PAC Learning and Genetic Programming

Timo Kötzing
Algorithms and Complexity
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany

Frank Neumann
School of Computer Science
University of Adelaide
Adelaide, SA 5005, Australia

Reto Spöhel
Algorithms and Complexity
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany

ABSTRACT

Genetic programming (GP) is a very successful type of learning algorithm that is hard to understand from a theoretical point of view. With this paper we contribute to the computational complexity analysis of genetic programming that has been started recently. We analyze GP in the well-known PAC learning framework and point out how it can observe quality changes in the the evolution of functions by random sampling. This leads to computational complexity bounds for a linear GP algorithm for perfectly learning any member of a simple class of linear pseudo-Boolean functions. Furthermore, we show that the same algorithm on the functions from the same class finds good approximations of the target function in less time.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Algorithms, Performance

Keywords

Genetic Programming, PAC Learning, Theory, Runtime Analysis

1. INTRODUCTION

Genetic programming (GP) [7] is an algorithmic approach inspired by the evolution process in nature that has found numerous applications in various domains (see e.g. Poli et al. [11]). With this paper, we contribute to the theoretical understanding of genetic programming by analyzing its behavior in a rigorous way. This approach was already very successful in the field of evolutionary algorithms, where computational complexity analysis has significantly increased the theoretical understanding. Initial results on the computational complexity of evolutionary algorithms for artificial pseudo-Boolean functions [13, 1] have set the basis for later results on classical combinatorial optimization (see Neumann and Witt [10] for an overview). Poli et al. [12] state, "we expect to see computational complexity techniques being used to model simpler GP systems, perhaps GP systems based on mutation and stochastic hill-climbing."

The computational complexity analysis of genetic programming has just been started by Durrett et al. [2]. In their paper, they focus on simple problems such as ORDER and MAJORITY introduced by Goldberg and O'Reilly [6] and analyze the time to achieve an exact solution for these given problems. Often it is the case that problems cannot be solved exactly and one is satisfied with a good approximation. The most classical approach in computational learning theory is to study this within the approximately correct (PAC) learning framework of Valiant [14]. Recently, it has been shown that a large class of functions is evolvable, i. e. PAC learnable by an evolutionary algorithm [15, 3, 4]. However, the goal of these papers is to understand natural evolution from a theoretical point of view rather than giving explanations why and how evolutionary algorithms that are used in practice work. The main difference of these studies to evolutionary algorithms used in practice relies in the mutation operator which is very powerful, as it has the only restriction that it works in polynomial time. On the other hand, in GP mutation and crossover operators are much more limited, as they either change solutions only slightly or combine them into new ones. Motivated by this, we examine what classes of functions can be learned by genetic programming that is restricted to simple mutation and/or crossover operators.

The PAC learning framework provides a formal basis for analyzing learning problems and has already been used for the analysis of learning experiments in genetic programming [8]. We start the computational complexity analysis of linear genetic programming [5] in the PAC learning framework. In contrast to tree-based genetic programming, linear genetic programming does not evolve a tree structure but evolves a linear representation of possible functions. Studying our algorithms in the PAC learning framework enables us to study the learnability of linear genetic programming in a rigorous manner. We provide a framework for such an analysis and study a simple learning problem that is related to the OneMax problem known from the computational complexity analysis of evolutionary algorithms in the discrete domain [9, 1].

We proceed as follows. Section 2 introduces the problem and the algorithms that are subject to our analysis. We point out how the difference in the quality of solutions can be observed by random sampling in Section 3. These insights are used in Sections 4 and 5 to analyze the computational complexity for learning exact functions as well as good approximations. Finally, we finish with some concluding remarks.

2. PROBLEM AND ALGORITHMS

We want to study simple genetic programming models in the PAC learning framework. In contrast to simple evolutionary algorithms whose computational complexity has been studied, we have to deal with two sources of randomness. The first source is due to the algorithm itself which is based on random decisions. The second source of randomness is due to the PAC learning framework which returns the quality of a potential solution based on random sampling.

Our goal is to learn a linear function on bit strings $x \in \{0,1\}^n$

$$f_{\mathrm{OPT}}(x) = \sum_{i=1}^{n} w_i x_i$$

where $w_i \in \{-1, 1\}$, $1 \leq i \leq n$.

We call this the *identification problem* and investigate a simple genetic programming algorithm that starts with a (random) function f and evolves it over time. Given a distribution D over the inputs of the target function f_{OPT}, the goal is to obtain a function f that outputs function values similar to those returned by f_{OPT} on random inputs from D. In this paper, we assume D to be the uniform distribution over $\{0,1\}^n$ and measure the quality of a solution f by the expected error $|f_{\mathrm{OPT}}(x) - f(x)|$ of a random bit string x. In the ideal case, we would like to get $f = f_{\mathrm{OPT}}$ but this is not always possible and not required for our model of learning.

We do not have direct access to the *expected* error of a given function f. Instead, we approximate the quality of f (depending on the target function f_{OPT} and the given distribution D) by sampling a multi-set of points S from D and then adding up relative errors of the $x \in S$ with respect to f_{OPT}. Formally, we let the error of f with respect to f_{OPT} on a single bit string $x \in \{0,1\}^n$ be

$$e_x(f, f_{\mathrm{OPT}}) = |f(x) - f_{\mathrm{OPT}}(x)|,$$

and the error of f with respect to f_{OPT} on a set $S \subseteq \{0,1\}^n$ be

$$e_S(f, f_{\mathrm{OPT}}) = \sum_{x \in S} e_x(f, f_{\mathrm{OPT}}).$$

We denote by $|S|$ the sample size that is used in each iteration of the algorithm. Of course, the sample size is of major importance: on the one hand, it needs to be high enough for the algorithm to be able to distinguish the quality of different search points; on the other hand, it should be low for the efficiency of practical implementations. A crucial point of our analysis below is to show that a low polynomial number of samples suffices in our setting.

Note that, depending on the given distribution, elements might occur in S more than once. However, if we consider the uniform distribution in our analysis and assume that the sample size is always small compared to the size of the sample space, this is very unlikely.

Algorithm 1: Linear GP for learning functions.

```
1 input Black-box target function $f_{\mathrm{OPT}}$ ;
2 input Sample size $z$;
3 initialization Uniformly at random choose an initial
  function $f = \sum_{i=1}^{n} w_i x_i$;
4 repeat
5     Choose $i \in [n]$ uniformly at random;
6     Obtain $f'$ from $f$ by flipping the $i$th weight;
7     Sample a set $S \subseteq \{0,1\}^n$ of size $z$;
8     if $e_S(f', f_{\mathrm{OPT}}) \leq e_S(f, f_{\mathrm{OPT}})$ then $f \leftarrow f'$
9 until forever;
```

We consider a simple linear GP algorithm [5] called Linear GP (see Algorithm 1) which is similar to the evolutionary algorithm analyzed by Valiant for the optimization of monotone conjunctions [15]. Note that it does not work with a tree structure as the tree-based genetic programming approach analyzed by Durrett et al. [2]. We are rather interested in how the right coefficients can be learned for the given class of functions.

Our algorithm starts with a randomly chosen function f and generates, in each iteration, a new function f' by flipping exactly one weight. Flipping a weight w_i means that w_i is replaced by $-(w_i)$, i. e. $+1$ turns into -1 and -1 turns into $+1$. The new function f' replaces f if it has a smaller error according to the error function e_S.

Our goal is to analyze the expected number of iterations until Linear GP has produced, for the first time, a solution f that makes with probability $1-\epsilon$ an error of at most δ when choosing an element $x \in \{0,1\}^n$ according to D. For exact learning, we set $\delta = 0$ and $\epsilon = 0$ and call the expected number of iterations to achieve a solution to this the *expected learning time* of the algorithm.

3. PROPERTIES OF UNIFORM SAMPLING

As pointed out previously, we consider sampling from $\{0,1\}^n$ according to the uniform distribution. To analyze the progress that our algorithm can make on the considered problem, we have to figure out how it can distinguish between the quality of solutions. Note that the fitness of a solution is not deterministic but itself a random variable that is determined by the set of samples that is chosen in each iteration.

We start with a lemma that will be important for our analysis. For all n, we denote with S_n and S'_n two independent random variables which are both the sum of n independent Bernoulli trials with success probability $1/2$.

The following lemma analyzes, for different values of k, the probability that S_{n+k}, the number of successes in $n+k$ trials, is larger than S'_n, the number of successes in n trials.

LEMMA 1. *Let $k, n \geq 0$ and $p_n = \frac{1}{2}\binom{2n}{n}2^{-2n}$. Then*

$$P(S_{n+k} > S'_n) = \frac{1}{2} + \sum_{i=0}^{k-1} \binom{2n+i}{n} 2^{-2n-i-1} - p_n.$$

Note that p_n is exactly the $i = 0$ term of the sum, and that (by Stirling's formula) we have

$$p_n = (1 + o(1)) \frac{c}{\sqrt{n}}$$

for an appropriate constant c.

PROOF. We prove Lemma 1 by induction on k. For $k = 0$, we have, using symmetry and Vandermonde's identity,

$$\begin{aligned} P(S_n > S'_n) &= \frac{1}{2}(1 - P(S_n = S'_n)) \\ &= \frac{1}{2} - \frac{1}{2}\sum_{i=0}^{n}\binom{n}{i}\binom{n}{i}2^{-2n} \\ &= \frac{1}{2} - \frac{1}{2}\binom{2n}{n}2^{-2n} \\ &= \frac{1}{2} - p_n. \end{aligned}$$

Suppose the formula holds for some k. By conditioning the random variable S_{n+k+1} onto the outcome of the first trial, we get the following.

$$\begin{aligned} &P(S_{n+k+1} > S'_n) \\ &= \frac{1}{2}P(S_{n+k} > S'_n) + \frac{1}{2}P(1 + S_{n+k} > S'_n) \\ &= P(S_{n+k} > S'_n) + \frac{1}{2}P(S_{n+k} = S'_n) \\ &= P(S_{n+k} > S'_n) + \frac{1}{2}\sum_{i=0}^{n}\binom{n+k}{i}\binom{n}{i}2^{-2n-k} \\ &= P(S_{n+k} > S'_n) + \binom{2n+k}{n}2^{-2n-k-1} \\ &= \frac{1}{2} + \sum_{i=0}^{k}\binom{2n+i}{n}2^{-2n-i-1} - p_n, \end{aligned}$$

where in the last step we used the induction hypothesis. □

For convenience we state the following consequences of Lemma 1.

COROLLARY 2. *For all $0 \le x \le y \le n$ we have*

$$P(S_y > S'_x) \begin{cases} \le \frac{1}{2} - p_n & \text{if } y = x \\ = \frac{1}{2} & \text{if } y = x + 1 \\ \ge \frac{1}{2} + \frac{p_n}{2} & \text{if } y \ge x + 2. \end{cases} \quad (1)$$

$$P(S_y \ge S'_x) \ \ge \frac{1}{2} + p_n \quad (2)$$

$$P(S_y < S'_x) \ \le \frac{1}{2} - p_n \quad (3)$$

$$P(S_y \le S'_x) \begin{cases} \ge \frac{1}{2} + p_n & \text{if } y = x \\ = \frac{1}{2} & \text{if } y = x + 1 \\ \le \frac{1}{2} - \frac{p_n}{2} & \text{if } y \ge x + 2. \end{cases} \quad (4)$$

PROOF. We only prove the statements concerning $P(S_y > S'_x)$ – the other statements are obtained similarly or by taking complements. The bounds for $y = x$ and $y = x + 1$ are obtained by applying Lemma 1 and observing that $p_x \ge p_n$. For the inequality when $y \ge x + 2$, we obtain by monotonicity and again with Lemma 1 that

$$\begin{aligned} P(S_y > S'_x) &\ge P(S_{x+2} > S'_x) \\ &= 1/2 + \binom{2x+1}{x}2^{-2x-2} \\ &\ge \frac{1}{2} + \frac{p_x}{2} \ge \frac{1}{2} + \frac{p_n}{2}. \end{aligned}$$

□

f_{OPT}	$+1, \dots, +1$	$+1, \dots, +1$	$-1, \dots, -1$	$-1, \dots, -1$
f	$+1, \dots, +1$	$-1, \dots, -1$	$+1, \dots, +1$	$-1, \dots, -1$
	$a(f)$	$b(f)$	$c(f)$	$d(f)$

Table 1: Difference between f_{OPT} and f.

It follows from standard Chernoff bounds that for an appropriate constant c,

$$z' = c(p_n)^{-2}\log(n)$$

coin tosses suffice to correctly identify the side of the bias of a coin with probability $\ge (1 - n^{-4})$, provided this bias is at least $p_n/2$ (i.e., the coin comes up heads with probability either $\ge 1/2 + p_n/2$ or $\le 1 - p_n/2$).

As we will see, in each iteration of Linear GP we are faced with the problem of determining such a bias, with the random samples playing the role of coin tosses. To be more precise, the number of coin tosses corresponds to half the number of samples: Only half of the samples help in identifying the bias, namely those which have a 1 at the position of the flip. As moreover the bias is indeed always at least $p_n/2$, the error probability with $z = 2z'$ samples is $\le n^{-4}$ in every iteration, which implies by a union bound that *with probability $1 - o(1)$ we correctly identify the bias in each of the first n^3 iterations* (the algorithm will be done long before that number of iterations). We may and will assume throughout the rest of this paper that the bias is correctly identified whenever Linear GP makes a fitness comparison as described, as we just argued that this is what happens in typical runs of our algorithms.

4. RUNTIME ANALYSIS

In this section, we analyze the behavior of Linear GP on the identification problem. From now on, suppose a target function f_{OPT} is given. We will examine how Linear GP starting with any function f will drift towards f_{OPT}.

To do this, we distinguish different regions a, b, c, d of the bit positions, depending on the values of f_{OPT} and f at those positions. Specifically, for all $y, z \in \{-1, +1\}$, we define a function $h_{y,z}$ on functions f such that

$$h_{y,z}(f) = |\{i \le n \mid f_{\text{OPT}}(e_i) = y \wedge f(e_i) = z\}|.$$

Further, we abbreviate $h_{+1,+1}$, $h_{+1,-1}$, $h_{-1,+1}$, $h_{-1,-1}$, respectively, with a, b, c, d, respectively. Table 1 summarizes these definitions. We see that f_{OPT} and f are identical on the regions a and d. On the regions b and c they take on opposite coefficients. Note that $f = f_{\text{OPT}}$ holds iff $b(f) = 0 = c(f)$.

To analyze the behavior of Linear GP it is crucial to examine which changes can be achieved by changing one coefficient of f. In the following, we examine this in detail and demonstrate how a progress towards f_{OPT} can be observed by the random sampling performed in the algorithm.

Let f and f' be two possible solutions that differ in exactly one weight. We think of f' as created from f by flipping exactly one weight. We are carrying out a complete case distinction.

- $a(f) = a(f') - 1$, $b(f) = b(f') + 1$, $c(f) = c(f')$ and

f' (shift)	condition
a-b	$r < s$
b-a	$r > s$
c-d	$r < s$
d-c	$r > s$

Table 2: Impact of possible shifts.

$d(f) = d(f')$. We then call f' an *a-b-shift* (intuitively, one bit position is shifted from region a to region b).

- $a(f) = a(f') + 1$, $b(f) = b(f') - 1$, $c(f) = c(f')$ and $d(f) = d(f')$. We then call f' a *b-a-shift.*
- $a(f) = a(f')$, $b(f) = b(f')$, $c(f) = c(f') - 1$ and $d(f) = d(f') + 1$. We then call f' a *c-d-shift.*
- $a(f) = a(f')$, $b(f) = b(f')$, $c(f) = c(f') + 1$ and $d(f) = d(f') - 1$. We then call f' a *d-c-shift.*

It is easy to see that these cases cover all possibilities, as f_{OPT} does not change. Note that b-a-shifts and c-d-shifts are desirable, while their opposities are not.

Let $x \in \{0,1\}^n$ be a random bitstring sampled by the algorithm. In order to analyze the drift towards f_{OPT}, we consider the difference in error $E = e_x(f, f_{\text{OPT}}) - e_x(f', f_{\text{OPT}})$ of f and f' depending on x. If E is nonnegative, then we prefer f' over f, given the sample x. Clearly, if x has a 0 at the bit position where f and f' differ, then $E = 0$. This happens with probability 1/2. Otherwise, i.e., if x has a 1 at the bit position where f and f' differ, we have $E \in \{-2, +2\}$ as follows.

Suppose x has r many 1s at the bit positions of $b(f)$ and s many 1s in the bit positions of $c(f)$. Then we have $e_x(f, f_{\text{OPT}}) = |2r - 2s|$, as the two different types of errors that can be observed cancel each other out. We distinguish the following cases.

- Suppose f' is an a-b-shift. Then $e_x(f', f_{\text{OPT}}) = |2(r + 1) - 2s|$. Thus we have $E = +2$ if $r < s$, and $E = -2$ otherwise.
- Suppose f' is a b-a-shift. Then $e_x(f', f_{\text{OPT}}) = |2(r - 1) - 2s|$. Thus we have $E = +2$ if $r > s$, and $E = +2$ otherwise.
- Suppose f' is a c-d-shift. Then $e_x(f', f_{\text{OPT}}) = |2r - 2(s - 1)|$. Thus we have $E = +2$ if $r < s$, and $E = -2$ otherwise.
- Suppose f' is a d-c-shift. Then $e_x(f', f_{\text{OPT}}) = |2r - 2(s + 1)|$. Thus we have $E = +2$ if $r > s$, and $E = +2$ otherwise.

The necessary conditions for accepting possible shifts, depending on x having a 1 where f and f' differ, are summarized in Table 2.

Thus in all four cases we have to analyze the probability of sampling an x with strictly more (strictly less) 1s in the $b(f)$ region than in the $c(f)$ region, *conditional on x having a 1 where f and f' differ.* If the position where f and f' differ is in the a- or d-region of f, this conditioning has no effect; however, if it is in the b- or c-region, it guarantees us a 1 where we would have a random coin toss otherwise. In view of this and using that the number of 1s on a given set

f' (shift)	$P(E = +2 \mid x_i = 1)$
a-b	$S_{b(f)} < S'_{c(f)}$
b-a	$S_{b(f)-1} \geq S'_{c(f)}$
c-d	$S_{b(f)} \leq S'_{c(f)-1}$
d-c	$S_{b(f)} > S'_{c(f)}$

Table 3: Overview of Distributions for the Error Terms

of k bits in a random bit string is distributed as S_k, we get Table 3 showing the corresponding conditional probabilities for $E = +2$. (Note that $S_{b(f)-1} \geq S'_{c(f)}$ is equivalent to $1 + S_{b(f)-1} > S'_{c(f)}$.)

We are now able to bound the runtime of the algorithm. Recall that we have to deal with the fact that the fitness values are random variables. We will do so as described in Section 3. Recalling that p_n is $\Theta(1/\sqrt{n})$, we see that the number of samples required is only $z = O((p_n)^{-2} \log n) = O(n \log n)$.

THEOREM 3. *If $|S| \geq c_0 n \log(n)$, c_0 a large enough constant, the expected learning time of Linear GP is $O(n^2)$.*

PROOF. To prove the theorem, we consider a typical run of the algorithm. We show that the algorithm will find the function f_{OPT} with probability at least α, where α is a constant, within Cn^2 generations, C a constant, independently of the starting point of the algorithm. If we have not found f_{OPT} after this number of iterations, we view this as a restart of the algorithm (with the current value of f as the new starting value). As the expected number of such restarts is at most α^{-1}, i.e. constant, this suffices to guarantee an expected number of generations of $O(n^2)$ until f_{OPT} is found.

We analyze the optimization time of Linear GP in two phases, and upper bound the probabilities that undesired events happen during the phases. The first phase ends as soon as, for the current solution f, $|b(f) - c(f)| \leq 1$. We will see that this condition will never be violated once established (with high probability; recall our assumption that the bias is correctly identified in every iteration of Linear GP from Section 3).

Phase 1: We consider a phase of $c_1 n \log n$, where c_1 an appropriate constant, steps and show that this phase is successful with probability $\alpha_1 = 1 - o(1/n)$.

Without loss of generality, assume $b(f) \geq c(f) + 2$. We show that, under the assumption that the bias is always correctly identified, the potential $|b(f) - c(f)|$ never increases, and decreases with probability at least $|b(f) - c(f)|/n$. By a standard coupon-collector argument, this implies that the first phase will finish in $O(n \log n)$ iterations with high probability.

We use Lemma 1 and Table 3 to get the following probabilities for $E = +2$ for the different shifts f', given a sample x.

f' (shift)	$P(E = +2 \mid x_i = 1)$
a-b	$\leq 1/2 - p_n$
b-a	$\geq 1/2 + p_n$
c-d	$\leq 1/2 - p_n/2$
d-c	$\geq 1/2 + p_n/2$

Thus, the undesired events consisting of a-b- and c-d- shifts have a negative bias of at least $p_n/2$ and are therefore never accepted, while the desired b-a-shifts have a positive bias of at least $p_n/2$ and are always accepted. Furthermore, a b-a-shift occurs with probability $b(f)/n \geq |b(f)-c(f)|/n$, which shows that our potential $|b(f) - c(f)|$ is decreased with at least the claimed probability. This completes our argument for Phase 1.

We now show that once a function f with $|b(f)-c(f)| \leq 1$ is reached, the condition $|b(f) - c(f)| \leq 1$ also holds for all subsequent generations. Clearly, if $b(f) = c(f)$, no shift f' will have $|b(f') - c(f')| > 1$. Thus, without loss of generality, assume $c(f) = b(f) + 1$. The only shift f' with $|b(f') - c(f')| > 1$ is an a-b-shift. According to Table 3, this shift has a negative bias of at least $p_n/2$ and is therefore not accepted.

Phase 2: For Phase 2, we consider a phase of $c_2 n^2$ steps, where c_2 an appropriate constant, and show that this phase is successful with probability $\alpha_2 = \Omega(1)$. For $b(f) = c(f)$, we get the following table of transition probabilities.

f' (shift)	$P(E = +2 \mid x_i = 1)$
a-b	$\leq 1/2 - p_n$
b-a	$= 1/2$
c-d	$= 1/2$
d-c	$\leq 1/2 - p_n$

We get a b-a- or c-d-shift with probability $(b(f) + c(f))/n$, and accept such a shift with probability $1/2$ (and reject the other possible shifts).

Without loss of generality, we focus on the case $b(f) = c(f) + 1$, for which we get the following table of transition probabilities.

f' (shift)	$P(E = +2 \mid x_i = 1)$
a-b	$\leq 1/2 - p_n$
b-a	$\geq 1/2 + p_n$
c-d	$\leq 1/2 - p_n/2$
d-c	$= 1/2$

We get (and accept) a b-a-shift with probability $b(f)/n$.

Taking these two tables together, we see that, when $b(f) = c(f)$, neither $b(f)$ nor $c(f)$ can increase, and both can decrease. However, such a decrease might be reversed later: If we start with a c-d-shift, then a d-c-shift has probability $1/2$. However, a b-a-shift would also be accepted (since then $b(f) > c(f)$), leading to $b(f) = c(f)$ again (but both one lower than at the start of the argument). As such a b-a-shift has a probability $\geq b(f)/n$, we will only repeat the initial c-d-shift about $n/b(f)$ times (in expectation) until we decreased both $b(f)$ and $c(f)$. As such a c-d-shift itself only occurs about every $n/b(f)$-th generation, we obtain that in total we have to wait an expected number of $\Theta((n/b(f))^2)$ steps to decrease $b(f) = c(f)$ by 1.

Thus, Phase 2 is done after an expected number of $\leq \sum_{i=1}^{n} \frac{n^2}{i^2} = O(n^2)$ many steps. Using Markov's inequality this phase is successful within $c_2 n^2$ steps, c_2 an appropriate constant, with probability $\alpha_2 = \Omega(1)$.

Both phases are successful with probability $\alpha = \alpha_1 \cdot \alpha_2 = \Omega(1)$ which implies that the expected learning time is upper bounded by

$$\alpha^{-1} \cdot (c_1 n \log n + c_2 n^2) = O(n^2).$$

□

5. APPROXIMATIONS

The results presented so far gave bounds on the expected time until the optimum is found. However, by taking a closer look at the above analysis of the optimization behavior of Linear GP, we also see that Linear GP exhibits a good approximation behavior in asymptotically less time than what is needed for exact identification.

THEOREM 4. *If $|S| \geq c_0 n \log(n)$, c_0 a large enough constant, the expected number of generations until the best-so-far function found by Linear GP has an expected error $\leq \delta$ is $O(n \log n + n^2/\delta^2)$.*

In particular, we can find a function with expected error at most $\sqrt{n/\log(n)}$ in time $O(n \log n)$.

PROOF. We use the notation of the proof of Theorem 3. Note that a function f with $b(f) = c(f) =: t$ has expected error $2\mathbb{E}[S_{b(f)} - S'_{c(f)}] = O(\sqrt{t})$. Thus we have an approximation as desired if we find a function f with $b(f) = c(f) = O(\delta^2)$.

As we saw in the proof of Theorem 3, a time of $O(n \log n)$ suffices to find a function f with $|b(f) - c(f)| \leq 1$.

For Phase 2, we are now interested in the potential $\lceil (b(f)+c(f))/2 \rceil$. As we have seen in the proof of Theorem 3, this potential never increases (with high probability), and decreases with probability $\Omega((b(f)/n)^2)$. Thus by a coupon-collector type argument, the expected time until the potential falls below δ^2 is $O(\sum_{i=\delta^2}^{n} \frac{n^2}{i^2}) = O(n^2/\delta^2)$. This proves the claimed bound on the number of generations until an approximation as desired is found. □

Note that Theorem 4 holds for all choices of f_{OPT}. However, for many choices of f_{OPT}, we get better time bounds for the expected time until a good approximation is reached.

THEOREM 5. *Suppose f_{OPT} has a linear number of weights 1 and -1 each. If $|S| \geq c_0 n \log(n)$, c_0 a large enough constant, the expected number of generations until the best-so-far function found by Linear GP has an expected error $\leq \delta$ is $O(n + n^2/\delta^2)$.*

PROOF. We use the notation of proof of Theorem 3. The first phase of the Linear GP finishes as soon as, for the current best solution f, $|b(f)-c(f)| \leq 1$. From the condition on f_{OPT} we get initial values for $b(f)$ and $c(f)$ linear in n. As the smaller of two does not decrease during Phase 1 (see proof of Theorem 3), Linear GP will (in expectation) lower the potential $|b(f) - c(f)|$ by a constant. Thus, after time linear in n, Phase 1 will be over. The initialization or Phase 1 fails with a sufficiently small probability, such that the result follows in that case from Theorem 4. Otherwise, the remainder of the analysis is as in the proof of Theorem 4 and gives the desired bound. □

The other extreme case for f_{OPT} compared with Theorem 5 is when only constantly many weights are $+1$ (or -1). In this case we also get a fast optimization behavior.

THEOREM 6. *Suppose f_{OPT} has either constantly many weights 1 or constantly many weights -1. If $|S| \geq c_0 n \log(n)$, c_0 a large enough constant, the expected number of generations until the best-so-far function found by Linear GP has an expected error $O(1)$ is $O(n \log n)$.*

PROOF. We use the notation of proof of Theorem 3. We have seen that Phase 1 ends after $O(n \log n)$ steps. From that time on we have $|b(f) - c(f)| \leq 1$. Since one of $b(f)$ and $c(f)$ is bounded above by a constant (from the condition of f_{OPT}), we have that both are bounded above by a constant. Now even the worst case error is bounded above by a constant. $\square$

6. CONCLUSIONS

Genetic Programming is a successful type of learning algorithm. Understanding GP in a rigorous way is a hard and challenging task. With this paper, we contributed to its theoretical understanding by analyzing it within the PAC learning framework for a simple type of linear function. We have pointed out how GP can observe differences in the quality of functions by random sampling. This insight leads to computational complexity bounds for learning the optimal function by Linear GP. Furthermore, we analyzed the approximation behavior of the algorithm and pointed out that good approximations can be achieved within even better runtime guarantees.

Future work should consider variants of our algorithm that are allowed to flip more than one weight in each iteration. Note that our analysis relied on the fact that always exactly one weight is flipped. Allowing more weights to be flipped in each iteration requires additional insights into the sampling process. We are optimistic that such results can be obtained in the near future. Another topic for future research is to extend the analysis to classes of functions where the coefficients can take on more than two values. Again, such results would be very valuable and should lead to new theoretical insights for genetic programming in the PAC learning framework.

7. REFERENCES

[1] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[2] G. Durrett, F. Neumann, and U.-M. O'Reilly. Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics. In *Proc. of FOGA'11*, pages 69–80. ACM, 2011.

[3] V. Feldman. Evolvability from learning algorithms. In *Proc. of STOC'08*, pages 619–628. ACM, 2008.

[4] V. Feldman. A complete characterization of statistical query learning with applications to evolvability. In *Proc. of FOCS'09*, pages 375–384. IEEE, 2009.

[5] C. Ferreira. *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, volume 21 of *Studies in Computational Intelligence*. Springer, 2006.

[6] D. E. Goldberg and U.-M. O'Reilly. Where does the good stuff go, and why? How contextual semantics influences program structure in simple genetic programming. In *EuroGP'98*, pages 16–36. Springer, 1998.

[7] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[8] I. Kushchu. Genetic programming and evolutionary generalization. *IEEE Trans. Evolutionary Computation*, 6:431–442, 2002.

[9] H. Mühlenbein. How genetic algorithms really work: mutation and hillclimbing. In *Proc. of PPSN'92*, pages 15–26. Elsevier, 1992.

[10] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[11] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. lulu.com, 2008.

[12] R. Poli, L. Vanneschi, W. B. Langdon, and N. F. McPhee. Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11:285–320, 2010.

[13] G. Rudolph. *Convergence properties of evolutionary algorithms*. Hamburg: Kovac, 1997.

[14] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27:1134–1142, 1984.

[15] L. G. Valiant. Evolvability. *J. ACM*, 56, 2009.