Fractional Genetic Programming with Probability Density Data

Artur Rataj

Institute of Theoretical and Applied Computer Science,
Bałtycka 5, Gliwice, Poland
arataj@iitis.gliwice.pl

Abstract. We extend the fractional genetic programming scheme with data elements that are no more scalar, but instead are similar to probability density functions. The extension straightforwardly fits into fractional programming, in which data elements are blended from several values. In the case of our previous work, the blend produced a single scalar value. The extension proposes to build an approximate probability density function out of the blended elements.
The extension turned out to be very effective in an unsuspected way: when a data element, despite being destined to approximate a probability density, represented a single–dimensional image of spatial data.

Keywords: genetic programming, real–coded genetic algorithm, evolutionary method, probability density

1 Introduction

The fractional genetic programming (FGP), which we presented in [10], was built around the notion of gradualism – an instruction could be added into a program gradually, by increasing that instruction's strength, beginning with its very small value. A weak instruction had only a minimal influence on the program, as it affected the fractional data structures in a correspondingly minimal way. Tests have shown, that the graduality may give the programming scheme a substantial advantage, similar to that observed in the real–coded genetic programming [5, 11]. In order to take a data element e_{out} out of a fractional structure, a possibly stark data loss was required, though – e_{out} was a weighted mean of M data elements e_i^{in}, $i = 1, \dots M$, each having a weight w_i, related to the fraction of e_i^{in} actually removed from the data structure, in order to read e_{out}. We will further call the variant FGP_s, where S stands for scalar data elements. The discussed extension attempts to alleviate the data loss – the mixture of values $\left[e_i^{\text{in}}, w_i\right]$ is approximately preserved in e_{out} as a probability density function–like (PDF) set of N pairs (value, probability). Operations on these data elements are similar to these on PDFs, and thus require costly convolutions. To limit the complexity, N has typically a small value. The extended variant will further be referred as FGP_d, with D standing for density. Making genetic algorithms able to process

stochastic data had been tried before, see e.g. [6], yet we are not aware of a solution similar to ours in the subfield of genetic programming.

The extension has been introduced to alleviate yet another problem – processing of vectors, whose subsequent elements are related to each other, e.g. the index represents a spatial coordinate. Consider, for example, that a program processes a single–dimensional image consisting of, say, 10 pixels. In FGP_s, to hold these elements on a fractional stack, pushing of these 10 elements one on top of another would be required. In a program, that does not feature a loop, this may seem like a quite unwieldy way of presenting the data. A program would need to *separately* evolve distinct sets of instructions for popping and for initial processing of the the subsequent input values, one set per input value. Yet, it might be hypothesised, that the subsequent values would require a similar processing, and thus a much simpler program might perform the same task, if the input values were presented in a compact form, apt for a parallel processing by a single instruction. A similar reasoning might be applied to vectors of output data, and obviously, to the possible internal vectors used by the program for storing intermediate results. Taking into account, that the computational complexity of evolving a program may strongly increase with that program's length, the advantage could be clear. And here is the second purpose of the presented extension. While the elements are processed similarly to PDFs, they need not be ones – the genetic program is just expected to use the elements in the way most suitable for the task to realise. The formalism of combining data using operators like addition or division of PDFs might simply be a good mechanism to reuse:

- if a PDF has a zero or low variance, the combining works similarly to the corresponding operations on scalars, so the mechanism provides basic arithmetic operations;
- the operands are convoluted, and convolution is one of the more popular operations in the processing of mathematical data;
- last but not least, the similarity of the data elements to PDFs provides an approximate way of processing probabilistic data.

The paper is constructed as follows. Section 2 describes the handling of data in FGP_d. Sections 3 and 4 show examples, in which, respectively, the training data contains approximations of PDFs, and the training data are images of a parameterised space. Finally, there is a concluding discussion in Sec. 5.

2 Approximated PDFs

Let a data element be a set of N pairs (v_i, p_i), $i = 1 \ldots N$, $v_i < v_{i+1}$, $p_i \in \langle 0, 1 \rangle$, where v_i defines a value, and p_i is the probability of occurrence of a certain region $\langle r_i, r_{i+1} \rangle$, $r_i < v_i, r_{i+1} > v_i$, around v_i. Let the region be further called a layer. Let, because the number of layers N is typically a small value, the data element be called a sandwich. A sandwich, even that it is an approximation of a continuous PDF, contains only a set of discrete points. It is for computational performance, but also for a convenient representation of subsequent elements of

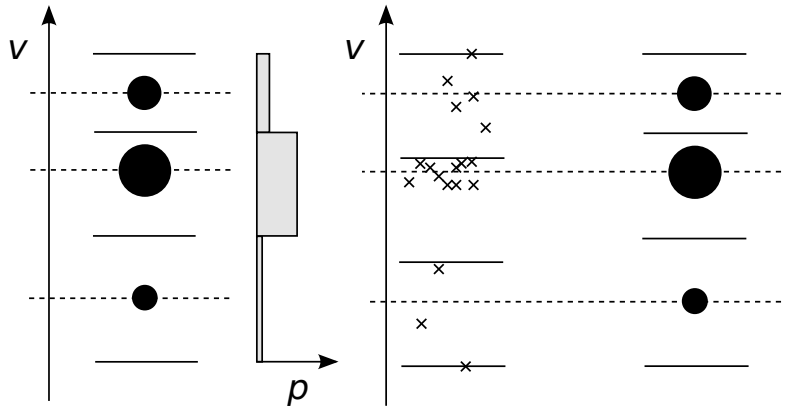


Fig. 1. An example sandwich and a continuous PDF representation, approximated by the sandwich (a); (b) creating a sandwich out of a cloud of points.

input or output vectors. Larger values of N provide a better approximation, but also a greater computational complexity.

2.1 A corresponding continuous PDF

The values r_i define a corresponding continuous PDF, approximated by a sandwich, as illustrated in Fig. 1(a). They are computed as follows:

$$r_i = \begin{cases} v_i - (r_{i+1} - v_i) & \text{for } i = 0 \\ \frac{v_{i-1}+v_i}{2} & \text{for } i \geq 1 \wedge i \leq N \\ v_{i-1} + (v_{i-1} - r_{i-1}) & \text{for } i = N+1 \end{cases} . \tag{1}$$

Thus, a border of 'influence' of the probability value is halfway between two neighbouring values v_i, and the two extreme regions are symmetric, relatively to the respective values. The reason for choosing such a corresponding PDF was a computational speed and the lack of any concrete statistical model between the sandwich – as will be seen in the example in Sec. 4, there is no general enforcement of any such model on the input/output data to the extend of p_i not representing the probability at all.

2.2 Unary operators

An instruction $W = f(U)$, representing an unary operator, involves only a single sandwich, and thus its definition is trivial:

$$\begin{aligned} v_i^W &= f(v_i^U), \\ p_i^W &= p_i^U, \\ i &= 1, \ldots N, \end{aligned} \tag{2}$$

where U consists of v_i^U, p_i^U and W consists of v_i^W, p_i^W. As we see, the definition is a generalisation of an unary operation on a scalar – U, and in effect V, would contain only a single non–zero value p_i^U in such a special case.

2.3 Binary operators

As p_i represent the probability of occurrence of an event of a value v_i, an instruction representing a binary operator on two sandwiches requires the computation of a joint probability. Thus, up to N^2 events of a non–zero probability may represent the outcome of the operation. Combining PDFs is a complex problem in general – two stochastic variables might be dependent on each other in various ways, and involved methods of tracking the dependence might be necessary [7]. Assuming, that two PDFs are always independent in a computer language, which is to be used by a human, may lead to counter–intuitive results. Yet, in our case, it is an evolutionary method that designs the program, and thus we can maintain the merely loose connection of sandwiches to PDFs – it is never explicitly stated in the method, that the sandwiches are PDFs anyway. Let us thus assume, that events in separate sandwiches are independent – no need of tracking the possible dependencies saves time, and the temporal performance is very often of an essential importance in evolutionary computing [4].

Let (v_i^L, p_i^L) and (v_i^R, p_i^R), represent, respectively, the left and the right operand, and let an arbitrary binary operator, which combines two scalar values into a third scalar value, be symbolised by $\odot$. Then:

$$\begin{aligned} w_{i,j} &= v_i^L \odot v_j^R, \\ p_{i,j} &= p_i^L p_j^R, \\ i = 1, \ldots N,\ j &= 1, \ldots N. \end{aligned} \tag{3}$$

Following the set of instructions defined in [10], in the further presented tests the following binary operators are employed: $+, -, *, /$.

There are too many resulting pairs $(w_{i,j}, p_{i,j})$ to fit into the sandwich representing the result – to avoid explosion of complexity, data loss might be necessary. The problem of creating a sandwich out of M points, $M > N$, occurs not only when a result of a binary operation needs to be parsed – as the example in Sec. 3 will illustrate, fitting of a cloud of points into a sandwich may also occur when preparing data for a FGP_d. The next section proposes a fitting method to be used in both cases.

2.4 Fitting a cloud of points

To create a sandwich based on a set of M points, M being an arbitrary value, we could use one of the many clustering methods [2, 3]. Yet in genetic programming, a fast rough tool might be better than a precise yet slow one – the latter may slow down the optimisation process, and in effect decrease, and not increase, the quality of the best candidate. Using a rough, fast tool might have a special

sense in our case, as the sandwiches, as discussed, are not strictly semantically defined, so it is hard to estimate, what would be meant by a precise tool.

Let the cloud consists of points $\left(v_i^C, p_i^C\right)$, $i = 1 \dots M$, where v_i^C is a scalar value and p_i^C is a probability of its occurrence. The value p_i^C is introduced mainly to deal with $(w_{i,j}, p_{i,j})$ defined in (3), but it might also represent e.g. an importance of some element in a training set. Let the sandwich to create out of the cloud consists of $\left(v_k^Q, p_k^Q\right)$, $k = 1 \dots N$. Let the cloud be split in regard to v_i^C into N subsets S_k,

$$\begin{gathered} t_j = \min_i v_i^C + \frac{j}{N+1}\left(\max_i v_i^C - \min_i v_i^C\right), \\ j = 1 \dots N+1, \\ \left(v_i^C, p_i^C\right) \in S_k \iff \left[\left(v_i^C = t_j \wedge j = 1\right) \vee v_i^C > t_j\right] \wedge v_i^C \le t_{j+1}. \end{gathered} \tag{4}$$

The mean value of points within S_k, weighted by probabilities, becomes v_k^Q, and the probabilities p_k^Q maintain the same ratios to each other, as the k sums of probabilities within S_k do:

$$v_k^Q = \frac{\sum\limits_{S_k} v_i^C p_i^C}{\sum\limits_{S_k} p_i^C}, \quad p_k^Q = \frac{\sum\limits_{S_k} p_i^C}{\sum\limits_{i} p_i^C}. \tag{5}$$

An example fitting is illustrated in Fig. 1(b).

2.5 Comparing sandwiches

Comparing two sandwiches, just like the comparison of two PDFs, can have many meanings. Consider applying of an absolute difference operator to (3):

$$\begin{gathered} w_{i,j} = \left|v_i^L - v_j^R\right|, \\ p_{i,j} = p_i^L p_j^R, \\ i = 1, \dots N,\ j = 1, \dots N. \end{gathered} \tag{6}$$

Intuitively, it is a stochastic equivalent of $d = |t - y|$, where $d, t, y \in \mathbb{R}$. The outcome d, in turn, can be a base for computing the mean square error (MSE) of fitting, if t is a desired value in a supervised training, and y is an actual value, computed by a candidate. Yet, conversely, the mean value $m = \overline{|T - Y|}$ might be a bad base for MSE, if T and Y are PDFs. It is because even if an ideal outcome T is exactly the same as the outcome Y produced by a candidate, $m \neq 0$ if $\mathrm{Var}(T) = \mathrm{Var}(Y) \neq 0$. Obviously, the stochastic nature of the desired outcome would creep into MSE. Thus, even that (6) might represent a helpful instruction in a program, we we will use another method here for comparing two sandwiches, such that assesses their goodness of fitting. A typical approach is the

Kolmogorov–Smirnov statistic [8], but we will rather use the Cramér–von Mises criterion [1, 9], as the former conveys only the most different part of the empirical distribution functions of T and Y, hiding any improvements or regressions in the other parts.

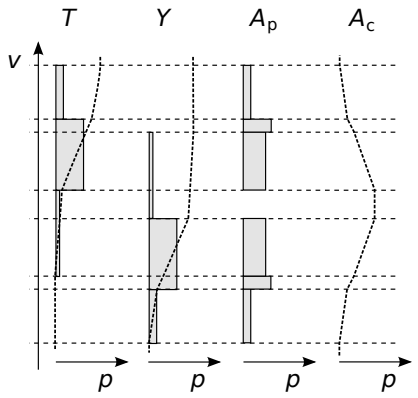


Fig. 2. Comparing of two sandwiches in order to estimate the goodness of fitting. Dashed lines represent CDFs (T, Y) or a difference between two CDFs (A_c).

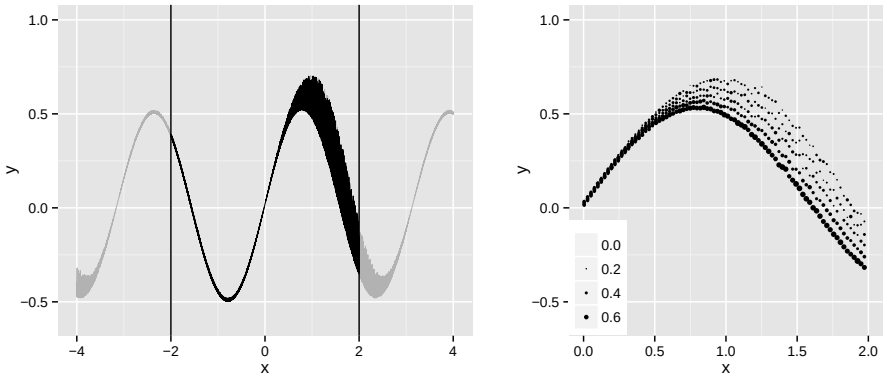
Fig. 2 shows the absolute differences A_p and A_c between, respectively, PDFs and cumulative distribution functions (CDFs). Obviously, the area of A_p would be a bad similarity estimator, as e.g. it would not increase, if two non–overlapping PDFs were moved away from each other. The area of A_c is the value of the said Cramér–von Mises criterion.

3 Extrapolation of a stochastic function

Let there be a stochastic function

$$y = \frac{\sin(2x)}{2} + \frac{e^{3\sin(x)u(0,1)}}{50} \tag{7}$$

where $u(0,1)$ is a random variable, having a uniform distribution spanning values $\langle 0, 1\rangle$. A cloud of points, consisting of 10000 observations of (7), is shown in Fig. 3(a). Let the training sandwiches be computed, as defined in Fig. 3(b). A supervised learning, using the evolutionary method described in [10], has been performed, with $\mathrm{MSE}_{\max} = 0.005$, and the maximum time allowed $t_{\max} = 10000\,\mathrm{s}$. A mean x value of observations fitted to a sandwich Q was inputted to a candidate, and the response of the candidate has been compared to Q. The better candidate in Fig. 4 was one of the best programs obtained. We see a poor to moderate fitting quality. The asymmetric nature of the noise applied to the scaled sine function is not visible in the candidate's response. There is an extrapolation of a moderate quality at $x \in \langle 2, 3.5\rangle$, yet it might be a coincidence, as the same can not be said about the extrapolation at $x < -2$.



(a) (b)

Fig. 3. A cloud generated by the extrapolated stochastic equation (a), training observations marked black, and (b) the corresponding fitted sandwiches for $x \in \langle 0, 2\rangle$, 60 observations having subsequent x per sandwich, $N = 5$; circle area denotes probability within a sandwich. The common horizontal position of a set of circles defines a single sandwich, and is equal to the mean x value within the respective observations.

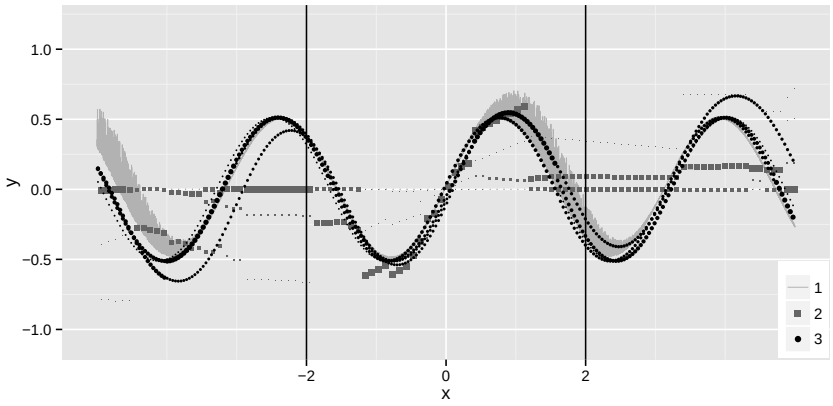


Fig. 4. Light gray region represents observations of the extrapolated equation, training observations limited to $\langle -2, 2\rangle$ as show in Fig. 3(a). Squares denote an example of a badly fitted candidate, circles denote a moderately fitted candidate, MSE ≈ 0.0014. Sandwiches are illustrated as described in Fig. 3(b).

4 Navigation in a parameterised space

Let us now try a very different task, to assess the universality of the method. There is a turtle, that starts somewhere in a two–dimensional space, parame-

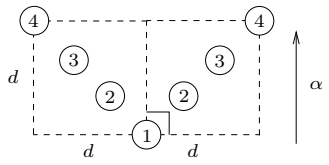


Fig. 5. Turtle's senses. In the FGP_d mode, a value f inside a circle denote a pair (v_f, p_f) in one of the two input sandwiches.

terised by a function $v = s(x, y)$, belonging to the family

$$\begin{aligned} x \in \langle -1, 1\rangle\,,\ y \in \langle -1, 1\rangle\,,\\ x_r = x + A\sin x + B\cos y,\\ y_r = y + C\cos x + D\sin y,\\ r = \sqrt{x_r^2 + y_r^2},\\ v = \tfrac{10\cos r}{10+r} \end{aligned} \tag{8}$$

The turtle has a position $t_x \in \langle -1, 1\rangle, t_y \in \langle -1, 1\rangle$, and an orientation $\alpha \in \langle 0, 2\pi)$. The turtle has two eyes. One is looking 45° to the left, relatively to α, and acquiring F parameters of the surrounding space e_f^{-1}, $f = 1 \ldots F$. Analogously, the other eye looks to the right, and collects e_f^1 parameters. If $d = 0.05$ is the step length along α, which the turtle performs in a single iteration, then the points $\left(e^D_{f,x}, e^D_{f,y}\right)$, $D = -1, 1$, at which the parameters are collected, are as follows:

$$\begin{aligned} g = \frac{f-1}{F-1}\frac{d}{5},\\ e^D_{f,x} = t_x + g(D\sin\alpha + \cos\alpha),\\ e^D_{f,y} = t_y + g(-D\cos\alpha + \sin\alpha). \end{aligned} \tag{9}$$

The way, in which a turtle can see its surroundings for $F = 4$, is illustrated in Fig. 5.

A turtle's goal is to find the maximum global value of the parameter (8), using only its eyes. According to (8), the maximum value is located at a point H, at which $r = 0$.

The training has been performed as follows. A space has been parameterised by choosing $A = 0.5, B = 0.6, C = 0.4, D = 0.3$. A number of fixed starting locations and orientations has been selected in the space. Both the parameters and the starting locations are shown in Fig. 6(a). To estimate a candidate's quality, a single turtle starts from each of these locations. Each turtle has 100 steps at its disposal, and thus can travel $100d$ at most. If a turtle either travels the maximum distance, or walks to a position closer than a single step d to H, it stops. What the turtle sees is fed to the estimated candidate, and the mean of the returned value modifies α. The modified α is used in the next step. So, the candidate's role is to determine, where a turtle turns after each

step, depending on what the turtle saw in its direct surroundings. After all turtles stop, their mean distance to H becomes the candidate's error. The same evolutionary method as in Sec. 3 has been used, with $\mathrm{MSE}_{\max} = 0.05$, being a misnomer in this case, and the maximum time allowed $t_{\max} = 10000$s.

Fig. 6 presents tracks of turtles for an example candidate, whose $\mathrm{MSE}_{\max} = 0.0022$. We see, that the testing of the candidate by choosing alternate starting points, not used within training, was largely successful, even that in the test most of the turtles started at much greater distances from H. Fig. 7 shows the fractional program behind that candidate.

The test in Sec. 3 did not allow for a comparison of FGP_s and FGP_d, because the former was not able to process stochastic data. Yet, in this test the data is not stochastic, so let us compare the efficiency of these two versions. Let in FGP_s the sensed parameters, or 'ground's height', are pushed to the stack in the following order: left eye's $f = 1, \ldots 4$, then right eye's $f = 1, \ldots 4$, the numbers f as shown in the circles in Fig. 5. In FGP_d, let $F = N$, and let us use only two sandwiches, (v_f^{-1}, p_f^{-1}) and (v_f^1, p_f^1) for respectively the left and the right eye, to avoid the problematic stacking of many values, discussed in Sec. 1. These sandwiches won't be used like PDFs at all – let

$$\begin{aligned} v_f^D &= \frac{f-1}{F-1}, \\ p_f^D &= \frac{s(e_{f,x}^D, e_{f,y}^D)}{n}, \end{aligned} \tag{10}$$

where n is a normalising constant, so that p_f^D sums to 1. See that p_f^D now conveys the 'ground's height', hardly related to a probability.

Let a test be a single run of the evolutionary method, that explores and exploits candidates, until a good one is found, or the maximum time is reached, as described in [10]. In the FGP_s version, we ran 60 such tests for $F = 3$, of whose only 4 produced a candidate meeting the quality criterion given by $\mathrm{MSE}_{\max}$, and 60 tests for $F = 5$, which produced only one such a candidate. In the FGP_d version, 120 tests were run for each value of $F = 2, 3, 5$. Fig. 8 summarises the results. We see in Fig. 8(a), that the mean number of evaluations of candidates, required to produce a successful candidate by a test with a given probability, decreases with F, and thus with N. Yet, it does not mean, that increasing the precision of sandwiches N brings a successful candidate faster. When compared to in Fig. 8(b), we see that a single evaluation lasts predictably longer for larger N, and in effect the optimal value of N is 3 here.

The number of tests that produced a successful candidates in the FGP_d version is 111, 119 and 108 for $F = 2, 3, 5$, respectively. Thus, more than 93% of tests brought candidates which were good enough, vs 7% in the case of the FGP_s version. The mean time in the FGP_s version, as seen in Fig. 8(b), was shortest for $F = 3$, reaching about 3000 s. For the FGP_dversion it obviously was much longer, as the unsuccessful 93% of tests ran for $t_{\max} = 10000$ s.

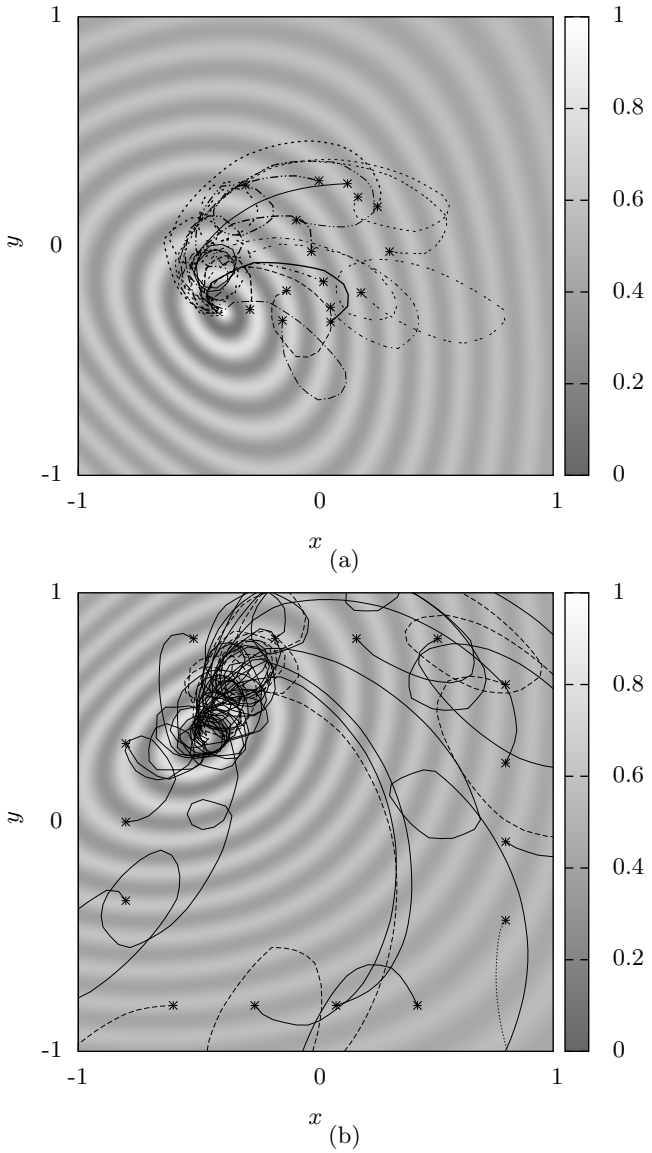


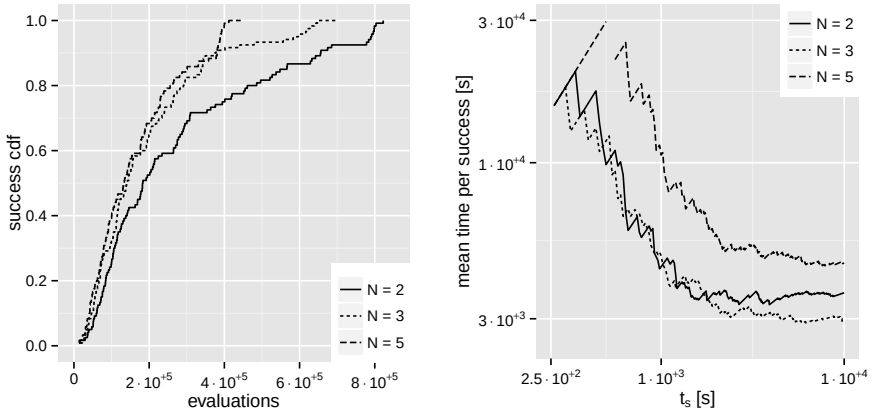
Fig. 6. Tracks of turtles, decided by an example candidate, FGP_d, $N = 3$: (a) starting at training locations, all succeeded, (b) starting at testing positions, with parameters computed using $A = 0.4, B = 0.7, C = -0.6, D = 0.5$; solid line: succeeded within 100 steps, dashed line: succeeded within 200 steps, dotted line: unsuccessful within 300 steps.

5 Discussion

An introduction of sandwiches into the fractional programs produced mixed results in the test of an extrapolation of a stochastic equation. We hypothesise,

```
0 <0.76>WRT -5.1@0.26|6.4@0.41|4.3@0.33
1 <0.64>WRT -1.0@0.45|2.8@0.09|-7.9@0.46
2 <0.68>MUL
3 <0.05>SIN
4 <0.49>POP
5 <0.60>WRT -0.4@0.66|6.3@0.18|-4.0@0.16
6 <0.44>WRT 6.2@0.35|-1.8@0.32|4.0@0.33
7 <0.46>INV
8 <0.04>MUL
9 <0.39>WRT 3.6@0.43|5.4@0.07|-2.7@0.50
10 <0.17>NEG
11 <0.88>ADD
12 <0.05>NEG
13 <0.27>WRT -5.3@0.55|-0.7@0.18|-5.0@0.27
14 <0.89>MUL
15 <0.10>WRT -7.5@0.51|5.4@0.45|6.9@0.04
16 <0.40>POP
17 <0.82>SIN
18 <0.17>INV
19 <0.79>SIN
```

Fig. 7. Candidate, which chose the tracks visible in Fig. 6.



(a) (b)

Fig. 8. Efficiency of FGP_d: (a) CDF of the probability of producing a successful candidate by a test, vs the number of required evaluations of the candidate; (b) mean CPU time required to produce a successful candidate for different $t_{\max}$.

that the candidates might be missing some special instructions, that directly operate on probabilities. We plan to search for such instructions in the future. Employing the sandwiches in passing of a sequence of training data showed a large improvement, even that the sandwiches did not necessarily operate as PDF approximations in that case. It might thus make it worth to introduce instructions, which treat the sandwiches as general data containers, and not merely approximations of PDFs.

References

1. Cramer, H.: On the composition of elementary errors. Scandinavian Actuarial Journal (1928)
2. Davé, R.N., Krishnapuram, R.: Robust clustering methods: a unified view. Fuzzy Systems, IEEE Transactions on 5(2), 270–293 (1997)
3. Filippone, M., Camastra, F., Masulli, F., Rovetta, S.: A survey of kernel and spectral methods for clustering. Pattern recognition 41(1), 176–190 (2008)
4. Fok, K.L., Wong, T.T., Wong, M.L.: Evolutionary computing on consumer-level graphics hardware. IEEE Intelligent Systems 22(2), 69–78 (2007)

5. Goldberg, D.E.: Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Systems 5, 139–167 (1990)
6. Iwamura, K., Liu, B.: A genetic algorithm for chance constrained programming. Journal of Information and Optimization Sciences 17(2), 409–422 (1996)
7. Li, W., Hyman, J.M.: Computer arithmetic for probability distribution variables. Rel. Eng. & Sys. Safety 85(1-3), 191–209 (2004)
8. Marsaglia, G., Tsang, W.W., Wang, J.: Evaluating kolmogorov's distribution. Journal of Statistical Software 8(18), 1–4 (2003)
9. von Mises, R.E.: Wahrscheinlichkeit. Statistik und Wahrheit (1928)
10. Rataj, A.: Fractional genetic programming for a more gradual evolution. In: CS&P. pp. 371–382 (2013)
11. Wright, A.H., et al.: Genetic algorithms for real parameter optimization. In: Foundations of Genetic Algorithms. pp. 205–218 (1990)