Comparison of Semantic-aware Selection Methods in Genetic Programming

Paweł Liskowski
Institute of Computing Science
Poznan University of Technology
60965 Poznań, Poland
pliskowski@cs.put.poznan.pl

Krzysztof Krawiec
Institute of Computing Science
Poznan University of Technology
60965 Poznań, Poland
kkrawiec@cs.put.poznan.pl

Thomas Helmuth
Department of Computer Science
University of Massachusetts
Amherst, MA, 01003, USA
thelmuth@cs.umass.edu

Lee Spector
Hampshire College
Amherst, MA, 01002, USA
lspector@hampshire.edu

ABSTRACT

This study investigates the performance of several semantic-aware selection methods for genetic programming (GP). In particular, we consider methods that do not rely on complete GP semantics (i.e., a tuple of outputs produced by a program for fitness cases (tests)), but on binary outcome vectors that only state whether a given test has been passed by a program or not. This allows us to relate to test-based problems commonly considered in the domain of coevolutionary algorithms and, in prospect, to address a wider range of practical problems, in particular the problems where desired program output is unknown (e.g., evolving GP controllers). The selection methods considered in the paper include implicit fitness sharing (IFS), discovery of derived objectives (DOC), lexicase selection (LEX), as well as a hybrid of the latter two. These techniques, together with a few variants, are experimentally compared to each other and to conventional GP on a battery of discrete benchmark problems. The outcomes indicate superior performance of LEX and IFS, with some variants of DOC showing certain potential.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

Keywords

Genetic Programming; Program Semantics; Selection Operators

GECCO '15, July 11 - 15, 2015, Madrid, Spain
© 2015 ACM. ISBN 978-1-4503-3488-4/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739482.2768505

1. INTRODUCTION

One of the primary application areas of genetic programming (GP) is synthesis of programs that operate in discrete domains. In such cases, it is typically assumed that the ultimate goal of an evolutionary process is to produce a perfectly working program, i.e., a program that passes all tests (fitness cases). The degree to which candidate solutions (programs) in a population meet this goal is measured by an objective function that counts the number of passed tests. In other words, such an objective function is the *search driver* [11, 10] for an iterative, evolutionary program synthesis process.

An objective function that counts the passed tests is symmetric with respect to the tests and in this sense totally unbiased. Only the count of passed tests matters – which of them a given program passes is irrelevant. In practice however tests usually differ with respect to *difficulty*. In particular, tests may vary in *objective difficulty*, i.e., the probability of being passed by a randomly generated program. But they will often differ also when it comes to *subjective difficulty*, meant as the probability that a given program synthesis algorithm (e.g., GP), provided with certain computational resources, produces a program that passes a test. In domain of test-based problems, the distribution of objective difficulty among tests is often highly non-uniform [6]. As a result, scalar objective functions are much less effective in guiding the search algorithm towards good solutions because programs that solve k easy tests and ones that solve k difficult tests are considered equally valuable during selection.

There are several GP extensions that originated in this observation. In this study, we experimentally compare the presumably oldest method of this type, Implicit Fitness Sharing (IFS, [19]) with two approaches proposed recently: discovery of objectives by clustering (DOC, [8, 12]) and lexicase selection (LEX, [4]). Additionally, compared to [8], we consider a new variant of DOC that employs a different clustering algorithm. We present the common conceptual framework in Section 2, the compared methods in Section 3, present the results of the experiment involving 18 discrete benchmark in Section 4, to conclude with Section 5.

2. TESTS AND SEMANTICS

We assume that program synthesis task is posed by providing a set T of *tests* (*fitness cases*), where every test $(in_i, out_i) \in T$ is a pair composed of the input in to be fed into a program and the corresponding desired output out. In accordance with most past studies on semantic GP, in this paper by *program semantics* (*semantics* for short) we mean the vector (tuple) of outputs returned by a given program p for a given set of tests (fitness cases) $(in_i, out_i) \in T$, i.e.,

$$s(p) = (p(in_1), p(in_2), \ldots, p(in_{|T|})). \quad (1)$$

Semantic GP methods rely on s to, among others, diversify populations and design search operators.

The methods considered in this paper rely only indirectly on program semantics. Rather than inspecting specific output *values* produced by a program, they are interested only in whether a given test has been *passed*. We formalize this program characteristics as *outcome vector*

$$o(p) = ([p(in_1) = out_1], \ldots, [p(in_{|T|}) = out_{|T|}]), \quad (2)$$

where $[\cdot]$ is Iverson bracket, i.e.,

$$[x] = \begin{cases} 1 & if\, x\, is\, true \\ 0 & otherwise \end{cases}.$$

An outcome vector is thus a vector where ones and zeros correspond to passing and failing respective tests. The conventional objective function that counts the number of passed tests can be formalized as

$$f(p) = \sum_{i=1}^{|T|} o_i(p), \quad (3)$$

where $o_i(p)$ denotes the ith element of the outcome vector $o(p)$.

In the following, we will make use of formalism similar to $o(p)$: the *set* of tests $T(p) \subseteq T$ passed by p, i.e.,

$$T(p) = \{(in, out) \in T : p(in) = out\}.$$

Conversely, by $P(t) \subseteq P$ we will denote the set of programs in population P that pass test $t = (in, out) \in T$, i.e.,

$$P(t) = \{p \in P : p(in) = out\}.$$

3. THE METHODS

3.1 Implicit Fitness Sharing

Implicit fitness sharing (IFS) introduced by Smith et al. [19] and further explored in GP by McKay [14, 13] originates in the observation that difficulty of particular tests may vary. Although in practice problems with uniform distribution of test difficulty are less common than problems where difficulties vary by tests, the conventional fitness function (Eq. 3) is oblivious to that fact and grants the same reward of 1 for solving every test in T. This may result in premature convergence, as the programs in population naturally tend to learn how to pass the easier tests first. In order to entice a search process to solve the more difficult tests, it would be thus desirable to increase the rewards for solving them. However, where can one obtain a reliable information on test difficulty? The objective and subjective difficulty discussed in Introduction are of little use here, as estimating either of them requires running a sample of programs, and thus incurs additional computational cost.

IFS uses the outcomes of interactions between the candidate programs in the working population P and the tests in T to obtain the difficulty estimates of the latter. Those interactions have to be carried out anyway, i.e., programs in P have to be applied to the tests in T, because they are needed to assess programs' performance. In this sense, IFS comes at no additional computational cost.

Given that information, IFS defines its own evaluation function. The IFS-fitness of a program $p \in P$ fitness is defined as:

$$f_{\text{IFS}}(p) = \sum_{t \in T(p)} \frac{1}{|P(t)|}. \quad (4)$$

Note that the denominator Eq. 4 never zeroes, because if p solves a given t, then $P(t)$ must contain at least p.

The term $\frac{1}{|P(t)|}$ in (4) is IFS's measure of difficulty of test t which depends reciprocally, and thus non-linearly, on the number of programs failed by t. Importantly, IFS estimates difficulty from the working population of programs, which is biased by having evolved under a specific selection pressure.

By the token of expressing test difficulty as a ratio, tests in IFS can be likened to limited resources: individuals in the population *share* the rewards for solving them, where a reward can vary from $\frac{1}{|P(t)|}$ to 1 inclusive. Higher rewards are granted for tests that are rarely solved by population members (small $|P(t)|$), and lower for the tests solved frequently (large $|P(t)|$). However, the allocation of rewards depends on the capabilities of the current population, and as such varies with time. This is in strong contrast to conventional evolutionary algorithms, where evaluation of a candidate solution is normally *context-free*, i.e., does not depend on the other candidate solutions. IFS can be thus seen as a simple form of *coevolutionary algorithm* [18], where individuals compete for tests with each other (even though there are no direct, face-to-face interactions between individuals, usually attributed to this paradigm). In a sense, IFS performs autonomous *shaping*, because by varying the contributions for solving particular tests it modifies its own *training experience* [22].

IFS is commonly considered as a diversity maintenance technique: an individual that solves a low number of tests can still survive if its competence is rare. In this way, IFS helps reducing *crowding* and *premature convergence*. It shares this objective with *explicit fitness sharing* proposed in [3], where population diversity is encouraged by monitoring genotypic or phenotypic distances between individuals. By allowing the same program to receive different fitness in two generations of an evolutionary run, IFS may also facilitate escaping from local minima.

3.2 Discovery of Objectives by Clustering

Discovery of search objectives by clustering (DOC) introduced by Krawiec and Liskowski [9] is a method that autonomously derives new search objectives by clustering the outcomes of interactions between programs in the population and the tests. Each derived objective is intended to capture a subset of 'capabilities' exhibited by the programs in the context of other individuals in population. The derived objectives replace the conventional fitness function (Eq. 3) and are subsequently used to drive the selection process in a single- or multiobjective fashion. DOC is inspired by our

Algorithm 1 Discovery of Objectives by Clustering (DOC)

Input: Population P and set of tests T.
Output: Derived interaction matrix G'.

1. Calculate the $m \times n$ interaction matrix G between the programs from the current population P, $|P| = m$, and the tests from T, $|T| = n$.
2. Cluster the tests. Every column of G, i.e., the vector of interaction outcomes of all programs from P with a test t, are treated as a point in an m-dimensional space. A clustering algorithm of choice is applied to the n points obtained in this way. The outcome of this step is a partition $\{T_1, \ldots, T_k\}$ of the original n tests in T into k subsets/clusters, where $1 \leq k \leq n$ and $T_j \neq \emptyset$.
3. Define the derived objectives. For each cluster T_j, we average row-wise the corresponding columns in G. This results in an $m \times k$ *derived interaction matrix* G', with the elements defined as follows:
$$g'_{i,j} = \frac{1}{|T_j|} \sum_{k=1}^{|T_j|} o_k(p_i) \tag{5}$$
where p_i is the program corresponding to the ith row of G, and $j = 1, \ldots, k$.

previous work in coevolutionary algorithms, and builds upon the approach we designed for test-based problems in [12].

DOC requires that the outputs returned by a each program $p \in P$ for a given set of tests $(in_i, out_i) \in T$ are gathered in an *interaction matrix* G. For a population of m programs and a set of n tests, G is an $m \times n$ matrix where $g_{ij} = [p_i(in_j) = out_j]$. The evaluation phase in the GP algorithm incorporating DOC is summarized in Algorithm 1.

The columns of G' implicitly define the k *derived objectives* that characterize the programs in P and form the basis for selecting the most promising programs from P. In a simplest case, they can be employed as objectives in conventional multiobjective evolutionary methods, such as the NSGA-II [2].

Derived objectives $d_1, d_2, \ldots, d_k \in D$ can be also used to drive the selection in a single-objective manner using for this purpose the hypervolume of program's performance, i.e.,
$$h(p) = \prod_{i=1}^{k} d_i(p). \tag{6}$$
The use of the hypervolume of a program's performance as a search objective can be viewed as a method that promotes balanced performance on all of the derived objectives.

DOC is designed to be sensitive to inherent difficulty of tests by avoiding the problem of compensation deeply rooted in aggregating fitness functions (cf. Section 1). For this purpose, DOC transforms a single objective-problem given by the original objective function into a multi-objective one to facilitate the use of *dominance relation*:
$$p_1 \succ_D p_2 \iff \forall_{d \in D} : d(p_1) \geq d(p_2) \wedge \exists_{d \in D} : d(p_1) > d(p_2).$$
The derived objectives form a multi-objective characterization of the candidate solutions in the context of the current population of test, yet they do not guarantee to preserve the original dominance relation defined in the space of tests. As a result of clustering, some information about the dominance structure can be lost, however this inconsistency brings a critical advantage: the number of resulting derived objectives is low, so that together they are able to impose an effective search gradient on the evolving population. Furthermore, by implementing selection in the space of derived objectives, candidate solutions that feature different 'skills' can coexist in population even if some of them are clearly better than others in terms of number of tests they pass.

Algorithm 2 Lexicase Selection (LEX)

Input: Population P and set of tests T.
Output: The selected individual.

1. Let $P' \leftarrow P$ and $T' \leftarrow T$.
2. Draw at random a test $t \in T'$.
3. Set $P' \leftarrow P' \cap P(t)$ and $T' \leftarrow T' \setminus \{t\}$.
4. If $|P'| > 1$ and $|T'| > 1$ go to step 2.
5. If $|P'| = 1$, return the only element of P'; otherwise return a randomly selected element in P'.

3.3 Lexicase Selection

Lexicase selection (LEX in the following) is a semantic-aware selection technique recently introduced by Spector [20]. Recent experiments by Helmuth et al. [4] demonstrate LEX's strengths in problem solving and diversity maintenance.

A single act of applying LEX to a given population P under the set of tests T, phrased using the formalisms provided in Section 2, proceeds as in Algorithm 2.

Every selection act (resulting with a single program) requires an independent run of the above algorithm. In the worst case, the loop comprising the lines 2–4 of the algorithm needs to iterate over all tests. Similarly to DOC, in addition to rewarding the programs for solving test cases, lexicase selection promotes diversified programs that pass randomly selected subset of tests. In this way, different tests are emphasized in each selection event. An individual that passes test(s) that are rarely passed by its competitors has substantial chance to propagate to the next generation even if it performs poorly on many other test. For instance, if an single individual alone has the best error for a particular case then that individual will be selected whenever that case comes up first in the random ordering, regardless of its error on other cases. This is an extreme situation — often it won't be just a single individual, and/or sets of cases will be relevant.

Note that in contrast to IFS and DOC, LEX does not explicitly define any objectives or alternative fitness functions. In this sense, it is 'natively' a selection method.

3.4 Hybridizing DOC with LEX

In its original form, the derived objectives identified by DOC are based on disjoint subsets of tests and drive the selection process in either single- or multi-objective fashion. In earlier studies, we typically combined it with NSGA-II [2], to avoid aggregation of interaction outcomes with all tests

Algorithm 3 DOCLEX selection.

Input: Population P and set of derived objectives D.
Output: The selected individual.

```
1. Let $P' \leftarrow P$ and $D' \leftarrow D$.
2. Draw at random a derived objective $d \in D'$.
   Let $P_d \subseteq P'$ be the subset of programs that perform the best on $d$.
3. Set $P' \leftarrow P_d$ and $D' \leftarrow D' \setminus \{d\}$.
4. If $|P'| > 1$ and $|D'| > 1$ go to step 2.
5. If $|P'| = 1$, return the only element of $P'$; otherwise return a randomly selected element in $P'$.
```

into single scalar value, which is characteristic for the traditional objective function discussed in 1. One of the main motivations for lexicase selection was also to avoid such aggregation, and the decisions based in particular iterations of Algorithm 2 are based on distinct tests. These observations encourage combining both methods into a hybrid approach in which lexicase selection is performed on the derived objectives.

Technically, we first derive new search objectives using Algorithm 1. Subsequently, we apply Algorithm 2, using the particular derived objectives d_i as if they were test cases in lexicase selection. In each iteration, a derived objective is drawn at random. Then, individuals in the population that do not achieve the best performance on that objective are eliminated. If more than one individual remains, the process repeats, eliminating any remaining individuals that do not achieve the best performance on the next derived objective drawn at random. This process continues until only one individual remains and is selected, or until all derived objectives have been processed, in which case a random program is selected from the remaining programs remaining.

The complete process is shown in Algorithm 3. In the following, we refer to it as DOCLEX.

4. EXPERIMENTS

We examine the performance of sematic-aware selection methods in the domain of tree-based GP. The compared algorithms implement generational evolutionary algorithm and vary only in the selection procedure. Otherwise, they share the same parameter settings, with initial population filled with the ramped half-and-half operator, subtree-replacing mutation engaged with probability 0.1 and subtree-swapping crossover engaged with probability 0.9. We run experiments lasting up to 200 generations with population size $|P| = 1000$. The search process stops when the assumed number of generation elapses or an ideal program is found; the latter case is considered a success.

Compared algorithms. We begin by describing the variants of DOC. Following our previous study, we employ X-MEANS [16] to group similar tests in the basic variant of DOC (line 2 of Algorithm 1). X-MEANS is an extension of the popular k-means algorithm that autonomously picks a number of clusters k that leads to clustering maximizing the Bayesian Information Criterion. In the experiments, we allow X-MEANS consider $k \in [1, 5]$ and employ the Euclidean

Algorithm 4 Estimating number of derived objectives using Gap Statistics

Input: Interaction matrix G, max number of objectives K.
Output: Expected number of objectives k.

```
1. For $k = 1, \ldots, K$ do:
   (a) Run a $k$-means on G to find initial clusters $T_1, \ldots, T_k$. Compute the dispersion:
       $$W_k = \sum_{r=1}^{k} \frac{1}{2|T_r|} D_r,$$
       where $D_r = \sum_{i,j \in C_r} d_{ij}$ is the sum of pairwise distances for all points in cluster $r$.
   (b) Generate a set of reference distributions by sampling uniformly from $G$'s bounding rectangle.
   (c) Define $k-$th gap:
       $$Gap_k = \mathbf{E}[log(\hat{W}_k)] - log(W_k)$$
       where $\mathbf{E}[log(\hat{W}_k)]$ denotes expected dispersion of the reference distributions.
2. Select the number of objectives to be the one that gives the maximum gap, i.e., $k = \arg\max_k Gap_k$.
```

metric to measure the distances between the observations (the columns of G). DOC employs NSGA-II to perform multiobjective selection in the space of derived objectives.

Another variant of DOC, DOCP, uses the derived objectives as dimensions in a hypervolume of program's performance and multiplies them together to obtain scalar fitness (Eq. 6). Fitness values calculated in this way are subsequently used in a tournament selection scheme with tournament size of 7.

We also consider two variants of DOC that incorporate other approaches to automatic selection of k. First of them, DOCGAP, employs the *gap statistics* [23]. Given clustering $C_1, \ldots, C_k$ and within-cluster sum of squares W_k around the cluster means, the basic principle behind the approach is to standardize the graph of $log(W_k)$ by comparing it with its expectation under an appropriate null reference distribution of the data. See Algorithm. 4 for complete description of the method.

Although gap statistics has been shown to find the optimal number of clusters in variety of cases, its performance might be hampered by the need of performing Monte Carlo simulations to estimate the reference datasets. In [17] Pham et al. address this issue and propose to estimate the clustering performance on the uniformly distributed reference dataset. Furthermore, the measure $f(k)$ given below is used to evaluate the clustering result:

$$f(k) = \begin{cases} 1 & if\, k = 1 \\ \frac{W_k}{\alpha_k W_{k-1}} & if\, W_{k-1} \neq 0, \forall k > 1 \\ 1 & if\, W_{k-1} = 0, \forall k > 1 \end{cases}$$

$$\alpha_k = \begin{cases} 1 - \frac{3}{4|P|} & if\, k = 2 \wedge |P| > 1 \\ \alpha_{k-1} + \frac{1-\alpha_{k-1}}{6} & if\, k > 2 \wedge |P| > 1 \end{cases}$$

where $\alpha_k \in [0,1]$ is the weight factor used to reduce the effect of high data dimensionality. Areas of concentration in the data distribution cause the ratio of real distortion to the estimated distortion to decrease and as such values of k that yield small $f(k)$ can be regarded as giving well-defined clusters. DOCFK employs $f(k)$ to evaluate clustering result and choose the optimal k for the given data.

Apart from these variants of DOC, we run the conventional Koza-style GP (GP in the following), which employs tournament of size 7 in the selection phase, implicit fitness sharing (IFS, Section 3.1) also with tournament of size 7, lexicase selection (LEX) [4] (Section 3.3) and DOCLEX, the hybrid of DOC and LEX proposed in Section 3.4.

Benchmark problems. In this study, we limit our interest to problems with discrete interaction outcomes since both DOC and LEX were designed with such problems in mind. The first group are Boolean benchmarks, which employ instruction set {*and, nand, or, nor*} and are defined as follows. For an v-bit comparator *Cmp* v, a program is required to return *true* if the $\frac{v}{2}$ least significant input bits encode a number that is smaller than the number represented by the $\frac{v}{2}$ most significant bits. In case of the majority *Maj* v problems, *true* should be returned if more that half of the input variables are *true*. For the multiplexer *Mulv,* the state of the addressed input should be returned (6-bit multiplexer uses two inputs to address the remaining four inputs). In the parity *Parv* problems, *true* should be returned only for an odd number of *true* inputs.

The second group of benchmarks are the algebra problems originating from Spector *et al.*'s work on evolving algebraic terms [21]. These problems dwell in a ternary domain: the admissible values of program inputs and outputs are $\{0,1,2\}$. The peculiarity of these problems consists of using only one binary instruction in the programming language, which defines the underlying algebra. For instance, for the a_1 algebra, the semantics of that instruction is defined as follows:

a_1	0	1	2
0	2	1	2
1	1	0	0
2	0	0	1

In the following, the employed algebra is indicated by the number in suffix part of the evolved term's name. See also [21] for the definitions of the remaining four algebras. For each of the five algebras considered here, we consider two tasks (of four discussed in [21]). In the *discriminator term* tasks (*Disc* in the following), the goal is to synthesize an expression that accepts three inputs x, y, z and is semantically equivalent to the one shown below:

$$t^A(x,y,z) = \begin{cases} x & if\, x \neq y \\ z & if\, x = y \end{cases} \tag{7}$$

There are thus $3^3 = 27$ fitness cases in these benchmarks. The second tasks (*Malcev*), consists in evolving a so-called *Mal'cev term*, i.e., a ternary term that satisfies the equation:

$$m(x,x,y) = m(y,x,x) = y \tag{8}$$

This condition specifies the desired program output only for some combinations of inputs: the desired value for $m(x,y,z)$, where x, y, and z are all distinct, is not determined. As a result, there are only 15 fitness cases in our *Malcev* tasks, the lowest of all considered benchmarks. The motivation for the discriminator and Mal'cev term problems is originally that they're of interest to mathematicians [1]. In this paper, however, we chose them as benchmarks because of their difficulty and formal elegance.

Performance. Table 1 reports the success rates of particular algorithms, resulting from 50 runs of each configuration on every benchmark. To provide an aggregated perspective on performance, we employ the Friedman's test for multiple achievements of multiple subjects [7].

Friedman's test operates on average ranks, which for the considered methods are as follows:

LEX	IFS	DOCP	DOCFK	DOCGAP	DOCLEX	DOC	GP
2.05	2.94	3.58	4.27	**4.80**	**5.36**	**5.94**	**7.02**

The p-value for Friedman test is $\ll 0.001$, which strongly indicates that at least one method performs significantly different from the remaining ones. We conducted post-hoc analysis using symmetry test [5]: bold font marks the methods that are outranked at 0.05 significance level by the *first* method in the ranking.

Program size. The ranking of configurations according to the size of the best-of-run program is as follows (the smaller programs, the lower rank):

LEX	DOCGAP	DOCLEX	DOCP	DOC	IFS	DOCFK	GP
1.67	2.89	2.92	3.31	3.78	**3.95**	**4.12**	**5.39**

The best-of-run programs produced by LEX turn out to be the smallest on average. To an extent this was expected: LEX achieves the best success rate, and the runs of this method last for the lowest number of generations on average (we do not report the average run length for brevity). As programs in GP tend to bloat with time, the best programs found in the early stages of evolution are likely to be smaller than their counterparts in the later stages.

However, IFS, which fared almost as well as LEX on success rate, turns out to produce much larger programs, although its average run lengths (not reported here) do not diverge much from those of LEX. Apparently, some other aspect must be involved here that causes LEX to yield succinct solutions. Interestingly, the only other method involving lexicase algorithm, DOCLEX, is the runner-up in this ranking. This may suggest that the particular way the lexicase algorithm selects candidate programs happens to promote more compact solutions. We have no sound hypothesis that would hint to possible explanations of this phenomenon, and intend to investigate it in further studies.

5. DISCUSSION

We interpret the high performance of LEX and IFS primarily as a strong indicator of the importance of diversification. In LEX, every test has the same chance of becoming a decisive factor in selection and, in consequence, programs that pass different tests can coexist in an evolving population. In particular, even if a program passes few tests (compared to other programs in a population), it has substantial chance to be selected if the tests it passes are rarely passed by competitors in population. IFS is similar to LEX in this respect.

On the other hand, we hypothesize that LEX is also quite resistant to overspecialization (also known as *focusing* in

Table 1: Success rate and average program size of best-of-run individuals aggregated over 50 evolutionary runs. In case of DISC and MALCEV benchmarks, suffix indicates the underlying algebra. Bold marks the best result for each benchmark.

	Success rate (percent)								Average program size [nodes]							
	GP	IFS	DOC	DOCGAP	DOCFK	DOCP	LEX	DOCLEX	GP	IFS	DOC	DOCGAP	DOCFK	DOCP	LEX	DOCLEX
Cmp6	36	**100**	54	84	**100**	98	**100**	**100**	215	175	146	175	201	174	**128**	132
Cmp8	0	78	0	0	0	34	**100**	50	272	335	**77**	94	166	264	292	317
Maj6	51	**100**	22	12	98	**100**	**100**	**100**	226	213	192	178	239	215	**159**	201
Maj8	0	45	0	0	0	0	80	0	451	492	**102**	109	231	280	499	364
Mux6	98	**100**	96	98	**100**	**100**	**100**	**100**	116	87	121	113	118	93	**68**	73
Mux11	45	**100**	0	0	0	88	**100**	76	291	286	**107**	143	186	244	216	279
Par5	4	38	2	0	8	6	92	4	327	365	251	**189**	310	316	335	329
Par6	0	0	0	0	0	0	0	0	374	329	169	**166**	268	235	322	305
disc-1	0	74	48	44	62	74	**100**	34	**13**	140	153	154	152	136	137	129
disc-2	2	74	50	72	62	94	**100**	30	**64**	153	147	163	140	146	120	126
disc-3	22	98	96	**100**	98	94	**100**	96	129	125	147	156	136	112	**83**	127
disc-4	0	38	14	16	6	32	96	2	**3**	172	170	160	155	180	182	65
disc-5	0	84	70	62	72	94	**100**	52	**8**	120	139	157	132	139	108	127
malcev-1	88	**100**	96	**100**	**100**	**100**	**100**	82	111	87	104	111	112	96	**76**	103
malcev-2	58	98	88	**100**	**100**	**100**	**100**	80	100	96	106	106	94	87	**69**	96
malcev-3	78	**100**	98	**100**	**100**	**100**	**100**	**100**	143	98	122	117	126	102	**69**	94
malcev-4	14	86	74	**100**	98	86	98	18	71	90	131	111	129	109	87	**53**
malcev-5	92	**100**	**100**	**100**	98	**100**	**100**	96	82	65	70	82	66	63	**52**	63

coevolutionary algorithms). A highly specialized 'outlier' program p with a unique capability of passing a specific test t is not certain to survive, because the *a priori* chance that a selection act involves t is relatively low. For this to happen, t needs to be drawn in the early iterations of the lexicase loop (steps 2-4 in Algorithm 2): if an earlier iteration picks a test failed by p, p is discarded from the working set P'. IFS, to the contrary, is very eager in supporting outliers: being the only program that passes a given test in a population P results in a $|P|$-times higher positive contribution to fitness than passing a test that all programs in P pass (Eq. 4). For the population size of 1000 used in this study, these rewards differ by three orders of magnitude.

All in all, LEX promotes diversification of skills in population while preventing excessive specialization. To be selected, a program needs to be also robust, i.e., successfully compete with other programs on various random sequences of tests considered in the lexicase loop.

The performance of DOC's hybrid with LEX (DOCLEX), the novel contribution of this paper, is substantially worse at the moment. Our working explanation is that LEX is likely to perform well when the number of tests is relatively large, and this is the case when it is applied directly to the benchmarks in this study. DOC, on the other hand, rarely produces more than a few derived objectives. When forced to work with just a handful of derived objectives, lexicase selection may find it difficult to diversify the population well enough. We plan to address this issue in the further research.

Of all the methods that work with derived objectives, we observe the best performance for DOCP, the variant of DOC that naively aggregates the derived objectives into a single scalar value, i.e., hypervolume. This corroborates our result obtained earlier in [9]. We attribute this regularity to the fact that, with all the shortcoming resulting from aggregation, scalar fitness makes it possible to enforce sufficiently strong selection pressure on an evolving population (controlled in our case by the size of tournament). All the remaining variants of DOC interpret the derived objectives in multiobjective fashion, and the strength of the resulting selection pressure depends in part on the number of derived objectives. In general, the more objectives, the weaker the pressure: higher number of objectives makes it more likely for two candidate programs to be mutually non-dominated, and thus occupy the same Pareto rank.

6. CONCLUSIONS

In this paper, we explored and compared several selection methods that, to a greater or lesser extent, avoid aggregation of outcomes of interactions between programs and tests (fitness cases). All these techniques performed on average better than the baseline GP equipped with tournament selection based on the conventional scalar objective function that simply counts the number of passed tests. We take it as an evidence that methods that widen the 'evaluation bottleneck' [11] are worth pursuing. A selection method that is better informed about the detailed characteristics of candidate solutions may help maintaining diversified skills in an evolving population while still imposing efficient gradient on it.

To operate, the methods considered in this paper need only outcome vectors of programs in population (Eq. 2). The exact knowledge of program semantics (Eq. 1) not necessary (though one might conceivably come up with some extensions to take that information into account). On one hand, this deprives these methods of more detailed information on program behavior, which could be potentially exploited to make search process more efficient, as practiced in typical approaches to semantic GP, in particular geometric semantic GP [15]. On the other hand, outcome vectors allow embracing a broader class of problems in which the desired semantics (i.e., the semantics of the sought program) is not known. For instance in typical control problems, the optimal control signal is not given explicitly. The performance of a controller can be assessed only by running it in a given environment (corresponding to a test in this paper). The result of such interaction, expressed in terms of domain-specific performance measure, determines one element of an outcome vector. Semantic-aware methods that rely on outcome vectors allow thus reach out beyond the typical semantic GP setting and relax the original formulation of program synthesis task.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1017817, 1129139, and 1331283. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. P. Liskowski acknowledges support from grant no. DEC-2013/09/D/ST6/03932.K. Krawiec acknowledges support from National Research Centre grant # 2014/15/B/ST6/05205

7. REFERENCES

[1] D. M. Clark. Evolution of algebraic terms 1: term to term operation continuity. *International Journal of Algebra and Computation*, 23(05):1175–1205, 2013.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[3] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-wesley, 1989.

[4] T. Helmuth, L. Spector, and J. Matheson. Solving uncompromising problems with lexicase selection.

[5] M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric statistical methods*, volume 751. John Wiley & Sons, 2013.

[6] W. Jaśkowski, P. Liskowski, M. Szubert, and K. Krawiec. Improving coevolution by random sampling. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1141–1148. ACM, 2013.

[7] G. K. Kanji. *100 statistical tests.* Sage, 2006.

[8] K. Krawiec and P. Liskowski. Automatic derivation of search objectives for test-based genetic programming. In *Genetic Programming*, pages 53–65. Springer, 2015.

[9] K. Krawiec and P. Liskowski. Automatic derivation of search objectives for test-based genetic programming. In P. Machado, M. Heywood, and J. McDermott, editors, *18th European Conference on Genetic Programming*, LNCS, Copenhagen, 8-10 Apr. 2015. Springer. Forthcoming.

[10] K. Krawiec and U.-M. O'Reilly. Behavioral programming: a broader and more detailed take on semantic GP. In C. Igel, editor, *GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 935–942, Vancouver, BC, Canada, 12-16 July 2014. ACM. Best paper.

[11] K. Krawiec and U.-M. O'Reilly. Behavioral search drivers for genetic programing. In M. Nicolau, editor, *17th European Conference on Genetic Programming*, volume 8599 of *LNCS*, pages 210–221, Granada, Spain, 23-25 Apr. 2014. Springer.

[12] P. Liskowski and K. Krawiec. Discovery of implicit objectives by compression of interaction matrix in test-based problems. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 611–620. Springer, 2014.

[13] R. McKay. Committee learning of partial functions in fitness-shared genetic programming. In *Industrial Electronics Society, 2000. IECON 2000. 26th Annual Confjerence of the IEEE*, volume 4, pages 2861–2866. IEEE, 2000.

[14] R. I. McKay. Fitness sharing in genetic programming. In *GECCO*, pages 435–442, 2000.

[15] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In C. A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31, Taormina, Italy, Sept. 1-5 2012. Springer.

[16] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[17] D. T. Pham, S. S. Dimov, and C. Nguyen. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.

[18] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. de Jong. *Handbook of Natural Computing*, chapter Coevolutionary Principles. Springer-Verlag, 2011.

[19] R. E. Smith, S. Forrest, and A. S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary computation*, 1(2):127–149, 1993.

[20] L. Spector. Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 401–408. ACM, 2012.

[21] L. Spector, D. M. Clark, I. Lindsay, B. Barr, and J. Klein. Genetic programming for finite algebras. In M. Keijzer, editor, *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1291–1298, Atlanta, GA, USA, 12-16 July 2008. ACM.

[22] M. Szubert. *Coevolutionary Shaping for Reinforcement Learning.* PhD thesis, Poznan University of Technology, 2014.

[23] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.