On the Analysis of Simple Genetic Programming for Evolving Boolean Functions

Andrea Mambrini and Pietro S. Oliveto(✉)

University of Sheffield, Sheffield, UK
{a.mambrini,p.oliveto}@sheffield.ac.uk

Abstract. This work presents a first step towards a systematic time and space complexity analysis of genetic programming (GP) for evolving functions with desired input/output behaviour. Two simple GP algorithms, called (1+1) GP and (1+1) GP*, equipped with minimal function (F) and terminal (L) sets are considered for evolving two standard classes of Boolean functions. It is rigorously proved that both algorithms are efficient for the easy problem of evolving conjunctions of Boolean variables with the minimal sets. However, if an extra function (i.e. NOT) is added to F, then the algorithms require at least exponential time to evolve the conjunction of n variables. On the other hand, it is proved that both algorithms fail at evolving the difficult parity function in polynomial time with probability at least exponentially close to 1. Concerning generalisation, it is shown how the quality of the evolved conjunctions depends on the size of the training set s while the evolved exclusive disjunctions generalize equally badly independent of s.

Keywords: Genetic programming · Theory · Runtime analysis

1 Introduction

Genetic programming (GP) was originally proposed by Koza as an evolutionary computation technique for evolving computer programs [4]. Traditionally GP represents programs using *syntax trees* and evaluates their fitness by executing them and then comparing their behaviour against an ideal one (eg., the desired input/output behaviour of the function to be evolved). A population of programs is evolved using typical genetic algorithm (GA) variation and selection operators adapted to work on syntax trees with the goal of eventually identifying a program with the desired functionality. GP has shown great potential by evolving, for example, quantum computing algorithms that outperformed all previous approaches [17], soccer-playing programs [7] and algorithms for the transmembrane segment identification protein problem [5], to name a few.

Despite the wide range of successful applications, there is still very little understanding of GP's behaviour [13,15]. Theoretical work concerning GP has always been undertaken since its early days [6], the majority of which has applied *schema theory* [4,16]. Schema theories are based on the idea of partitioning

M. Heywood et al. (Eds.): EuroGP 2016, LNCS 9594, pp. 99–114, 2016.
DOI: 10.1007/978-3-319-30668-1_7

the search space into subsets, called *schemata*, and modelling the behaviour and dynamics of the population over the schemata. However, such an analysis does not allow any insight towards the understanding of the performance of GP. Chapter 11.1 of [15] concludes that through schema theories *"...we have no way of closing the loop and fully characterising the behaviour of GP systems which is always a result of the interaction between the fitness function and the search biases of the representation and genetic operations used in the system"*. Such characterisations and interactions can, instead, be understood by analysing the time and space complexity of a GP system when attempting to evolve a given class of functions. This approach has been applied to other classes of bio-inspired optimisation heuristics, with remarkable success [1]. Nowadays, the performance quality of various bio-inspired optimisation heuristics is known concerning sophisticated population-based heuristics and even for standard combinatorial optimisation problems with practical applications [11]. These results shed light on which kind of problems a given algorithm works on efficiently and on which it is inefficient and provide a relationship between the size of the problem and the time and space required to solve it. Along the way guidelines towards optimal parameter settings are given.

Some initial runtime analysis results concerning GP systems have already appeared [10]. Such first studies regarded two functions classes called ORDER and MAJORITY where the fitness of a tree (i.e., a candidate solution) depends on the structure of the tree rather than on its execution. Although this is a considerable simplification compared to the problems to which GP is usually applied, these results show that very simple GP systems can optimise both structures efficiently. Furthermore, understanding how and when correct structures are evolved will be necessarily crucial in an analysis of more realistic GP scenarios. Recently, the same simple GP systems have been analysed on the MAX Problem [6] where, given a set of functions, a set of terminals and a bound D on the maximum depth of the solution, the goal is to evolve a tree that returns the maximum value given any combination of functions and terminals [3]. The analysis shows that the simple GP systems can efficiently evolve MAX with function set $F = \{+, *\}$ and one constant as terminal set. Compared to the previous functions, MAX is more similar to those evolved by GP in practical applications since the fitness indeed depends on the behaviour of the computed function on the input. Still, dependence is not very strong, since the space of possible inputs can be partitioned into just two subsets such that for every input in a subset, the optimal solution to the problem is the same.

In this paper we make a further step forward and provide an analysis of GP for typical benchmark functions used in the field of GP [4,6]. Hence, we consider proper learning problems where the fitness depends on the input/output behaviour of the trees. When the initial foundations for a systematic time complexity analysis of EAs were being set, very simple EAs were considered (eg., the (1+1) EA) for simple benchmark problems which are easy for EAs (eg., OneMax and LeadingOnes) and others which are hard (eg., Trap Functions and Needle-In-A-Haystack) [2]. In a similar fashion we will analyse the simple and minimalistic (1+1) GP considered in previous runtime analyses of GP [3,10] for simple Boolean functions with minimal function and terminal sets. Since under the *evolvability* notion in the PAC-learing framework it is well-understood that conjunctions (i.e., AND) are evolvable efficiently while parity problems (i.e., XOR) are not [18], we naturally choose these boolean functions as our starting points for the analysis. In particular, the presented AND problem may be regarded as a GP analogue to OneMax for EAs, while XOR the analogue to Needle. Moreover, we will take into consideration the generalization ability of the solutions found by the algorithms when using incomplete training sets since, as the problem size grows, it is not possible to test the candidate solutions on the complete training set efficiently. We point out that runtime results are available concerning the recently introduced GP variant called Geometric Semantic Genetic Programming (GSGP) [9]. The long term aim of the work presented herein is to understand the behaviour and performance of standard and widely used GP systems.

In the next section we introduce the two simple (1+1) GP and (1+1) GP* systems and formally define the learning problems. In Sect. 3 we present the results for the AND and XOR functions of n variables using the complete training set and minimal function and terminal sets (i.e., respectively $F = \{AND\}, L = \{X_1, \ldots, X_n\}$ and $F = \{XOR\}, L = \{X_1, \ldots, X_n\}$). In particular, we show that both the (1+1) GP and the (1+1) GP* can evolve conjunctions efficiently while they are both inefficient when evolving parity. However, if we add another function to F (i.e., NOT), then the algorithms become inefficient also for evolving conjunctions. In Sect. 4 we present the results when only a training set of polynomial size in the problem size is allowed. We show that the algorithms fit the training set for the AND function in logarithmic time while the XOR function becomes harder than a Needle function for larger than logarithmic training sets because points leading closer to the optimal solution may be rejected. We conclude the section by providing results on how the evolved solutions generalise to the complete training set. In the Conclusions we discuss future work directions.

2 Preliminaries

We consider the (1+1)-GP (Algorithm 1) and (1+1)-GP* (Algorithm 2) from [10], both working with a population of size one and producing at each generation one new offspring using the HVL-Prime mutation operator [14] which chooses, uniformly at random, to either insert, to remove or to replace a node according to the procedures described in Algorithm 3.

Algorithm 1. (1+1) GP
```
1: Initialise an empty tree X
2: for $t := 1$ to $\infty$ do
3:     $X' :=$ HVL-Prime($X$)
4:     if $f(X') \leq f(X)$ then
5:         $X := X'$
```

Algorithm 2. (1+1) GP*
```
1: Initialise an empty tree X
2: for $t := 1$ to $\infty$ do
3:     $X' :=$ HVL-Prime($X$)
4:     if $f(X') < f(X)$ then
5:         $X := X'$
```

The only difference between (1+1) GP and (1+1)-GP* is that the former accepts an offspring which is at least as fit as its parent, while the latter accepts only strictly better offspring. An individual X is represented as a binary tree such that each internal node can be an element of the function set F and each leaf can be an element of the terminal set L. The two algorithms do not have a termination criterion since, here, we are interested in the first point of time when the optimal solution is found. For simplicity we initialise the algorithms with empty trees[1]. However, all the presented results can be easily adapted to random tree initialisation with only slightly differing theorem statements.

Let the complete truth table $C = \{(x_1, y_1), ..., (x_N, y_N)\}$ describe the complete input-output behaviour of a Boolean function $\hat{h} : \{0,1\}^n \rightarrow \{0,1\}$ over n variables (i.e., the table has $N = 2^n$ rows). A training set T consisting of $s \leq N$ test cases $T \subset C = \{(x_1, y_1), ..., (x_s, y_s)\}$ is sampled from the truth table uniformly at random with replacement. The black box Boolean learning problem consists of using just the training set T to learn a Boolean function $h : \{0,1\}^n \rightarrow \{0,1\}$ matching as well as possible the input-output behaviour described by C. Given a candidate solution (i.e., a Boolean expression h), the fitness function returns the *training error* $\epsilon_t(h)$ which is the number of rows on which the expression h mismatches the input-output behaviour described by T: $\epsilon_t(h) = \sum_{(x_i,y_i)\in T} I[h(x_i) \neq y_i]$ where $I[\cdot]$ is the indicator function that is 1 if its inner expression is true and 0 otherwise. We impose that the fitness function returns a value of $2^n + 1$ for an empty tree, which is worse than the fitness of any tree.

Algorithm 3. HVL-Prime

```
1: procedure HVL-Prime($X$)
2:     Select uniformly a random an action among INS, DEL, SUB
3:     if action is INS then
4:         Choose a node $v \in X$ uniformly at random
5:         Select uniformly at random a terminal $v'$ from $L$
6:         Replace $v$ with a node $f$ selected uniformly at random from $F$
7:         Set $v$ and $v'$ as children of $f$, choosing the order of the children uniformly
   at random.
8:     if action is DEL then
9:         Choose, uniformly at random, a leaf node $v$ with parent $p$ and sibling $s$
10:        Replace $p$ with $s$
11:        Delete $p$ and $v$
12:    if action is SUB then
13:        Choose a leaf $v$ uniformly at random
14:        Select uniformly at random a terminal $v'$ from $L$
15:        Replace $v$ with $v'$
```

[1] We assume the SUB and DEL of an empty tree return an empty tree.

We will analyse the (1+1) GP and (1+1) GP* on two boolean problems, AND_n with target function $\hat{h} = AND(X_1, \ldots, X_n)$ and XOR_n with target function $\hat{h} = XOR(X_1, \ldots, X_n)$. We say that an algorithm *solves* a boolean problem *efficiently* if it can evolve a solution fitting the training set (i.e., having training error equal to zero) in expected polynomial time, where time is defined as the number of fitness function evaluations.

We will first analyse the situation in which $s = N$, thus the training set encompasses all the possible input-output cases (i.e. complete dataset). In this situation finding an expression that fits the training set T will obviously also lead to an expression that fits the complete set, and thus the original Boolean function. Afterwards we will consider training sets of at most polynomial size, $s = \text{poly}(n) < N$ (i.e. incomplete training set). In this case, minimizing the error on the training set will lead to a generalization error which can be defined as $\epsilon_g(h) = \sum_{(x_i,y_i)\in C} I[h(x_i) \neq y_i]$ where C is the complete truth table.

We define the *generalization ability* of an algorithm $\mathcal{A}$ as $G(\mathcal{A}, s) = 1 - \frac{E[\epsilon_g(\widetilde{X})]}{N}$ where $\widetilde{X}$ is the best individual found by the algorithm (which tries to minimize the error on the training set) after a polynomial number of steps.

3 Analysis for Complete Training Sets

In this section we analyse the (1+1) GP and the (1+1) GP* on AND_n and XOR_n in the case of training sets of size $s = 2^n = N$ (i.e. complete datasets).

3.1 Analysis for AND_n with Complete Training Sets

Theorem 1 shows that both the (1+1)-GP and the (1+1) GP* evolve AND_n efficiently with $L = \{X_1, \ldots, X_n\}$ and $F = \{AND\}$. The theorem also shows that the strict selection of (1+1) GP* enforces solutions of exactly n variables while the (1+1) GP may produce solutions that are asymptotically larger by a logarithmic factor. The following Lemma will be useful.

Lemma 1. *Every conjunction of v distinct variables differs from the target function $AND(X_1, \ldots, X_n)$ on $f_v = 2^{n-v} - 1$ rows.*

Proof. We prove the statement by induction. The base case $f_1 = 2^{n-1} - 1$ follows because the truth table of any conjunction of one variable has $2^n/2$ ones and $2^n/2$ zeros. The target function and the conjunction of one variable will agree on one row, i.e. the one in which all the variables are set to 1. On the rest of the $2^{n-1} - 1$ rows the two functions will not agree.

To prove the inductive step we need to assume $f_i = 2^{n-i} - 1$ and prove that $f_{i+1}(x) = 2^{n-(i+1)} - 1$. When one variable is added to the conjunction, the number of ones will halve (since half of them will be anded with a zero). Since just one of them agrees with the target function, the new conjunction will differ on $f_{i+1} = (f_i + 1)/2 - 1$, which by hypothesis is $2^{n-(i+1)}$. □

Theorem 1. *The (1+1)-GP and the (1+1) GP* using $F = \{AND\}$ and $L = \{X_1, \ldots, X_n\}$ efficiently solve AND_n, using the complete truth table as training set, in time $O(n \log n)$. The size of the final expression is n for (1+1) GP* and $O(n \log n)$ in expectation for (1+1)-GP.*

Proof. We divide the search space into $A_1, \ldots, A_n$ fitness levels such that each level A_i contains all the conjunctions of i distinct variables. By Lemma 1 we have a fitness level partition with increasing fitness and the artificial fitness levels technique [1] is in force. We need to derive a lower bound on the probability p_i that an individual leaves level A_i and reaches level A_{i+1}.

For both (1+1) GPs at level A_i the number of distinct i variables may not be reduced as the fitness would decrease. Hence to reach level A_{i+1} it is sufficient for both algorithms to choose an INS operation and then insert one of the $n-i$ variables that is not already in the conjunction. This probability is $p_i \geq \frac{1}{3}\frac{n-i}{n}$. Thus, by the fitness level method the expected runtime is $E[T] \leq \sum_{i=1}^{n} 1/p_i = \sum_{i=1}^{n} \frac{3n}{n-i} = O(n \log n)$, which proves the first statement.

For the (1+1) GP* (strict selection) an INS operation adding a duplicate of an existing variable will not be accepted since it would have the same fitness of the parent. The same holds for a SUB operation replacing one variable with a duplicate of an existing one. Thus at each iteration the current solution has each variable appearing at most once and, for this reason, a DEL operation will never be accepted, since it would reduce the fitness. Since no duplicates are allowed in the (1+1) GP*, the size of the final expression is exactly n. For the (1+1) GP (weak selection) the situation is different because an INS operation can insert a variable that already exists in the current expression. We observe that the only operation increasing the size of the current expression is INS. Since in expectation $O(n \log n)$ of these operations occur, the expected size of the final expression cannot be more than $O(n \log n)$. $\square$

In the following theorem we show that just adding the negation of all the variables to the terminal set makes both algorithms inefficient.

Theorem 2. *Both the (1+1)-GP and the (1+1)-GP* using $F = \{AND\}$ and $L = \{X_1, \ldots, X_n, \overline{X_1}, \ldots, \overline{X_n}\}$ cannot solve AND_n with probability at least $p > 1 - \left(\frac{1}{4}\right)^{n/3}$ using the complete truth table as training set.*

Proof. Differently from the situation of Theorem 1, the search space contains many local optima from which the algorithms cannot escape. All solutions containing both a variable X_i and its negation $\overline{X_i}$ evaluate to 0, hence have a fitness of 1. We prove that such a solution is found with probability exponentially close to 1. When the current expression is missing just $i \leq n/3$ variables the probability of adding a new variable is bounded from above by the probability of doing so with an INS operation plus the probability that a SUB operation adds a missing variable, which is $p_{add} \leq \frac{1}{3}\frac{n/3}{2n} + \frac{1}{3}\frac{n/3}{2n} = 1/9$. At the same time the probability of inserting the negation of a variable which is already in the current expression is at least $p_{neg} \geq \frac{(2/3)n}{2n} = 1/3$. The probability of the second event happening before the first is $P(neg|neg \cup add) \geq \frac{p_{neg}}{p_{neg}+p_{add}} \geq \frac{1/3}{1/3+1/9} = 3/4$. The probability that neg never happens before all the $n/3$ missing variables are added is $p \leq (1-3/4)^{n/3} = \left(\frac{1}{4}\right)^{n/3}$, thus the algorithm ends up in one of these local optima with probability $p > 1 - \left(\frac{1}{4}\right)^{n/3}$.

Since a deletion of $\overline{X_i}$ decreases fitness, the only way to leave such local optima is by adding all the missing variables and then removing $\overline{X_i}$ (plus all the other negated variables that might have been inserted in the process). In the case of strict selection, this is not possible and the (1+1) GP* is stuck forever. For the case of weak selection, the algorithm will have to walk on a plateau of fitness 1 until it reaches a point when all the n variables are in the tree. Only at that point it would be allowed to walk on another plateau by removing all the negated variables. When this happens the optimum would be found. Similarly to the proof of Theorem 4 it is possible to show that this cannot happen in less than exponential time with probability exponentially close to 1. We don't report a complete proof here due to space restrictions. □

3.2 Analysis for XOR_n with Complete Training Sets

The analysis for XOR_n will show a needle-like fitness landscape (Proposition 2). In fact when the training set encompasses all the possible input-output pairs, all the solutions in the search space but the optimum will have the same fitness. As a result, we will show in Theorem 3 that the (1+1) GP* cannot solve XOR_n in finite time because it cannot find strictly improving solutions and in Theorem 4 that the (1+1) GP cannot solve XOR_n in less than exponential time with probability exponentially close to 1. We state the two following facts.

Proposition 1. *Any exclusive disjunction of m variables ($X_1 \oplus \ldots \oplus X_m$) on a truth table of $n \geq m$ variables outputs* 1 *for half of its inputs and* 0 *for the other half.*

Proposition 2. *Any exclusive disjunction of m variables ($X_1 \oplus \ldots \oplus X_m$) on a truth table of $n \geq m$ variables differs from $X_1 \oplus \ldots \oplus X_n$ on 2^{n-1} inputs, thus has fitness 2^{n-1}.*

Now we are ready to state and prove the theorems.

Theorem 3. *The (1+1)-GP* using $F = \{XOR\}$ as function set and $L = \{X_1, \ldots, X_n\}$ as terminal set cannot solve XOR_n, using the complete truth table as training set.*

Proof. Since by Proposition 2 all the points in the search space have the same fitness, any individual after the first one will not be accepted. Thus XOR_n cannot be solved using strict selection. □

The following Lemma will be useful in the proof of Theorem 4.

Lemma 2. *In a tree containing m leaves, each one sampled uniformly at random from $L = \{X_1, \ldots, X_n\}$ with replacement, with probability $p \geq 1 - e^{-\Omega(n)}$ each variable X_i appears at most $M < \frac{m \log n}{n}$ times for $m \geq \frac{n^2}{e}$, and at most $M < \frac{2n}{\ln\left(\frac{n^2}{em}\right)} < 2n$ times for $m < \frac{n^2}{e}$.*

Proof. We will bound M from above by using a balls and bins argument [8], where the m balls represent the total number of variables in the tree and each of the n bins represents a different variable X_i. The probability that $M > \gamma$ is the probability that after throwing m balls into n bins by selecting each bin uniformly at random, there is at least one bin containing at least γ balls. Let X be a random variable counting the number of balls in a given bin and $P(X > \gamma)$ be the probability that one bin contains more than γ balls. Then by the union bound $P(M > \gamma) \leq n \cdot P(X > \gamma)$. We will calculate $P(X > \gamma)$ for two separate cases.

(1) Let $m \geq \frac{n^2}{e}$. Since the probability of selecting a bin is $1/n$, the expected number of balls in the bin is $\mu = E(X) = m/n$. Applying the Chernoff bound $P(X > (1+\delta) \cdot \mu) \leq [\frac{1}{1+\delta}(\frac{e}{1+\delta})]^{\mu} \leq (\frac{1}{1+\delta})^{\mu}$ for any $\delta > 0$ [8] and by exploiting $m \geq \frac{n^2}{e}$ we get:

$$P\left(X > \frac{m \log n}{n}\right) \leq \left(\frac{1}{\log n}\right)^{\frac{m}{n}} \leq \left(\frac{1}{\log n}\right)^{\frac{n}{e}} \leq \left(\frac{1}{e}\right)^{\frac{n}{e}} = e^{-\Omega(n)}$$

where the inequality before the last one holds for $n > e^e$. Thus $M < \frac{m \log n}{n}$ with probability at least $1 - ne^{-\Omega(n)} = 1 - e^{-\Omega(n)}$.

(2) Let $m < \frac{n^2}{e}$. By applying the Chernoff bound again with $\tau = 1 + \delta$ we get for any $\tau > 1$

$$P(X > \tau \cdot \mu) \leq \left(\frac{e}{\tau}\right)^{\tau\mu}$$

We want τ such that $P(X > \tau \cdot \mu) \leq e^{-n}$ which is equivalent to

$$\left(\frac{e}{\tau}\right)^{\tau\mu} \leq e^{-n} \iff e^n \leq \left(\frac{\tau}{e}\right)^{\tau\mu} \iff e^{\frac{n}{e\mu}} \leq \left(\frac{\tau}{e}\right)^{\frac{\tau}{e}} \iff e^{\frac{n^2}{e \cdot m}} \leq \left(\frac{\tau}{e}\right)^{\frac{\tau}{e}}$$

Now, let $K = \tau/e$ and $N = e^{\frac{n^2}{e \cdot m}}$. Since $K > \frac{2 \ln N}{\ln \ln N}$ implies $K^K > N$ provided that $N > e$ which holds since $m < n^2/e$, we get that $\left(\frac{\tau}{e}\right)^{\frac{\tau}{e}} \geq e^{\frac{n^2}{e \cdot m}}$ holds for

$$\frac{\tau}{e} > \frac{2 \ln(e^{\frac{n^2}{em}})}{\ln \ln(e^{\frac{n^2}{em}})} \iff \tau > \frac{2n}{\mu \ln(\frac{n^2}{em})}$$

Hence $\tau > 1$ for $m < 2n^2$ and we can apply the Chernoff bound. Thus,

$$P\left(X \geq \frac{2n}{\ln(\frac{n^2}{em})}\right) \leq e^{-n}$$

So $M \leq \frac{2n}{\ln(\frac{n^2}{em})} < 2n$ with probability at least $1 - ne^{-n}$. □

Theorem 4. *The (1+1)-GP using $F = \{XOR\}$ and $L = \{X_1, \ldots, X_n\}$ to evolve XOR_n using the complete truth table as training set requires more than $2^{\Omega(\frac{n}{\log n})}$ steps with probability $p > 1 - 2^{-\Omega(\frac{n}{\log n})}$ to reach the optimum.*

Proof. We apply the simplified negative drift theorem [12]. Let k_t denote the number of missing variables in our current solution after simplification[2] (thus e.g. for $X_1 \oplus X_4 \oplus X_1$, $k_t = n - 1$) at step t. Given an expression with i variables missing after simplification we denote with $E[\Delta(i)] = E[(k_{t+1} - k_t)|k_t = i)]$ the negative drift, which is the expected increase in the number of missing variables after simplification in the next step. Since each of the operations (INS, DEL, SUB) happens with equal probability, $E[\Delta(i)] = \frac{1}{3}E[\Delta_{\text{INS}}(i)] + \frac{1}{3}E[\Delta_{\text{DEL}}(i)] + \frac{1}{3}E[\Delta_{\text{SUB}}(i)]$.

When an INS operation occurs $E[\Delta_{\text{INS}}(i)] \geq \frac{n-i}{n} - \frac{i}{n}$ since we decrease k_t by one when we insert a variable which was not in the current solution and increase it by one when we add a variable that was already there (because simplification would remove it). For $i \leq \frac{1}{4}n$, $E[\Delta_{\text{INS}}(i)] > 1/2$.

When a DEL operation occurs we notice that the number of missing variables after simplification increases if we delete one of the variables appearing an odd number of times in the expression (thus not being simplified out), while we decrease the number of missing variables if we delete a variable appearing an even number of times (because it would not be simplified out anymore). Thus calling m the number of leaves in the tree (i.e. the number of variables before simplification), and $M = \max_i [\text{count}(X_i)]$ the maximum number of occurrences of a variable, we observe pessimistically that $E[\Delta_{\text{DEL}}(i)] \geq \frac{m-M\cdot i}{m} - \frac{M\cdot i}{m} = \frac{m-2\cdot M\cdot i}{m}$. The conditional drift when $m \geq \frac{n^2}{e}$ (thus $M < \frac{m\log n}{n}$ with probability exponentially close to 1 by Lemma 2) is

$$E[\Delta_{\text{DEL}}(i)] \geq \frac{m - 2\cdot M\cdot i}{m} \geq \frac{m - 2\cdot \frac{m\log n}{n}\cdot i}{m}$$

which is positive for $i < \frac{n}{2\log n}$ On the other hand the conditional drift when $m < \frac{n^2}{e}$ (thus $M < 2n$ with probability exponentially close to 1) is

$$E[\Delta_{\text{DEL}}(i)] \geq \frac{m - 2\cdot M\cdot i}{m} \geq \frac{m - 4n\cdot i}{m}$$

which is positive for $i < \frac{m}{4n} < \frac{n}{4e}$ (since $m < n^2/e$).

When a SUB operation occurs we notice that the number of missing variables after simplification increases by two if we replace one of the variables appearing an odd number of times in the expression with another different variable appearing an odd number of times, while it decreases by two if we replace one of the variables appearing an even number of times in the expression with one different variable appearing an even number of times. In all other cases the number of

[2] Simplification is a conceptual tool used for the proofs. The actual tree contains all the variables (i.e., the algorithm does not simplify the trees).

missing variables stays the same. Thus $E[\Delta_{\mathrm{SUB}}(i)] \geq 2\cdot\frac{m-Mi}{m}\cdot\frac{n-i-1}{n} - 2\cdot\frac{Mi}{m}\cdot\frac{i-1}{n}$. For $i \leq n/2$ [thus $\frac{n-i-1}{n} \geq \frac{i-1}{n}$] we get

$$E[\Delta_{\mathrm{SUB}}(i)] \geq \frac{2(i-1)}{n}\left[\frac{m-Mi}{m} - \frac{Mi}{m}\right] = \frac{2(i-1)}{n}\left[\frac{m-2Mi}{m}\right]$$

Now we check the two cases of Lemma 2. For $m \geq \frac{n^2}{e}$, we get

$$E[\Delta_{\mathrm{SUB}}(i)] \geq \frac{2(i-1)}{n}\left[\frac{m - 2\cdot\frac{m\log n}{n}\cdot i}{m}\right]$$

For $2 \leq i \leq \frac{n}{4\log n}$ we get $E[\Delta_{\mathrm{SUB}}(i)] \geq \frac{1}{n}$. For $m < \frac{n^2}{e}$, we get

$$E[\Delta_{\mathrm{SUB}}(i)] \geq \frac{2(i-1)}{n}\left[\frac{m - 2\cdot 2n\cdot i}{m}\right]$$

For $2 \leq i \leq \frac{m}{8n} < \frac{n}{8e}$ (since $m < \frac{n^2}{e}$) we get $E[\Delta_{\mathrm{SUB}}(i)] \geq \frac{1}{n}$.

Finally, choosing $a = \frac{n}{8\log n}$ and $b = \frac{n}{16\log n}$, the expected negative drift $E[\Delta(i)] = \frac{1}{3}(E[\Delta_{\mathrm{INS}}(i)] + E[\Delta_{\mathrm{DEL}}(i)] + E[\Delta_{\mathrm{SUB}}(i)]) \geq (1/3)[1/2 + 0 + \frac{1}{n}] \geq 1/6 + o(1)$ for $i \in (a,b)$. Since the probability of performing steps greater than 2 is 0 and $p(\Delta(i) = 2) \leq 1/3 \leq (1/2)^{2-r}$ for $r = 1$ the drift theorem is in force. Thus, conditional to the failure probabilities of Lemma 2, the optimum is found in time $T < 2^{\frac{n}{16\log n}}$ with probability $2^{-\Omega\left(\frac{n}{\log n}\right)}$. By the union bound the probability that the bounds on M do not hold in $2^{\frac{n}{16\log n}}$ steps is less than $2^{\frac{n}{16\log n}} \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$. Summing up the failure probabilities completes the proof. □

4 Analysis for Incomplete Training Sets

In this section we consider training sets of at most polynomial size, $s << 2^n$. The algorithm will thus calculate the fitness just on $s = poly(n)$ rows. We say that an algorithm *efficiently solves* a boolean problem if it can find a solution fitting the training set (i.e. with training error equal to zero) in expected polynomial time. We first analyse the time the algorithms takes to get to a solution with fitness zero on the training set and afterwards we will analyse the generalization error.

4.1 Analysis for AND_n with Incomplete Training Set

The analysis for AND_n shows that a polynomial training set is fit in logarithmic time (Theorem 5). Theorem 6 gives an upper bound on the generalization ability and gives a necessary condition on the size of the training set to achieve a generalization ability over a fixed threshold.

Theorem 5. *Let $s = poly(n)$ be the size of a training set chosen from the truth table uniformly at random with replacement. Then both the (1+1) GP and the (1+1) GP* using $F = \{AND\}$ and $L = \{X_1, \ldots, X_n\}$ will solve AND_n in expected time $O(\log s) = O(\log n)$.*

Proof. Since the elements of the truth table are binomially distributed with parameters n and $p = 1/2$, the expected number of 1 s in a randomly chosen row of the training set is $n/2$. By a simple application of Chernoff bounds, the probability that more than $Y = n/2 + \epsilon n$ 1s are in the chosen element is bounded from above by $e^{-\Omega(n)}$. By a union bound the probability that any of the s elements of the training set contains more than Y 1s is less than $s \cdot e^{-\Omega(n)} = poly(n) \cdot e^{-\Omega(n)} = e^{-\Omega(n)}$. The algorithm will reach the minimum error (i.e. fitness) of 0 when for each row of the training set there exists a variable X_i in the constructed tree that has a 0 in that row. In this case the AND of X_i with itself or any other variable will return 0. We call a step *successful* if in that step the algorithm adds a new unseen terminal to the tree. For strict selection such a terminal is accepted if it reduces the error in the training set while for non-strict selection it is always accepted as the fitness cannot decrease. The other operators SUB and DEL do not contribute to fitness since DEL may only remove redundant terminals in the non-strict selection version and SUB may only exchange terminals that do not decrease fitness. If a terminal X_i is exchanged for a terminal X_j leading to an improvement (i.e., more rows of the training set have a 0) this may only speed up the algorithm. We consider a phase of k successful steps of the (1+1)-GP and calculate the probability that at the end of the k steps not even one variable with value 0 in a given row of the training set has been added. This probability is the probability that the first k terminals that are selected to be added to the tree have value 1 in the chosen row. Since for each row of the training set there are at most Y 1s, the probability that a terminal has a 1 in that position is at most Y/n and the probability that k consecutive terminals are all 1 s in that position is bounded from above by $(Y/n)^k$. By the union bound, the probability that in any of the s rows of the training set all the k terminals have values 1 is less than $p_s = s \cdot (Y/n)^k$. We calculate the value of p_s after a phase of length $k = \log_{\frac{n}{Y}}(2s)$: $p_s \leq s \cdot \left(\frac{Y}{n}\right)^k = s \cdot \left(\frac{Y}{n}\right)^{\log_{\frac{n}{Y}}(2s)} = s \cdot \frac{1}{2s} = \frac{1}{2}$. Hence with probability at least 1/2, after $k = \log_{\frac{n}{Y}}(2s)$ successful steps, for each row of the training set the constructed tree contains at least one terminal X_i that has value 0 in that row. This implies that, in expectation, 2 phases of length k each are sufficient to reduce the error in the training set to 0. All that remains to be done is to calculate the expected time for $2k$ successful steps. Given that the tree consists of i different terminals, the probability that the (1+1)-GP adds a new terminal to the tree is $p_i = \frac{1}{3}\frac{n-i}{n}$ where 1/3 is the probability that an INS operation is chosen. Hence, by a simple coupon collector argument the expected time to collect the AND of $2k = 2\log_{\frac{n}{Y}}(2s)$ different terminals is bounded by:

$$\sum_{i=0}^{2\log_{\frac{n}{Y}}(2s)-1} \frac{3n}{n-i} \leq \sum_{i=0}^{2\log_{\frac{n}{Y}}(2s)-1} \frac{3n}{n-(2\log_{\frac{n}{Y}}(2s)-1)}$$

$$\leq \sum_{i=0}^{2\log_{\frac{n}{Y}}(2s)-1} \frac{3n}{cn} = O(\log_{\frac{n}{Y}}(2s)) = O(\log n)$$

Finally, we need to remember that the above expected time is conditional to starting with at most Y ones in each row of the training set. This may not happen with probability at most $e^{-\Omega(n)}$. If this is the case we pessimistically assume that a row of the training set consists of its worst case value of $n-1$ 1s and one 0. Then, by the same coupon collector argument used in Theorem 1, the conditional expected runtime is bounded from above by $O(n \log n)$. The statement of the theorem now follows by an application of the law of total expectation:

$$\begin{aligned} E(T) &= p(X) \cdot E(T|X) + p(\overline{X}) \cdot E(T|\overline{X}) \\ &\le (1-e^{-\Omega(n)}) \cdot O(\log n) + e^{-\Omega(n)} \cdot O(n \log n) = O(\log n) \end{aligned}$$

□

Theorem 6. *Both the (1+1) GP and the (1+1) GP* using $F = \{AND\}$, $L = \{X_1, \dots, X_n\}$ and a training set of size $s = poly(n)$, have a generalization ability on AND_n of $G \le 1 - 2^{-\log(s)}$.*

Proof. Recall that the generalization error $\epsilon_g(h)$ of an expression h is the number of rows mismatching the target function. In the case of AND, $\epsilon_g(h) \ge \text{ones}(h) - 1$, since all but at most one 1 s are mismatched.

Theorem 5 states that the algorithms will stop at an expression $\tilde{h}$ having at most $\log(s)$ variables, thus having at least $2^{n-\log(s)}$ ones. Thus $\epsilon_g(\tilde{h}) \ge 2^{n-\log(s)} - 1$. The generalization ability is then $G = 1 - \frac{\epsilon_g(\tilde{h})}{2^n} \le 1 - 2^{-\log(s)}$. □

Corollary 1. *A necessary condition to get a generalization ability greater than $1-\epsilon$, is to have a training set of size $s > \frac{1}{\epsilon}$.*

Proof. From Theorem 6 the generalization ability is $G \le 1 - 2^{-\log(s)}$. Thus:

$$1 - 2^{-\log(s)} > 1 - \epsilon \Leftrightarrow 2^{-\log(s)} < \epsilon \Leftrightarrow 1/s < \epsilon \Leftrightarrow s > 1/\epsilon$$

□

4.2 Analysis for XOR_n with Incomplete Training Set

The analysis for XOR_n shows that if the training set size is at most $\log(n)$, the training set can be fit efficiently (Theorem 7). On the other hand if the size of the training set grows asymptotically faster than $\log n$, then the algorithms do not fit the training set in polynomial time with probability exponentially close to 1 (Theorem 8). However, in both cases the generalization ability will be equal to 1/2 with probability exponentially close to 1 in the former case, and with probability 1 in the latter case (Theorem 9).

Theorem 7. *Let $s \le \ln n$ be the size of the training set chosen from the truth table uniformly at random with repetition. Then the (1+1) GP using $F = \{XOR\}$ as function set and $L = \{X_1, \dots, X_n\}$ as terminal set will find a solution fitting the training set of XOR_n in time $O(n^2)$ with probability at least $1 - e^{-\Omega(n)}$.*

Proof. Since each row of the training set is binomially distributed with parameters n and $p = 1/2$, also each variable X_i is binomially distributed, but with parameters s and $p = 1/2$ because the training set has s rows. An element j (i.e., $1 \leq j \leq s$) of the optimal solution *opt* fitting the training set has value 1 if the number of 1 s in the n variables $X_1, \ldots, X_n$ at position j is odd while it has value 0 if the number of 1 s at position j in the n variables is even. Hence, once the training set has been created, the optimal solution is determined and each variable X_i will have the same value as *opt* at each position j with probability 1/2. By the principle of deferred decisions [8], the same holds for $X_i \oplus, \ldots, \oplus X_m$, $1 \leq i, m \leq n-1$ at each position j. In fact if the first $m-1$ terms $X_i, \oplus, \ldots, \oplus X_{m-1}$ have the "correct" value at position j, then with probability 1/2 the output will still be correct after the first $m-1$ terms are XOR-ed with X_m (i.e., X_m has a 0 at position j). On the other hand if it was not correct it would become "correct" with probability 1/2 (i.e., X_m has a 1 at position j). Thus

$$\begin{aligned} P(\text{j is correct}) &= P(X_m(j) = 0 \mid \text{j was correct}) + P(X_m(j) = 1 \mid \text{j was not correct}) \\ &= 1/2 \cdot 1/2 + 1/2 \cdot 1/2 = 1/2 \end{aligned}$$

As a result the probability that any solution, $X_i \oplus, \ldots, \oplus X_m$, $1 \leq i, m \leq n-1$ is equal to *opt* in all positions is bounded by $p(X_t = opt) = \frac{1}{2^s} \geq \frac{1}{2^{\ln n}} = \frac{1}{n}$. Hence the probability that a solution $X_i \oplus, \ldots, \oplus X_m$, $1 \leq i, m \leq n-1$ is different from *opt* is less than $(1 - 1/n)$ and the probability that cn^2 solutions are all different than *opt* is bounded by

$$\left(1 - \frac{1}{n}\right)^{cn^2} \leq \left(1 - \frac{1}{n}\right)^{n \cdot cn} \leq \left(\frac{1}{e}\right)^{cn}$$

where $c < 1$ is a positive constant. As a result, after visiting cn^2 distinct solutions, with probability at least $1 - e^{-cn}$, the training set has been fitted by the algorithm.

All that remains to be shown is that cn^2 distinct solutions are visited by the (1+1)-GP. We consider a current solution of the algorithm $X_1, X_2, \ldots, X_m$, $\overline{X_{m+1}}, \ldots, \overline{X_n}$ where the $\overline{X_i}$ are the variables that are missing after simplification. By just considering an INS operation, from each such solution it is possible to reach n different neighbours (i.e., $n - i$ neighbours are reached by adding a missing variable and i neighbours are reached by adding one of the m variables, hence effectively removing a variable after simplification). If less than $(9/10)n$ of the neighbours on the current level have not been visited, the probability of visiting one is at least $1/3 \cdot 9/10 = 3/10 > 1/4$. Otherwise we consider the set of neighbours as "full" and look at the probability of moving to a new solution not having any neighbours in common with the current solution. Such a solution can be visited by either: (a) adding two missing variables; (b) removing two of the m variables in the current solution; (c) adding one missing variable and removing one of the m variables. Each of (a), (b), (c) can be achieved by performing two consecutive INS operations of which the first operation must lead to an accepted search point (i.e., which happens with probability at least 1/2). If, after the first INS operations all the neighbours of $c'n$ different solutions (not having neighbours in common with the previous search point) were all "full" then $c'(9/10)n^2$ distinct solutions would have been visited (recall a neighbourhood is full when $(9/10)n$ neighbours have been visited). We set $c'(9/10) > c$. Hence, unless $(9/10)c'n^2$ distinct solutions have been seen, there must be at least a constant fraction $c''n$ of neighbours reached by the second INS that lead to unseen new solutions by another INS operation. Overall, the probability of reaching a new solution with 'unfilled neighbourhood' is at least $1/3 \cdot 1/3 \cdot 1/2 \cdot (c''n)/n = c^*$. Since each of the distinct solutions is found with constant probability, the expected time to find them all is bounded from above by $(c'n^2 \cdot c^*)$. By another application of Chernoff bounds with success probability c^* and a phase of length $2c'c^*n^2$, we get that $c'n^2$ distinct solutions have not been seen with probability at most $e^{-1/8 \cdot c * c'n^2}$. Summing up the failure probabilities proves the statement of the theorem. □

We now consider larger training sets.

Theorem 8. *Let $s = \omega(\log n)$ be the size of the training set chosen from the truth table uniformly at random with repetition. Then the (1+1) GP using $F = \{XOR\}$ and $L = \{X_1, \ldots, X_n\}$ will not find a solution fitting the training set of XOR_n with fitness better than $(1-\delta)s/2$, $\delta > 0$ any constant, in polynomial time with probability at least $1 - e^{-\Omega(s)}$.*

Proof. As shown in the proof of Theorem 7 each column of the training set is binomially distributed with parameters s and $p = 1/2$, and by the principle of deferred decisions the same holds for each candidate solution $X_i \oplus, \ldots, \oplus X_m$, $1 \leq i, m \leq n-1$. As a result the probability that each row of a candidate solution is equal to the value in the respective row in *opt* with probability $p(Y_i) = 1/2$ and the expected number of rows where they are equal is $E(Y) = s/2$. Hence, by a simple application of Chernoff bounds, the probability that a candidate solution has $(1+\delta)s/2$ rows that agree with the respective rows in *opt* is

$$P\left(Y > \frac{s}{2} + \delta\frac{s}{2}\right) \leq e^{-(\delta/6) \cdot s)}$$

By the union bound the probability that after a polynomial number of steps n^c any seen solution agrees in $(1+\delta)s/2$ rows is less than

$$n^c \cdot e^{-(\delta/6) \cdot s)} \leq e^{c \log n - (\delta/6)s} \leq e^{-\Omega(s)}$$

for any $s = \omega(\log n)$, which proves the statement of the theorem. □

Theorem 9. *Both the (1+1) GP and the (1+1) GP* using $F = \{XOR\}$, $L = \{X_1, \ldots, X_n\}$ and a training set of size $s = poly(n)$ have a generalization ability of $G = 1/2$.*

Proof. By Proposition 2 all the expressions but $X_1 \oplus \ldots \oplus X_n$ have a generalization error of 2^{n-1} thus if the algorithms do not find the expression $Y = X_1 \oplus \ldots \oplus X_n$ in polynomial time their generalization ability is $G = 1 - \frac{2^{n-1}}{2^n} = 1/2$. By Theorem 3 the (1+1) GP* will never move from its first point, hence does not find Y with probability $1 - 2^{-s}$. The (1+1) GP with $s = \omega(\log n)$ cannot fit the training set in polynomial time by Theorem 7, thus cannot find Y efficiently. When $s < \log(n)$ we use a similar argument to that of Theorem 4 to show that, even if the training set can be fit in polynomial time, the expression Y will not be found in polynomial time. In fact since with probability 1/2 a new solution will have fitness better than its parent, the expected drift will be half of that calculated in Theorem 4 (i.e., in expectation half of the offspring will not be accepted). Thus the time to get to Y is still at least exponential with probability exponentially close to 1, which concludes the proof. □

5 Conclusions

A further step has been made towards the rigorous computational complexity analysis of GP for evolving functions with desired input/output behaviour. We have analysed the (1+1) GP and (1+1)-GP* for evolving two common boolean functions, XOR_n and AND_n (which may be considered the GP analogues to OneMax and Needle for EAs) both in the case of complete input-output information and with incomplete training sets of polynomial size.

We have rigorously proved that AND_n can be efficiently solved by both the (1+1) GP and the (1+1) GP* in case of complete datasets, with (1+1) GP* producing shorter expressions. We have also shown that both the algorithms can efficiently fit a training set of polynomial size and provided a necessary condition to achieve a given generalization ability. The analysis for XOR_n reveals a needle-like fitness landscape, leading to the (1+1) GP* not being able to solve the problem at all and the (1+1) GP requiring exponential time. The analysis for the incomplete datasets has shown that there is a $\log(n)$ threshold for the size of the training set under which the training set can be fit efficiently and over which asymptotically it cannot be fit in polynomial time with probability exponentially close to 1. The analysis on the generalization ability has shown that, despite the size of the training set, the generalization ability is equal to 1/2 with probability exponentially close to 1.

Future work will be directed, on one hand, towards extending the results to increasingly deal with more comprehensive terminal and function sets, while on the other hand will focus on more sophisticated GP systems and benchmark functions where typical bloat and overfitting problems can be studied.

Acknowledgements. The authors would like to thank Alberto Moraglio for constructive discussions which initialised this work. Further preliminary discussions occurred at Dagstuhl Seminar N.15211. This work was supported by EPSRC under Grant n. EP/M004252/1.

References

1. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics: Foundations and Recent Developments. World Scientific, Singapore (2011)
2. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), 51–81 (2002)
3. Kötzing, T., Sutton, A.M., Neumann, F., O'Reilly, U.M.: The max problem revisited: the importance of mutation in genetic programming. Theor. Comp. Sci. **545**, 94–107 (2014)
4. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
5. Koza, J.R.: Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, Cambridge (1994)
6. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer, Heidelberg (2002)
7. Luke, S.: Genetic programming produced competitive soccer softbot teams for RoboCup97. In: Proceedings of the Third Annual Conference on Genetic Programming 1998, pp. 214–222. Morgan Kaufmann (1998)
8. Mitzenmacher, M., Upfal, E.: Probability and Computing. Cambridge University Press, Cambridge (2005)
9. Moraglio, A., Mambrini, A., Manzoni, L.: Runtime analysis of mutation-based geometric semantic genetic programming on boolean functions. In: Proceedings of FOGA XII, pp. 119–132. ACM (2013)
10. Neumann, F., O'Reilly, U.M., Wagner, M.: Computational complexity analysis of genetic programming - initial results and future directions. In: Riolo, R., Vladislavleva, E., Moore, J.H. (eds.) Genetic Programming Theory and Practice IX. Genetic and Evolutionary Computation, pp. 113–128. Springer, New York (2011)
11. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity. Natural Computing Series. Springer, Heidelberg (2010)
12. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica **59**(3), 369–386 (2011)
13. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Genet. Program. Evolvable Mach. **11**(3–4), 339–363 (2010)
14. O'Reilly, U.-M., Oppacher, F.: Program search with a hierarchical variable length representation: genetic programming, simulated annealing and hill climbing. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 397–406. Springer, Heidelberg (1994)
15. Poli, R., Langdon, W.B., McPhee, N.F.: A field guide to genetic programming (2008). http://lulu.com
16. Poli, R., McPhee, N.F., Rowe, J.E.: Exact schema theory and Markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. Genet. Program. Evolvable Mach. **5**(1), 31–70 (2004)
17. Spector, L., Barnum, H., Bernstein, H.J., Swamy, N.: Quantum computing applications of genetic programming. In: Spector, L., Langdon, W.B., O'Reilly, U.M., Angeline, P.J. (eds.) Advances in Genetic Programming 3, pp. 135–160. MIT Press, Cambridge (1999)
18. Valiant, L.G.: Evolvability. J. ACM **56**(1), 3:1–3:21 (2009)