Genetic Programming with Alternative Search Drivers for Detection of Retinal Blood Vessels

Krzysztof Krawiec[1(✉)] and Mikołaj Pawlak[2]

[1] Poznan University of Technology, Poznań, Poland
krawiec@cs.put.poznan.pl
[2] Poznan University of Medical Sciences, Poznań, Poland
mpawlak@ump.edu.pl

Abstract. A classification task is a test-based problem, with examples corresponding to tests. A correct classification is equivalent to passing a test, while incorrect to failing it. This applies also to classifying pixels in an image, viz. image segmentation. A natural performance indicator in such a setting is the accuracy of classification, i.e., the fraction of passed tests. When solving a classification tasks with genetic programming, it is thus common to employ this indicator as a fitness function. However, recent developments in GP as well as some earlier work suggest that the quality of evolved solutions may benefit from using other search drivers to guide the traversal of the space of programs. In this study, we systematically verify the usefulness of selected alternative search drivers in the problem of detection of blood vessels in ophthalmology imaging.

Keywords: Genetic programming · Search drivers · Binary classification · Image segmentation

1 Introduction

Genetic programming (GP) is a branch of evolutionary computation (EC) devoted to synthesis of programs and other executable structures. Programs are discrete, variable-length structures, and as such require specialized search operators. In this perspective, program synthesis tasks approached in GP are considered as a specific subclass of optimization problems.

There are however other aspects that make GP quite unique, one of them being the class of objective functions employed therein. Typically, they assess programs by running them on a set of *tests* (fitness cases). A test is a pair composed of program input and the corresponding desired output. If a program provided with the input returns an output that is close enough to the desired output, it is said to *pass* the test; otherwise, it *fails* it. The objective function aggregates the outcomes of program's application to particular tests, usually treating every test in the same way. For discrete domains, this may boil to simply counting the number of passed tests. For continuous outputs, the errors committed on particular tests are aggregated via mean square error or mean

A.M. Mora and G. Squillero (Eds.): EvoApplications 2015, LNCS 9028, pp. 554–566, 2015.
DOI: 10.1007/978-3-319-16549-3_45

absolute deviation, which corresponds to measuring the Euclidean or city-block distance between the vector of actual outputs and the vector of desired outputs.

This conventional formulation leads to a scalar objective function and so causes the program synthesis problem subscribe to the classical optimization paradigm, which is often considered as an asset. We argue however that there is also certain price to pay. The search algorithm has no access to program's performance on particular tests: it knows only its aggregated characteristics, averaged over the entire training set of tests. In effect, the outcomes of particular tests may compensate, so that two programs with completely different capabilities may receive the same fitness. Because in most of real-world problems some tests are easier than others, this entices evolution to focus on passing the easier tests, while neglecting the more difficult. Such overfocusing on a subset of tests is likely to lead to premature convergence. Also, postponing learning how to solve the harder tests may not be the best strategy for finding an optimal solution.

The risks incurred by relying on conventional scalar evaluation measures apply also to evolutionary image analysis. The problems solved in this domain are by definition test-based, with tests corresponding to entire images (image classification), objects in scenes (object recognition), and even single pixels (segmentation). By often engaging large numbers of information-rich tests, they particularly call for better ways of exploiting the outcomes of interactions between the evolving programs and tests.

Several approaches have been proposed in the past to address this issue. Typically, they boil down to replacing the conventional fitness with an alternative *search driver*. The objective of this study is to systematically assess the usefulness of selected search drivers on the real-world image analysis task, and so possibly arrive at certain recommendations concerning 'good practices' in this domain. More specifically, after shortly presenting the background in Sect. 2, we apply the methods reviewed in Sect. 3 to the task of segmentation of ophthalmological imagery (Sect. 4). In the experiment conducted in Sect. 5, we combine each alternative search driver with discrete and continuous feature definitions, and assess the generalization performance. Apart from this systematic analysis, the original contribution of this work is a new variant of implicit fitness sharing [1], capable of handling continuous program outputs.

2 Background

One of the most common applications of genetic programming in image analysis is image segmentation. In this study, we are interested in delineating a single type of anatomical structure in an image (blood vessels), viz. separating such structures from the background. In other words, the objective is to *segment out* a single category of objects (pixels in this case). Therefore, we will pose this task as follows: given an image, classify every pixel in that image to one of the two decision classes: positive or negative.

It is assumed that the decision on pixel's class can be made based on its neighborhood. A GP classifier will thus have access on an $m \times m$ square window

(region of interest, ROI) for defining image features. By sliding a ROI over the entire image and applying a GP system to it, all pixels in an image can be classified.

The data gathered from different positions of the sliding window in a training image naturally forms a GP task. Each ROI position in a training image gives rise to a single test (fitness case). More formally, each test is a pair $(\mathbf{x}, y)$, where $\mathbf{x}$ is a vector of elementary features extracted from the ROI, and y is the desired outcome of applying a GP classifier to $\mathbf{x}$. An interaction between a GP program and a test $(\mathbf{x}, y)$ boils down to feeding $\mathbf{x}$ into the program, running the program, and comparing its output $\hat{y}$ with the desired output y.

An important characteristics of applying GP to binary image segmentation in the above manner is that GP programs are supposed to operate as classifiers. Because the desired output y may take on only two values, it may be tempting to pose this task in the Boolean domain. However, program input, by being gathered from a raster image, is by nature composed of continuous image features. To fully exploit the data available therein, it may be more appropriate to rely on instructions that operate in the continuous domain. In other words, posing this problem as a regression task is more natural given the nature of the input data.

The tension between the input characteristics that votes in favor of posing this problem as a regression task, and the desired output being a binary variable is the central topic of the study. It is not obvious what is the 'right' way of assessing the discrepancy between the desired output y and the actual output $\hat{y}$, i.e., what is the right *search driver* to guide the GP search process.

3 The Methods

In this section, we review the search drivers used in the subsequent experiment and propose a novel variant of one of them. We limit our attention exclusively to scalar search drivers.

Let y_i denote the desired output for the ith test, and $\hat{y}_i$ denote the actual program output. When approaching regression tasks, y_i's can be arbitrary real values. In this study however, we are solving binary *classification* tasks, and we assume that $y_i \in \{-1, 1\}$, with -1 denoting the negative decision class and 1 corresponding to the positive class. Regardless of that, we will equip our GP systems with an instruction set typically used for regression, and interpret the continuous output of a GP tree as an indication of decision class. The interpretation of program output, i.e., how its divergence from the desired output impacts program's fitness, will vary depending on search driver. For clarity, we define all search drivers as measures that are to be minimized.

L1. Our first search driver is the total absolute error of $\hat{y}$ with respect to y:

$$f_{L1} = \sum_i |\hat{y}_i - y_i|.$$

Many selection methods common in GP, like tournament selection, interpret fitness as an ordinal (non-metric) variable, and are thus insensitive to the

absolute values. Under this assumption, L1 is equivalent to the mean absolute deviation (MAD) and city-block distance.

L2. Analogously to L1, we define f_{L2}, which is the total *square* error of $\hat{y}$ with respect to y, i.e.,

$$f_{L2} = \sum_i (\hat{y}_i - y_i)^2.$$

Similarly to f_{L1}, f_{L2} is equivalent, up to ordering, to the Euclidean distance and means square error (MSE).

L1 and L2 are designed for continuous spaces and as such penalize for *any* divergence from the desired values. Because we are interested here in solving binary classification tasks, and -1 and 1 are the only desired output values possible, these search drivers seem overly restrictive. Intuitively, one should allow a program express its decision in a more general way, for instance by the sign of the output value. This observation leads to the next search driver.

Hamming distance. The next search driver counts the number of tests for which program output has the same sign as the desired value, i.e.,

$$f_H = \sum_i \begin{cases} 1 & if\ y_i \hat{y}_i > 0 \\ 0 & otherwise \end{cases}.$$

This search driver is equivalent to the Hamming distance between the vector of desired outputs and the binarized vector of program outputs. It is also equivalent to classification error (meant as the complement of classification accuracy used in machine learning).

Hamming distance is clearly less restrictive than L1 and L2 in expecting only the sign of program output to agree with the desired output. However, it assumes that the threshold between the negative and positive indications is zero, which, again, may seem quite arbitrary. An evolutionary process can hypothetically produce a program that perfectly classifies all training examples, but only when its output is thresholded on a different level. The subsequent search driver addresses this issue.

Pearson correlation. Another plausible search driver is the Pearson linear correlation coefficient of the vectors of actual and desired outputs:

$$f_P = 1 - \frac{cov(y, \hat{y})}{\sigma_y \sigma_{\hat{y}}}.$$

Compared to Hamming distance, f_P is advantageous in not requiring the evaluated program to produce positive values for the positive class and vice versa. For that instance, a program that returns 3 for all the negative examples and 7 for all the positive ones will be (rightfully) considered as perfect by this measure ($f_P = 0$) In this sense, this search driver is adaptive.

On the other hand, the correlation coefficient is, similarly to L1 and L2, sensitive to any intra-class variance in the observed program outputs $\hat{y}_i$. For instance, a program that returns, as in the above example, the output value 3

for all negative examples, but 7 *or* 11 for the positive ones, will receive inferior fitness. This is unfortunate, as this program's predictive capability is supreme: it is enough to threshold its output at, say, 5, to arrive at perfect decision rule. Ideally, we would like to have a search driver that is both liberal about the location of the boundary between the negative and positive output indications, and on the other hand interprets program output as an ordinal, rather than metric, variable. The subsequent search driver fulfills these expectations.

Area Under ROC curve (AUC). AUC is the total area under the Receiver Operating Characteristics (ROC) curve, the parametric curve spanning the false positive (FP) rate and the true positive (TP) rate. In our case, the parameter that controls the traversal of ROC curve is simply the threshold τ imposed on program output: if $\hat{y}_i < \tau$, the example is classified as negative, otherwise it is assigned to the positive class. AUC summarizes the behavior of this decision rule of all values of τ. By definition, AUC $\in [0, 1]$.

AUC is a natural way to characterize the trade-off between the FP rate and the TP rate. It can be cheaply calculated using the Mann-Whitney U test:

$$f_{AUC} = \frac{n^- n^+ + n^-(n^- + 1)/2 - r}{n^- n^+},$$

where n^- and n^+ are the sizes of the negative and the positive class, respectively, and r is the sum of ranks of sorted program outputs for the positive examples, i.e., $\hat{y}_i$s. AUC thus does not impose any specific threshold on program output (like f_H), and also does not assume the outputs to linearly correlate with the desired outputs (like f_P). What matters is only the *ordering* of the outputs produced by a program.

All the search drivers reviewed so far share one common characteristics: they consider all tests equally important. In practice however, some tests are often more difficult than others. This is particularly evident in image segmentation and diagnostic problems like the one considered here, where for instance some pixels evidently belong to the background. The following search driver takes this aspect into account.

Implicit fitness sharing. The last search driver considered in this study is implicit fitness sharing (IFS), introduced by Smith *et al.* [2] and further explored for genetic programming by McKay [1,3]. IFS lets evolution assess the difficulty of particular tests and *weighs* the rewards granted for solving them. Given a set of tests T, the IFS fitness of a program p in the context of a population P is defined as:

$$f_{IFS}(p) = \sum_{t \in T(p)} \frac{1}{|P(t)|}, \tag{1}$$

where $T(p)$ is the subset of tests (pairs $(\mathbf{x}, y)$) solved by p, and $P(t)$ is the subset of programs in P that solve test t. Thus, the $\frac{1}{|P(t)|}$ term serves here as an indicator of t's difficulty. Note that the denominator in the above formula never zeroes, because if p solves a given t, then $P(t)$ contains p.

IFS treats tests as limited resources: programs *share* the rewards for solving them, where a reward can vary from $\frac{1}{|P|}$ to 1 inclusive (the latter being the case when a program is the the only one in population that solves a test). Higher rewards are granted for solving tests that are rarely solved by population members (small $P(t)$), while importance of tests that are easy (large $P(t)$) is diminished. The assessed difficulties of tests change as P evolves, which can help escaping local minima.

By assessing programs in the context of the current population, fitness sharing can be perceived as a simple form of coevolution, where individuals compete for tests and their fate depends on the performance of other individuals (though there are no direct, face-to-face interactions between them). From yet another perspective, fitness sharing is 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 characteristics with explicit fitness sharing proposed in [4], where population diversity is enforced by monitoring the distances between individuals.

IFS requires defining what does it mean that a program passes a test. The repertoire of search drivers presented above clearly demonstrates that there are may ways in which this can be done when a continuously-valued program has to be interpreted as a binary classifier.

We propose the following procedure to determine which tests in a given training set T have been passed by which programs in the current population P. Assume there are n tests in T, and let $T^- \subset T$ denote the negative examples in T, and let $T^+ \subset T$ denote the positive examples in T. Let $n^- = |T^-|$ and $n^+ = |T^+|$.

1. For every program $p \in P$:

1.1. Sort the outputs $\hat{y}_i$ produced by p for tests $t_i \in T$ in an ascending order.
1.2. Let $T_p^- \subset T$ denote the subset of tests corresponding to the first n^- elements of the sorted list, and let $T_p^+ \subset T$ the be the subset of the remaining n^+ tests. Define the subset of tests $T(p)$ solved by p as

$$T(p) = T^- \cap T_p^- \ \cup \ T^+ \cap T_p^+ \tag{2}$$

2. For every test t, define the set of programs that pass t as

$$P(t) = \{p \in P : \ t \in T(p)\}. \tag{3}$$

3. Use $T(p)$'s and $P(t)$'s to evaluate the individuals in P according to Eq. (1).

Let us explain the rationale behind this algorithm. We assume that a perfect program would for all n^- negative examples produce smaller output values than for any of the n^+ positive examples. By sorting the program outputs in step 1.1 and partitioning them into the subset of first n^- elements and the remaining n^+ elements, we assess how close the program is to such behavior. The more negative examples end up in the former subset and the more positive examples in the latter, the better program's performance (cf. Eq. 2). On the other hand,

how many programs fail a test in this sense is a natural measure of its difficulty (Eq. 3). Redefining $T(p)$ and $P(t)$ in this way allows us to retain the original formula of IFS (Eq. 1).

Similarly to AUC, by referring only to the ordering of outputs, the above algorithm will adapt to any boundary between the decision classes that a program comes up with.

Apart from IFS, other methods have been proposed that reward solutions for having rare characteristics. An example is co-solvability [5] that focuses on individual's ability to properly handle *pairs* of fitness cases, and as such can be considered a 'second-order' IFS. Such pairs are treated as elementary competences (skills) for which solutions can be awarded. Lasarczyk *et al.* [6] proposed a method for selection of fitness cases based on a concept similar to co-solvability. The method maintains a weighted graph that spans fitness cases, where the weight of an edge reflects the historical frequency of a pair of tests being solved simultaneously. Fitness cases are then selected based on a sophisticated analysis of that graph.

4 Clinical Problem and Image Segmentation Task

The condition of the human vascular system is an important diagnostic factor in a large number of medical conditions like atherosclerosis or diabetes, to mention the common ones. No wonder thus that almost every modality of medical imaging features an operation mode that visualizes veins and/or arteries, like X-ray/CT angiography or Doppler ultrasonography.

A body organ that is particularly sensitive to pathologies of the vascular system is the eye. A malfunctioning of blood vessels in the retina has severe impact on the quality of vision. Contemporary, the most common cause of such anomalies is diabetes, which in 2012 had 9.3 % incidence in the US[1] and continues to be on the rise. As a result, the diabetic retinopathy affects over a quarter of adults with diabetes, and is currently the most common cause of blindness in the Western world.

There are several medical imaging techniques that allow assessing the state of retinal vascular system, including fundus imaging, fluoresceine angiography, and optical coherence tomography (OCT, [7]). In this study, we consider the first of them, which is arguably technically most straightforward: fundus images are simply pictures of the back of the eye taken in visible band using a camera fitted with appropriate optics. Our source of data is the DRIVE database [8][2], the result of a screening study conducted in the Netherlands on 400 subjects aged 25–90. The database contains a sample of 40 subjects from that group, 7 of which show mild signs of early diabetic retinopathy, and the remaining ones represent the clinical norm. Each image is a color raster taken with a 3CCD camera, where the anatomical structures occupy the central circular area with a radius of approximately 540 pixels.

[1] http://www.diabetes.org/diabetes-basics/statistics/.
[2] http://www.isi.uu.nl/Research/Databases/DRIVE/.

The database is divided into a training set and a test set, both composed of 20 images taken from different patients. For every image, manual segmentation of retinal vessels is provided (the original database contains two segmentations for the test set, but only the 'gold standard' one is used in this experiment). Figure 1 presents an exemplary training image and the corresponding manual segmentation.

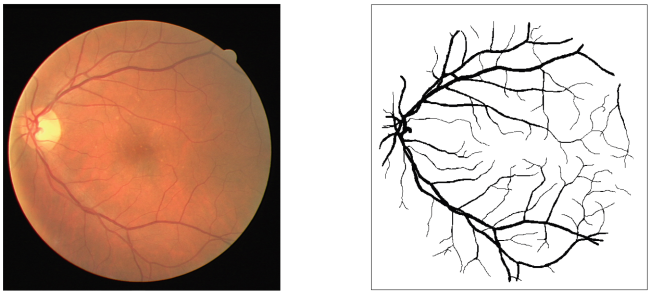

Fig. 1. An exemplary training image from the DRIVE database (left) and the corresponding manual segmentation (right).

5 Experimental Verification

In the experiment, we verify how useful are the particular GP search drivers presented in Sect. 3 for learning segmentation of blood vessels in the DRIVE database of fundus images (Sect. 4). In particular, we want to find out if the continuous variant of IFS proposed here brings any benefits. In parallel, we will determine whether the answers to these research questions depend to any extent on the characteristics of image features fed into programs.

Methods. The compared methods differ only in the search drivers presented in Sect. 3, which we employ as fitness function. Otherwise, all setups implement generational evolutionary algorithm and tree-based GP [9,10], with the initial population filled with the ramped half-and-half operator, subtree-replacing mutation engaged with probability 0.1, subtree-swapping crossover with probability 0.9, and tournament selection with tournament size 7.

Evolutionary runs last for 200 generations and work with population of 1000 programs. The instruction set comprises arithmetic ($+$, $-$, $*$ and protected division) and transcendental functions: exp,$sig(x) = 1/(1 + e^{-x})$, $lg|x|$, and normal distribution $n(x) = N_{(0,1)}(x)$. The terminals include the instructions that fetch image features, and constants drawn uniformly from the interval $[-1, 1]$. The incidence of constants in randomly generated trees is set to 0.1.

The final outcome of an evolutionary run is the best-of-run individual, i.e., the best individual found in any generation of the run.

Training set. We formulate the problem for GP as a classification task on the level of individual pixels. For a given pixel, the task for a GP classifier is to predict whether it represents a blood vessels or not. In this sense, each pixel forms a test (cf. Sect. 2).

Each of the 20 training images contains approximately 230 000 labeled pixels, far too many to be used for evolutionary training within a reasonable timeframe. Therefore, we select the training examples via sampling: from each training image, we draw at random 50 positive examples (pixels representing blood vessels) and 50 negative examples (pixels representing the background). Thus, the training set comprises $20 \times (50 + 50) = 2000$ fitness cases. The negative class is thus undersampled with respect to the positive one, and so GP works with balanced decision classes which should facilitate training. Originally, the decision classes are strongly imbalanced, with the positive class accounting for only about 12.7 percent of pixels [8].

Image features. As suggested earlier, this study focuses on the capability of different variants of GP to learn from the elementary image features, rather than on the features themselves. Therefore, we rely on relatively simple elementary image features inspired by BRIEF, Binary Robust Independent Elementary Features [11]. Originally, a single BRIEF feature is a binary indicator that tests the relationship between the values of two randomly chosen pixels p_1 and p_2 in the window I (ROI), i.e., whether $I(p_1) > I(p_2)$. To this aim, the originally color images are first converted to the monochrome scale.

We hybridize the BRIEF features with GP in the following way. Firstly, we delegate the choice of pixels to be compared to the evolutionary process, rather than drawing a sample of such pairs (which was originally done in [11]). To that aim, we include in the instruction set a terminal $d(p_1, p_2)$ that compares the brightness values of the pixels p_1 and p_2, which leads to a **binary** outcome:

$$d_b(p_1, p_2) = \begin{cases} 1 & I(p_1) > I(p_2) \\ 0 & otherwise \end{cases}.$$

The locations of p_1 and p_2 are defined in the reference frame of the current ROI of the detector. Given a rectangular ROI I of $m \times m$ pixels, there are $m^2(m^2 - 1)/2$ unique pairs of pixels. We use $m = 7$, which implies 1176 binary BRIEF features – input variables for a GP classifier.

Transforming two continuous pixel values into a binary indicator incurs substantial information loss. To preserve more information while still abstracting from the absolute pixel values, we consider also a **continuous** variant of BRIEF:

$$d_c(p_1, p_2) = I(p_1) - I(p_2).$$

Objective performance indicator. The compared methods use different fitness definitions as search drivers, which cannot be directly compared. To objectively compare the resulting classifiers, we employ the Area Under ROC curve indicator (AUC) detailed in Sect. 3. This performance indicator provides the full account (albeit aggregated to a single scalar) of the trade-off between the false positive rate and the true positive rate.

6 Results

Table 1 presents the average AUC of the best-of-run programs applied to the test set, averaged over 20 evolutionary runs. The estimation of AUC on the test set proceeds as for the training set, however this time 200 pixels are selected from each class for each of the 20 testing images, so the test set comprises $2 \times 200 \times 20 = 8000$ pixels.

Table 1. Test-set AUC of particular search drivers, aggregated over 20 evolutionary runs: average (left, with 0.95 confidence intervals) and median (right).

	Average		Median	
	Binary (d_b)	Continuous (d_c)	Binary (d_b)	Continuous (d_c)
L1	0.389 ± 0.008	0.547 ± 0.046	0.381	0.511
L2	0.717 ± 0.014	0.836 ± 0.008	0.720	0.836
Hamming	0.619 ± 0.075	0.811 ± 0.011	0.744	0.808
Pearson	0.799 ± 0.006	0.849 ± 0.010	0.802	0.853
GP-AUC	0.802 ± 0.004	0.902 ± 0.007	0.803	0.903
IFS	0.775 ± 0.006	0.904 ± 0.006	0.777	0.906

The superiority of the continuous features is unquestionable. For all search drivers, the AUC of the programs that access image by means of binary BRIEF features is substantially worse. For some methods (L1, Hamming) the gap on this performance indicator is dramatic. This applies to both training and testing set. Apparently, the instruction set used here does not interact well with the BRIEF features. When it comes to the usefulness of the particular search drivers, some observations are also evident. The worst performing search driver is L1: it does not perform well in any configuration. Compared to that, L2 achieves a much better level. The Hamming measure fares in between.

The top achievers are clearly the methods that abstract from the exact values of outputs produced by programs and take into account only their ordering, i.e. GP-AUC and IFS, followed by the Pearson correlation coefficient.

The similar performance of GP-AUC and IFS is not incidental. By comparing their descriptions in Sect. 3, it is easy to notice that they have much in common; in particular they both treat interpret output as an ordinal variable.

The differences between GP-AUC and IFS seem negligible, especially for the better performing continuous features. One cannot definitely claim any of them significantly better. Does it mean that the adaptation of IFS to the continuously-valued programs proposed in Sect. 3 does not bring any benefits?

It turns out it is not necessarily the case. In Table 2 we present the average sizes of the programs in the last populations of evolutionary runs and the best-of-run programs. The programs produced by IFS are substantially smaller than the ones evolved by GP-AUC, in spite of achieving roughly the same performance.

Table 2. Average size of the programs in the last populations of evolutionary runs (left) and of the best-of-run programs (right).

	Average size in population		Average size of best-of-run	
	Binary (d_b)	Continuous (d_c)	Binary (d_b)	Continuous (d_c)
L1	1.0	12.0	1.0	14.6
L2	12.6	52.5	16.1	57.3
Hamming	20.0	36.5	21.6	40.5
Pearson	56.0	30.7	60.5	35.1
GP-AUC	46.2	31.9	50.7	34.0
IFS	31.3	19.1	40.4	25.5

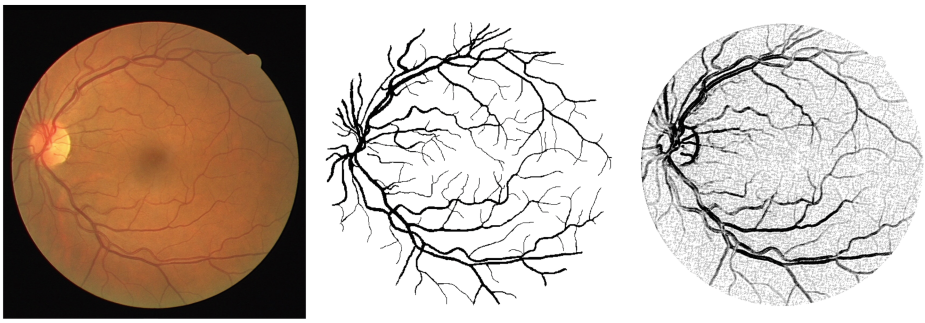

Fig. 2. A test image from the DRIVE database (test), the corresponding manual segmentation (middle), and the segmentation generated by the best program (GP-AUC, right). The rightmost image is not binary, because program output is not thresholded to visualize the continuous response of the detector.

The differences are observable in the averages taken over last populations, as well as in the average size of the best-of-run programs. The precise cause of this phenomenon is unclear; at this point we may only state that paying more attention to hard tests and less to the easy ones reduces code bloat.

Table 2 explains the surprising fact of L1 not surpassing 0.5 AUC in the binary configuration (Table 1): the average size of programs in that configuration is 1, which suggests that the specific way in which this search driver interprets the continuous program output precluded it from providing an effective search gradient, and search gets stuck with programs fetching single image features.

Figure 2 presents the segmentation obtained by applying the best detector from the GP-AUC runs and applying it an image from the testing set.

7 Conclusions

From the viewpoint of evolutionary image analysis, this work brings more evidence for the usefulness of BRIEF-like random features. Simplicity notwithstanding,

they offer reasonably good performance at the low expense of testing a few pixels in the ROI. In absolute terms, the best test-set AUC obtained here is 0.925 (one of the runs of GP-AUC, followed by the best of GP-IFS runs with AUC of 0.923), not far from 0.952 reported by Staal *et al.* in [8].

The overall conclusion of this study is that the choice of search driver is essential for the performance of evolutionary program synthesis. Although, given enough time, evolutionary algorithm should in principle be able to produce a continuously-valued program that closely matches the discrete desired outputs, it may be more reasonable to rely on the alternative search drivers. The experimental outcomes clearly indicate the AUC or the extension of IFS proposed here when solving binary classification tasks. We anticipate analogous results for similar medical and non-medical detection tasks.

In a broader perspective, with this work we reveal the potential dwelling in the alternative search drivers, especially those like IFS, which scrutinize program behavior on every test and do not reward programs equally for passing them. Here, we proposed and examined only a very simple approach of this kind; there are no principle reasons for extending this to, e.g., semantic GP [12].

Acknowledgments. This study has been supported by the National Centre for Research and Development grant # PBS1/A9/20/2013 and National Science Centre grant NCN grant 2011/01/DNZ4/05801.

References

1. McKay, R.I.B.: Fitness sharing in genetic programming. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.G. (eds) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Las Vegas, Nevada, USA, pp. 435–442. Morgan Kaufmann, 10–12 July 2000
2. Smith, R.E., Forrest, S., Perelson, A.S.: Searching for diverse, cooperative populations with genetic algorithms. Evol. Comput. **1**(2), 127–149 (1993)
3. McKay, R.I.B.: Committee learning of partial functions in fitness-shared genetic programming. In: Industrial Electronics Society, 2000. IECON 2000. 26th Annual Conference of the IEEE Third Asia-Pacific Conference on Simulated Evolution and Learning 2000, Nagoya, Japan, vol. 4, pp. 2861–2866. IEEE Press, 22–28 October 2000
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-wesley, Reading (1989)
5. Krawiec, K., Lichocki, P.: Using co-solvability to model and exploit synergetic effects in evolution. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part II. LNCS, vol. 6239, pp. 492–501. Springer, Heidelberg (2010)
6. Lasarczyk, C.W.G., Dittrich, P., Banzhaf, W.: Dynamic subset selection based on a fitness case topology. Evol. Comput. **12**(2), 223–242 (2004). (Summer 2004)
7. Sikorski, B., Bukowska, D., Ruminski, D., Gorczynska, I., Szkulmowski, M., Krawiec, K., Malukiewicz, G., Wojtkowski, M.: Visualization of 3d retinal microcapillary network using oct. Acta Ophthalmol. **91** (2013)
8. Staal, J., Abrà moff, M.D., Niemeijer, M., Viergever, M.A., van Ginneken, B.: Ridge-based vessel segmentation in color images of the retina. IEEE Trans. Med. Imaging **23**(4), 501–509 (2004)

9. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
10. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming. http://lulu.com and http://www.gp-field-guide.org.uk (2008) (With contributions by J. R. Koza)
11. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: BRIEF: binary robust independent elementary features. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 778–792. Springer, Heidelberg (2010)
12. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012)