy, extending the observations in [12] to another MOEA domain.

4.2 Influence of the global search strategy on the efficacy of a given local search method

Following on from the previous observation, we can examine the influence of the evolutionary global search strategies on local search methods. It is clear from Table 5 that a given local search algorithm following a steady-state global search performs better than the corresponding algorithm that uses generational GP local search, except for the three pairs: SSGP-RDO-1 versus GenGP-RDO-1, SSGP-RDO-2 versus GenGP-RDO-2, SSGP-RDO-3 versus GenGP-RDO-3, between which no statistically significant differences were detected. (It is noteworthy that all six algorithms in this ‘no difference’ category use RDO as the local search method; we observe below that RDO does not appear to be particularly good as a a local search technique. Thus it seems likely that these six algorithms are not representative results.) Since the starting point for all local search is the final population produced by the global search strategy, there seems strong evidence that the generally superior population produced by the steady-state strategy facilitates more productive local search, regardless of the local search algorithm employed. It seems logical that starting from a ‘better initial position’ will help the subsequent local search to find superior solutions.

At the same time, comparing the average tree sizes generated by the various algorithms, the trees generated by generational global search are statistically smaller on a like-for-like basis than those created by a steady-state GP, again except for the three pairings listed above for which no statistically significant differences can be detected. As pointed out above, however, if presented with this trade-off, most practitioners would favor the methods yielding the smaller generalization errors.

4.3 Comparing RDO in generational and steady-state global strategies

Algorithms using RDO as the genetic operator exhibit different performances when used with generational compared to steady-state evolutionary strategies. Compared to the baseline GenGP, the RDO genetic operator used in a generational strategy yields performances that range from the seventh best performer (GenRDO-1) via a middle-ranking performer (GenRDO-2) to rapid deterioration to one of the worst algorithms (GenRDO-3). GenRDO-3 performs even worse than the baseline GenGP due to the fact that the population in these runs invariably collapsed to a single or small number of identical individuals, thereby dramatically damaging the searching ability of the algorithm due to lack of diversity.

4.3.1 The role of evolutionary strategy with global RDO

The general performance of RDO as a search operator in a steady-state strategy, however, shows a great difference. The average test errors of all the SSRDO algorithms are statistically worse than those of the baseline SSGP. The inference is that subtree replacement from the randomly-initialized static library harms the search efficiency of a steady-state GP. A possible reason for this might be that the RDO operator, which replaces selected subtrees with specific randomly-generated library programs, induces significant disruption during a steady-state evolution process. The evolution process of a generational GP is itself highly disruptive since the majority of chromosomes in each new generation are produced through crossover operations; in this circumstance, the RDO operator appears to improve the search efficiency and generates more accurate trees than the baseline GenGP. The steady-state strategy, however, relies on a continual advancement towards the Pareto front that RDO seems to repeatedly disrupt leading to poor overall search performance.

4.3.2 The role of subtree selection strategy with global RDO

From the perspective of the subtree selection approach used with RDO, for both generational and steady-state strategies, algorithms selecting subtrees with equal node probability generate more accurate models than those using the equal depth selection method. Ito's depth fair subtree selection method produces the worst results. This suggests that the performance of the RDO operator is sensitive to the method of selecting subtrees.

4.4 The role of the generational and steady-state strategies for local search

From Table 5, clearly the SSGP-SSGP-3 is the best performer among all the algorithms. Unlike the previous observation that the global search ability of a steady-state GP is always better than a generational GP, when GP is used as a local search operator, the steady-state GP does not exhibit any consistent advantage over the generational GP.

When compared by subtree selection methods, however, algorithms using Ito's depth fair method produce the most accurate models. Algorithms selecting subtrees with equal node probabilities are ranked lowest among all the GP-based local search algorithms. This suggests that a subtree with a shorter path to the root node of its parent tree is likely to be more influential on the entire tree in the overall evaluation; this conclusion is consistent with a hypothesis proposed by Igel et al. [21]. To verify this, we investigated the relationships between success rate and MSE reduction with the normalized depth of selected subtrees. The normalized depth of each subtree is calculated by dividing the depth of a selected subtree from the root node by the full depth of the whole tree. All the selected samples were divided into ten groups according to their normalized depth with increments of 0.1. A illustrative group of five of the above thirteen benchmark functions were used, and the corresponding graphs of the relationships between successful subtree replacement rate (Fig. 1) and MSE reduction (Fig. 2) with normalized depth of selected subtrees.

From the graphs in Fig. 1, the success rates are roughly constant with increasing normalized depth values, which shows that subtrees of different normalized depths have almost identical probabilities of being successfully replaced. This suggests that the good performance of algorithms using Ito's depth fair subtree selection method is not caused by more frequent modification of subtrees near the root node of a GP tree. From the graphs in Fig. 2, an inverse relationship between the magnitudes of MSE reduction with the normalized depth of selected subtrees can be observed. This implies that a more efficient optimization of GP trees can be achieved by selecting subtrees with shorter path to the root node. In other words, an improvement of subtrees near the root node is more likely to have a larger beneficial effect on the whole tree. We consider this the main reason that causes good performances of algorithms using Ito's depth fair selection method.

4.5 Influence of the number of cycles of local search

Whether it is possible to achieve comparable results with fewer generations of global GP search and/or less effort on the local search is of great practical interest. Taking the best performing SSGP-SSGP-3 algorithm as an example, we conducted an experiment to further explore the balance between these factors. Typically, the CPU runtime of one local search cycle over all the trees in a final population takes $\sim$ 12 s (on a given computer), which is far longer than that of the baseline steady-state global search (SSGP) lasting $\sim$ 4 s. Thus by far the greatest proportion of the computational effort is spent on the local search process. The influence of the numbers of local search cycles on the model accuracy is presented for five representative test functions Fig. 3.

From Fig. 3, it is clear that the test error reduces with increasing numbers of local search cycles. This reduction, however, slows significantly after 2 or 3 cycles of local search. In a sense, this is very welcome since local search is so time consuming—it appears that only a little local search is needed beyond which the benefits diminish rapidly.

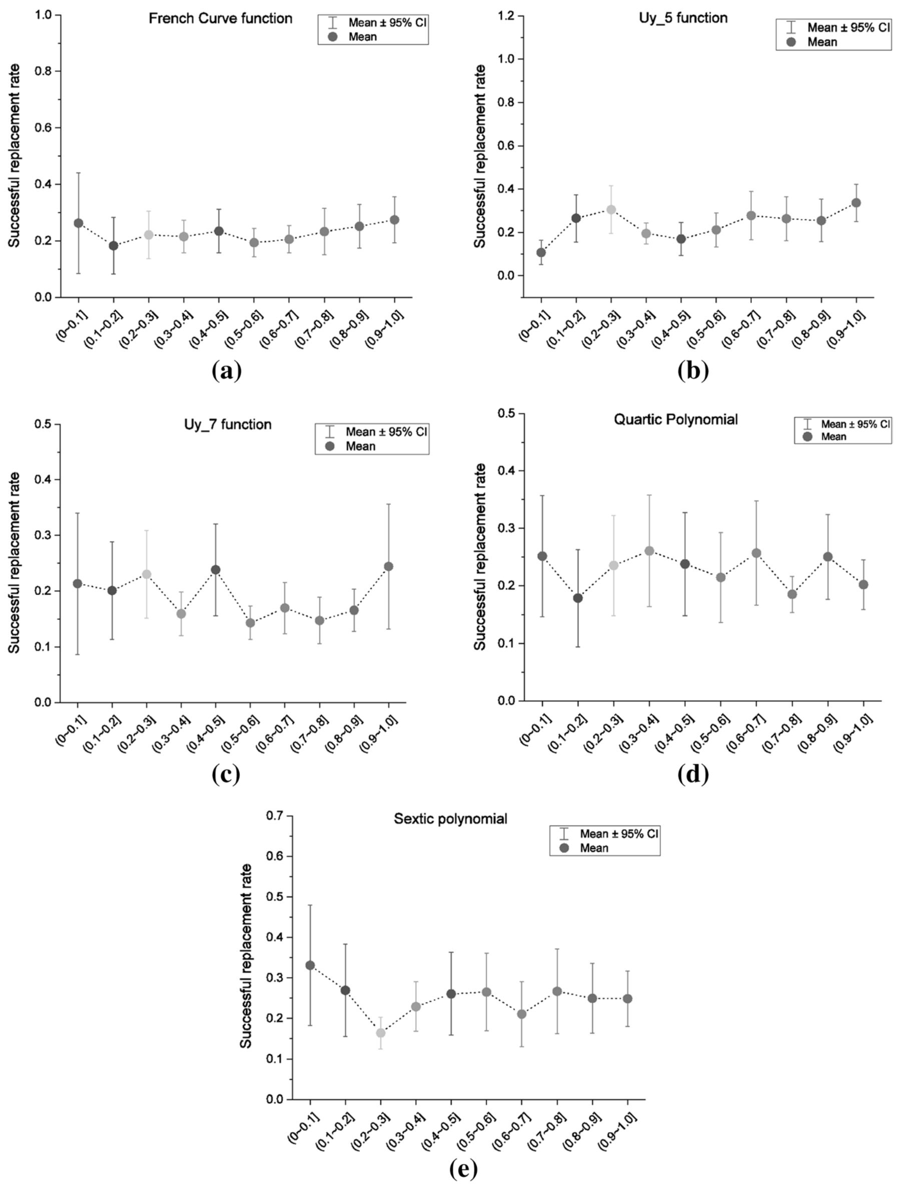


Fig. 1 Relationship between successful replacement rate with normalized subtree depth over five benchmark functions

4.6 Influence of local search on expected tree sizes

Considering the tree size comparisons in Table 6, all the evolutionary local-search methods based on either steady-state or generational global search produce trees that are either smaller or statistically the same size as the trees produced by their

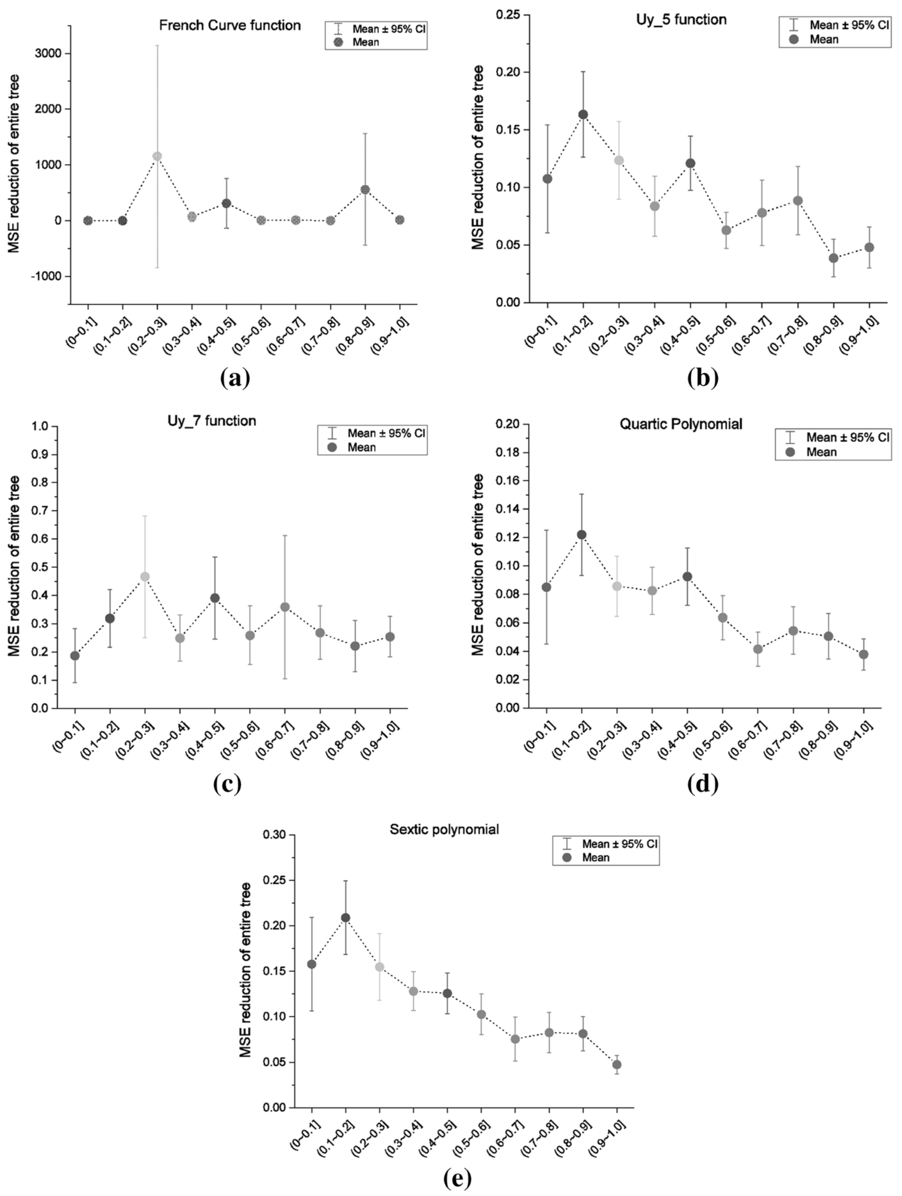
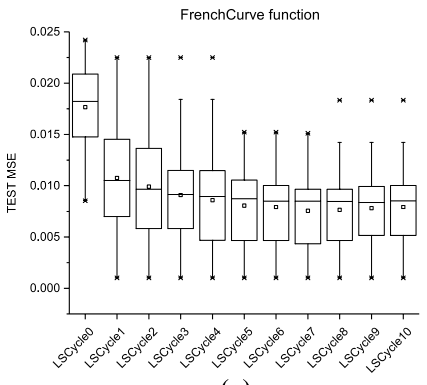


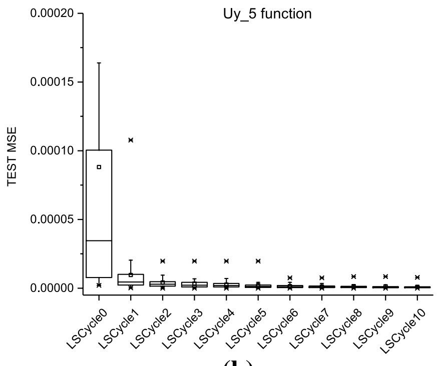
Fig. 2 Relationship between MSE reduction with normalized subtree depth over five benchmark functions

corresponding baseline algorithms. Thus, for example, local search following SSGP tends to produce smaller trees than SSGP without local search.

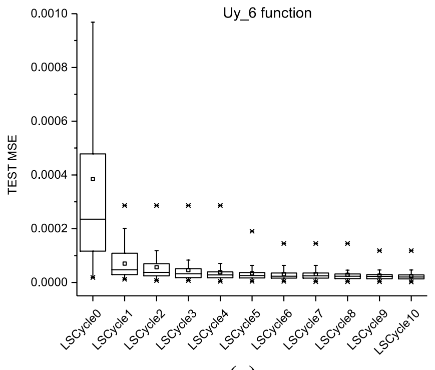
Intriguingly, the observation that local search tends to *reduce* tree sizes seems counterintuitive given that the local search methods were designed only to prevent code growth, not to produce more parsimonious structures—see Sect. 3.5.



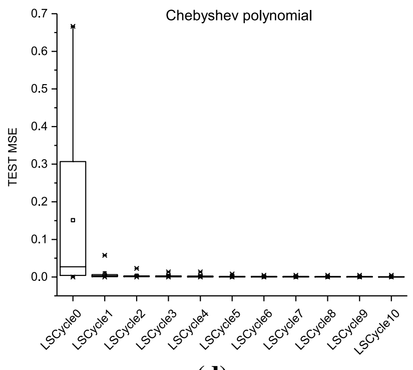
(a)



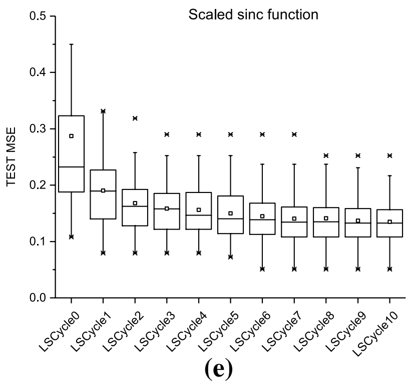
(b)



(c)



(d)



(e)

Fig. 3 Relationship between test MSE and the number of local search cycles of SSGP-SSGP-3 over five benchmark functions; the number following 'LSCycle' denotes the number of local search cycles in the local search process

For a given parent tree in the final population, we observed that local GP search almost invariably reduced the size of the tree—namely, GP local search seems effective at finding smaller trees better matched to the desired backpropagated subtree semantics. Now the final population generated by the baseline algorithm comprises an (approximation to) the Pareto set of equivalent solutions ranging from underfitted solutions with high training MSE/few nodes through to overfitted solutions with small training MSE/large numbers of nodes; as a rough rule, the solution yielding the best test MSE tends to lie around the middle of the Pareto front. Although it tends to shrink the size of the trees, we observe that local GP search rarely improves the test error of the best-performing individual produced by the baseline GP such that it continues to be the best-performing individual after local search terminates. Rather, one of the overfitted individuals tends to be modified in the local search procedure and is promoted to having a better test error than the best individual produced by the global search method. This reinforces the approach of applying local search to the *whole* of the final population of the global search algorithm rather then just the best-performing individual produced by the global algorithm. Recently, Trujillo et al. [46] have made a similar observation for local search in the context of single-objective GP. More generally, the same sort of phenomenon has been previously seen in decision trees, which are typically trained to overfitting and then heuristically pruned to improve generalization [42].

4.7 Performance of random subtree generation as a local search operator

The overall performances of the algorithms that use random tree search is variable. The SSGP-Ran-1,2,3 algorithms are all 4th ranked for MSE whereas the performances of GenGP-Ran-1,2,3 are more varied: the first two are better than SSGP, the last worse than SSGP but on a par with GenGP-GenGP-1,2 and GenGP-SSGP-1. The superior performance of the random subtree replacement algorithms that use SSGP as a global search algorithm is presumably connected to the general superiority of the steady-state strategy in global search.

Superficially, at least, there appears a similarity between local search by randomly generating replacement subtrees (the GenGP-Ran-1 ...GenGP-Ran-3 family of algorithms) and the RDO method. RDO constructs a large library of randomly-generated subtrees from which one is chosen to replace a target subtree in the parent. This generation-by-lookup table process could be viewed as an alternative way of randomly generating a subtree. GenRDO-1, however, is statistically better than random search implying this approach is not equivalent to random local search following global GP; at this point, nonetheless, we sound a note of caution about the size effect observed here. The reason for the apparent superiority of RDO-based methods is not completely clear and will be the subject of future work.

4.8 The performance of RDO as a local search operator

We have also investigated using RDO as a local search method (SS/Gen-RDO-1 to SS/Gen-RDO-6) to improve the final population generated by the baseline SS/GenGP algorithms—essentially, replacing the local search by random subtree generation with selection of replacements from an RDO-style static library. Again, superficially, these could be seen as equivalent processes. The results of using RDO

Table 7 CPU runtimes for one cycle of local search on the French curve function

Local search algorithm	GenGP-1	GenGP-2	GenGP-3	SSGP-1	SSGP-2	SSGP-3
Runtime (s)	3.483	6.942	14.219	39.150	68.942	95.990
Local search algorithm	Ran-1	Ran-2	Ran-3	RDO-1	RDO-2	RDO-3
Runtime (s)	7.994	6.749	7.112	2.841	2.558	2.539
Local search algorithm	RDO-4	RDO-5	RDO-6			
Runtime (s)	2.561	2.599	2.571			

for local search were overwhelmingly negative with little improvement in the population generated by the corresponding global SS/GenGP algorithm. We conclude, therefore, that RDO functions poorly as a local search method although clearly performs well as a genetic operator (in the generational strategy). Its superficial resemblance to a random local search operator would thus appear coincidental.

4.9 RDO compared to global GP + GP local search

One of the principal findings of this work is that using GP as a local search procedure is able to produce generalization performance that is better than the state-of-the-art GenRDO approach, and does so using trees of significantly smaller sizes; this observation applies to all of the thirteen test functions considered. To take a typical example, GenRDO-1 with the French curve function produced best test-error tree sizes in the range 29 to 723 with an average of 196. The SSGP-SSGP-3 algorithm, on the other hand, yielded trees of 23 to 467 nodes with an average of 135. We believe this results from the careful implementation of the GP local search method to avoid code growth—see Sect. 3.5.

The RDO genetic operator exhibits good search ability in a generational strategy, but with a steady-state strategy, the RDO operator performs even worse than the baseline SSGP algorithm. This implies the RDO search operator is sensitive to the evolutionary strategy. Moreover, the rapid performance deterioration from GenRDO-1 to GenRDO-3 indicates the RDO genetic operator is also sensitive to the subtree selection method. This is a disappointing characteristic of RDO since evolutionary methods are generally considered to be very robust to sub-optimal choices of parameters, etc. This robustness does not appear to extend to the RDO approach. On the contrary, GP local search appears much less sensitive than RDO to a different choice of subtree selection method. The use of Ito's method that prefers selecting target subtrees near the root node seems to encourage model generalization of the entire tree.

4.10 Computational complexity resulting from different local search strategies

We have also considered the additional computation resulting from various local search methods. Taking the French curve function as as example, and experiments run on a computer with a 3.40 GHz processor. The average CPU runtime for the baseline GenGP is around 0.68 s and for the baseline SSGP around 3.83 s. Table 7 lists the average CPU runtime cost for one cycle of refinement using various local search methods. From this table, it is clear that the most time-consuming local search is SSGP-3 that also produces the most accurate models. Local search algorithms using RDO turn out to be the least time-consuming, but provide minimal improvement to (and are sometimes worse than) the corresponding baseline GP. Clearly, the SSGP local search strategies consume more time than their corresponding GenGP methods. For both SSGP and GenGP local search, one cycle of optimization of the final population using Ito's depth fair subtree selection method takes longer than the equal-depth selection method, and the equal node subtree selection method the least. Generally, local search using random subtree generation takes about 7 s.

4.11 Future work and overall discussion

The work described in the paper has been deliberately constrained to local search methods that change the 'shapes' of GP trees by altering sub-trees. Successful local search has also been reported using methods that introduce additional 'tuning' parameters into the tree nodes—for example, [46]. An obvious area for future work is a quantitative comparison between these different approaches to local search, or indeed possible hybridization between them.

Although this paper presents a large range of algorithms, methods of local search, and their combinations, much future work remains to be done. In carrying out the work reported here, we have deliberately adopted a 'breadth first' philosophy rather than seeking detailed explanations for every observation. That said, a very clear and fertile area for future work is to revisit the promising research directions that we have identified to gain a fuller understanding of the phenomena involved; in our experience, such studies tend to be time-consuming hence we have deferred them to future work.

Another area that warrants further study is the extension to more complex, higher dimensional test functions. In the present paper, we have employed the univariate functions that tend to be regarded as "standard" within the GP community. While they represent a valid starting point for a study, these functions have received some criticism and other, more challenging datasets have been proposed in the literature [32]. An important research issue is to establish whether the advantages of local search identified in the present paper extend to higher dimensions. In addition, explicitly considering real-world datasets—which often present different challenges—would be a major extension of this work.

On the subject of test functions, one of the reviewers suggested that 'genomic'-type datasets—characterized by hundreds or thousands of features but only tens of records—would be an appropriate subject for study in the present paper; such challenging datasets have recently been addressed by Chen et al. [9] using GP. In our view, the main research issues when applying GP to 'genomic' datasets are two-fold: firstly, to constrain the complexity of a GP model to prevent overfitting when learning in what are effectively 'empty' pattern spaces, bearing in mind that one of the major advantages of GP is its ability to automatically adjust its own complexity. Secondly, genomic-type datasets are typically characterized by the presence of large numbers of uninformative/redundant features. In the context of such challenging learning problems, we think there is little reason that local search on its own would have much impact on datasets with these characteristics without also explicitly addressing the complexity constraint and feature selection challenges.

A further area that might warrant additional investigation is the mechanism of semantic back propagation that is the precursor for local search. In common with other reports, we have adopted the strategy of back propagating errors from the root node of a tree under the assumption that all of a given node's siblings possess the correct structure. Although a reasonable simplification, this would seem to significantly constrain the scope of any local search. In this context, we suggest a sensitivity-based approach [44] may improve search efficiency, and this too will be the subject of future work.

We have pointed out in Sect. 1 that memetic algorithms combine global exploratory search with local exploitative search. We believe our work fits very much within this paradigm. Since subtrees for replacement by local search are stochastically chosen, it is possible that consecutive passes of local search over a parent tree will select exactly the same target subtree leading to inefficient, duplicated search. Our use of Ito's selection strategy ('3') that tends to prefer subtrees rooted near the top of the parent will exacerbate this effect since there are fewer choices near the tree's root. We suggest improving the efficiency of our method with a tabu-like approach whereby subtrees that have been subjected to local search are not then immediately re-subjected to it in the next pass of local search. This could easily be implemented by tagging nodes with a timestamp of when they are selected as targets, and examining this timestamp before proceeding with local search; this is an area for future work.

Finally, we note that genetic programming has proved an extremely effective and practical technique for solving the combinatorial optimization problem of searching over a set of possible functions. Local search over the set of possible subtrees in a parent GP tree could thus be viewed as a recursive reduction of the overall problem. In light of this, it is perhaps logical that GP should perform well as a local search strategy.

5 Conclusions

The most significant conclusion from this paper is that semantic-based genetic programming local search is able to produce better generalization performance that is statistically different from the state of the art GenRDO-1 method of Pawlak et al. [40], and achieves this with trees of significantly smaller size. This has obvious computational implications. A contributory factor to this reduction in tree size has been the careful design of the local search procedure so as to avoid tree growth. We observe that our GP local search seems to operate by pruning overfitted trees down to the point of best test performance rather than necessarily improving the best test case individuals generated by the global SS/GenGP algorithms. Trees generated by the (SS/GenGP)-(SS/GenGP)-3 approach tend to be (statistically) smaller than those generated by the corresponding baseline algorithms, while at the same time exhibiting better prediction performance.

We have also found that the RDO operator was obviously effective when used as a genetic operator within the generational paradigm. The performance of RDO with a steady-state strategy, however, is noticeably worse than with a generational strategy. The trees generated by the steady-state variants of RDO are less accurate than even those generated by the baseline SSGP algorithm. We infer that the disruption caused by RDO search counteracts the otherwise good search performance of the steady-state strategy.

Additionally, we observed significant effects of the method for selecting the subtree for local search. On the basis of the work here, the RDO operator appears sensitive to the choice of selection operator, yielding performance that ranges from the seventh best performer (GenRDO-1) via the 12th-ranked performer (GenRDO-2) to population collapse and the lowly-ranked performer (GenRDO-3). GP local search, on the other hand, appears to display far less sensitivity to the choice of subtree selection method. The SSGP-SSGP-3 method ranks top while a less helpful choice of subtree selection method only reduces this form of GP local search to a middling (3rd or 5th ranked) performer rather than a bottom-ranked performer. The reason that the Ito's depth fair selection method performs best was investigated and it was concluded that the optimization of subtrees closer to the root are more influential in the improvement of the entire tree. Furthermore, the test error reduces with increasing numbers of cycles of local search although the gains appear modest after only two or three cycles of local search.

References

1. F. Archetti, S. Lanzeni, E. Messina, L. Vanneschi, Genetic programming for human oral bioavailability of drugs, in *8 th Annual Conference on Genetic and Evolutionary Computation (GECCO '06)* (2006), pp. 255–262. https://doi.org/10.1145/1143997.1144042
2. R.M.A. Azad, C. Ryan, A simple approach to lifetime learning in genetic programming-based symbolic regression. Evol. Comput. **22**(2), 287–317 (2014). https://doi.org/10.1162/EVCO_a_00111
3. L. Beadle, *Semantic and Structural Analysis of Genetic Programming*. Ph.D. thesis, University of Kent (2009)
4. L. Beadle, C.G. Johnson, Semantically driven mutation in genetic programming, in *11th Conference on Congress on Evolutionary Computation (CEC'09)* (2009), pp. 1336–1342
5. A. Benavoli, G. Corani, F. Mangili, Should we really use post-hoc tests based on mean-ranks? J. Mach. Learn. Res. **17**(1), 152–161 (2016)
6. M. Castelli, S. Silva, L. Vanneschi, A C++ framework for geometric semantic genetic programming. Genet. Program. Evol. Mach. **16**(1), 73–81 (2015). https://doi.org/10.1007/s10710-014-9218-0
7. M. Castelli, L. Trujillo, L. Vanneschi, Energy consumption forecasting using semantic-based genetic programming with local search optimizer. Intell. Neurosci. **2015**, 57:57 (2015). https://doi.org/10.1155/2015/971908

8. M. Castelli, L. Trujillo, L. Vanneschi, S. Silva, E. Z-Flores, P. Legrand, Geometric semantic genetic programming with local search, in *Annual Conference on Genetic and Evolutionary Computation (GECCO '15)* (Madrid, 2015), pp. 999–1006. https://doi.org/10.1145/2739480.2754795
9. Q. Chen, M. Zhang, B. Xue, Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. IEEE Trans. Evol. Comput. **21**(5), 792–806 (2017). https://doi.org/10.1109/TEVC.2017.2683489
10. V. Cherkassky, F.M. Mulier, *Learning from Data: Concepts, Theory and Methods*, 2nd edn. (Wiley-IEEE Press, New York, 2007)
11. J. Demšar, Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
12. J.J. Durillo, A.J. Nebro, F. Luna, E. Alba, On the effect of the steady-state selection scheme in multi-objective genetic algorithms, in *5th International Conference Evolutionary Multi-criterion Optimization (EMO 2009)* (Nantes, 2009), pp. 183–197. https://doi.org/10.1007/978-3-642-01020-0_18
13. R. Ffrancon, M. Schoenauer, Memetic semantic genetic programming, in *Conference on Genetic and Evolutionary Computation (GECCO '15)* (2015), pp. 1023–1030. https://doi.org/10.1145/2739480.2754697
14. C.M. Fonseca, P.J. Fleming, Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation. IEEE Trans. Syst. Man Cybern. Syst. A **28**(1), 26–37 (1998). https://doi.org/10.1109/3468.650319
15. E. Galván-López, M. O'Neill, A. Brabazon, Towards understanding the effects of locality in GP, in *8th Mexican International Conference on Artificial Intelligence* (2009), pp. 9–14. https://doi.org/10.1109/MICAI.2009.17
16. C. Giraud-Carrier, *Unifying Learning with Evolution Through Baldwinian Evolution and Lamarckism* (Springer, Dordrecht, 2002), pp. 159–168. https://doi.org/10.1007/978-94-010-0324-7_11
17. I. Gonçalves, S. Silva, C.M. Fonseca, M. Castelli, Arbitrarily close alignments in the error space: a geometric semantic genetic programming approach, in *Genetic and Evolutionary Computation Conference Companion, GECCO '16 Companion* (Denver, 2016), pp. 99–100. https://doi.org/10.1145/2908961.2908988
18. F. Gruau, D. Whitley, Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. Evol. Comput. **1**(3), 213–233 (1993). https://doi.org/10.1162/evco.1993.1.3.213
19. K. Harries, P. Smith, Exploring alternative operators and search strategies in genetic programming, in *2nd Annual Conference on Genetic Programming* (1997), pp. 147–155
20. H. Iba, H. de Garis, T. Sato, Genetic programming with local hill-climbing, in *3rd Conference on Parallel Problem Solving from Nature (PPSN III): International Conference on Evolutionary Computation* (1994), pp. 302–311. https://doi.org/10.1007/3-540-58484-6_274
21. C. Igel, K. Chellapilla, Investigating the influence of depth and degree of genotypic change on fitness in genetic programming, in *1st Annual Conference on Genetic and Evolutionary Computation (GECCO'99)*, vol. 2 (1999), pp. 1061–1068
22. T. Ito, H. Iba, S. Sato, Non-destructive depth-dependent crossover for genetic programming, in *1st European Workshop on Genetic Programming (EuroGP '98)* (London, 1998), pp. 71–82
23. D. Jackson, Promoting phenotypic diversity in genetic programming, in *11th International Conference on Parallel Problem Solving from Nature: Part II (PPSN'10)* (2010), pp. 472–481
24. P. Juárez-Smith, L. Trujillo, Integrating local search within neat-GP, in *Genetic and Evolutionary Computation Conference Companion* (Denver, 2016), pp. 993–996. https://doi.org/10.1145/2908961.2931659
25. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1 (MIT Press, Cambridge, 1992)
26. N. Krasnogor, Self generating metaheuristics in bioinformatics: the proteins structure comparison case. Genet. Program. Evol. Mach. **5**(2), 181–201 (2004). https://doi.org/10.1023/B:GENP.0000023687.41210.d7
27. K. Krawiec, Genetic programming with local improvement for visual learning from examples, in *9th International Conference on Computer Analysis of Images and Patterns (CAIP)* (2001), pp. 209–216. https://doi.org/10.1007/3-540-44692-3_26

28. K. Krawiec, P. Lichocki, Approximating geometric crossover in semantic space, in *11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)* (2009), pp. 987–994. https://doi.org/10.1145/1569901.1570036
29. W. La Cava, T. Helmuth, L. Spector, K. Danai, Genetic programming with epigenetic local search, in *Annual Conference on Genetic and Evolutionary Computation (GECCO '15)* (2015), pp. 1055–1062. https://doi.org/10.1145/2739480.2754763
30. N. Le, H.N. Xuan, A. Brabazon, T.P. Thi, Complexity measures in genetic programming learning: a brief review, in *IEEE Congress on Evolutionary Computation (CEC)* (2016), pp. 2409–2416. https://doi.org/10.1109/CEC.2016.7744087
31. H. Majeed, *The Importance of Semantic Context in Tree Based GP and Its Application in Defining a Less Destructive, Context Aware Crossover for GP*. Ph.D. thesis, University of Limerick (2007)
32. J. McDermott, D.R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, U.M. O'Reilly, Genetic programming needs better benchmarks, in *14th Conference on Genetic and Evolutionary Computation* (New York, 2012), pp. 791–798. https://doi.org/10.1145/2330163.2330273
33. N.F. McPhee, B. Ohs, T. Hutchison, Semantic building blocks in genetic programming, in *11th European Conference on Genetic Programming (EuroGP'08)* (2008), pp. 134–145
34. A. Moraglio, K. Krawiec, C.G. Johnson, Geometric semantic genetic programming, in *12th International Conference on Parallel Problem Solving from Nature—Volume Part I (PPSN'12)* (Taormina, 2012), pp. 21–31. https://doi.org/10.1007/978-3-642-32937-1_3
35. P. Moscato et al., On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech Concurrent Computation Program, C3P. Report **826**, 1989 (1989)
36. O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models* (Springer, Berlin, 2001)
37. F. Neri, C. Cotta, P. Moscato, *Handbook of Memetic Algorithms*, vol. 379 (Springer, Berlin, 2012)
38. J. Ni, R.H. Drieberg, P.I. Rockett, The use of an analytic quotient operator in genetic programming. IEEE Trans. Evol. Comput. **17**(1), 146–152 (2013). https://doi.org/10.1109/TEVC.2012.2195319
39. J. Ni, P. Rockett, Tikhonov regularization as a complexity measure in multiobjective genetic programming. IEEE Trans. Evol. Comput. **19**(2), 157–166 (2015)
40. T.P. Pawlak, B. Wieloch, K. Krawiec, Semantic backpropagation for designing search operators in genetic programming. IEEE Trans. Evol. Comput. **19**(3), 326–340 (2015). https://doi.org/10.1109/TEVC.2014.2321259
41. R. Poli, W.B. Langdon, N.F. McPhee, *A Field Guide to Genetic Programming* (Lulu Enterprises, UK Ltd., Morrisville, 2008)
42. J.R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, 1993)
43. S. Ruberto, L. Vanneschi, M. Castelli, S. Silva, ESAGP—a semantic GP framework based on alignment in the error space, in *Genetic Programming* (Berlin, Heidelberg, 2014), pp. 150–161
44. A. Saltelli, S. Tarantola, F. Campolongo, M. Ratto, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models* (Wiley, Nrew York, 2004)
45. A. Topchy, W.F. Punch, Faster genetic programming based on local gradient search of numeric leaf values, in *3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01)* (San Francisco, 2001), pp. 155–162
46. L. Trujillo, E. Z-Flores, P.S. Juarez Smith, P. Legrand, S. Silva, M. Castelli, L. Vanneschi, O. Schütze, L. Munoz, Local search is underused in genetic programming, in *Genetic Programming Theory and Practice XIV* (Springer, Ann Arbor, 2017)
47. N.Q. Uy, N.X. Hoai, M. O'Neill, B. McKay, The role of syntactic and semantic locality of crossover in genetic programming. in *11th International Conference on Parallel Problem Solving from Nature: Part II (PPSN'10)* (Krakow, 2010), pp. 533–542
48. N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. Mckay, E. Galván-López, Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet. Program. Evol. Mach. **12**(2), 91–119 (2011). https://doi.org/10.1007/s10710-010-9121-2

49. L. Vanneschi, M. Castelli, L. Manzoni, S. Silva, A new implementation of geometric semantic gp and its application to problems in pharmacokinetics, in *Genetic Programming* (Berlin, Heidelberg, 2013), pp. 205–216
50. L. Vanneschi, M. Castelli, S. Silva, A survey of semantic methods in genetic programming. Genet. Program. Evol. Mach. **15**(2), 195–214 (2014). https://doi.org/10.1007/s10710-013-9210-0
51. E.J. Vladislavleva, G.F. Smits, D. den Hertog, Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009). https://doi.org/10.1109/TEVC.2008.926486
52. G. Wahba, S. Wold, A completely automatic French curve: fitting spline functions by cross validation. Commun. Stat. **4**(1), 1–17 (1975). https://doi.org/10.1080/03610927508827223
53. P. Wang, K. Tang, E.P.K. Tsang, X. Yao, A memetic genetic programming with decision tree-based local search for classification problems, in *IEEE Congress of Evolutionary Computation (CEC)* (2011), pp. 917–924. https://doi.org/10.1109/CEC.2011.5949716
54. E. Z-Flores, L. Trujillo, O. Schütze, P. Legrand, Evaluating the effects of local search in genetic programming, in *EVOLVE—A Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation V* (Cham, 2014), pp. 213–228. https://doi.org/10.1007/978-3-319-07494-8_15
55. M. Zhang, X. Gao, W. Lou, A new crossover operator in genetic programming for object classification. IEEE Trans. Syst. Man Cybern. Syst. B **37**(5), 1332–1343 (2007)
56. M. Zhang , W. Smart, Genetic programming with gradient descent search for multiclass object classification, in *7th European Conference on Genetic Programming (EuroGP 2004)* (Coimbra, 2004), pp. 399–408. https://doi.org/10.1007/978-3-540-24650-3_38