The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

On the Time and Space Complexity of Genetic Programming for Evolving Boolean Conjunctions

Andrei Lissovoi, Pietro S. Oliveto
Rigorous Research, Department of Computer Science
University of Sheffield, Sheffield S1 4DP, United Kingdom
{a.lissovoi,p.oliveto}@sheffield.ac.uk

Abstract

Genetic Programming (GP) is a general purpose bio-inspired meta-heuristic for the evolution of computer programs. In contrast to the several successful applications, there is little understanding of the working principles behind GP. In this paper we present a performance analysis that sheds light on the behaviour of simple GP systems for evolving conjunctions of n variables (AND_n). The analysis of a random local search GP system with minimal terminal and function sets reveals the relationship between the number of iterations and the expected error of the evolved program on the complete training set. Afterwards we consider a more realistic GP system equipped with a global mutation operator and prove that it can efficiently solve AND_n by producing programs of linear size that fit a training set to optimality and with high probability generalise well. Additionally, we consider more general problems which extend the terminal set with undesired variables or negated variables. In the presence of undesired variables, we prove that, if non-strict selection is used, then the algorithm fits the complete training set efficiently while the strict selection algorithm may fail with high probability unless the substitution operator is switched off. In the presence of negations, we show that while the algorithms fail to fit the complete training set, the constructed solutions generalise well. Finally, from a problem hardness perspective, we reveal the existence of small training sets that allow the evolution of the exact conjunctions even in the presence of negations or of undesired variables.

1 Introduction

Genetic Programming refers to a class of evolutionary algorithms introduced by (Koza 1992) to evolve computer programs. Traditionally syntax trees have been used to represent programs. Their quality is evaluated by executing the trees on a set of inputs and their output is compared with that of a target function (the function to be evolved). The set of input/output test cases is usually referred to as the *training set*. Typical Genetic Algorithm (GA) variation and selection operators, adapted to work on syntax trees, are used to create new programs and natural selection principles are used to evolve a population of programs with the aim of eventually identifying one with the desired functionality. Although during the evolution the quality of programs is measured using the training set, the goal is to evolve a program that works well on all possible inputs. In this case the program is said to have good generalisation. GP can be applied to problems for which no efficient problem-specific algorithms are available, aiming to produce reasonable, if not optimal, solutions without requiring problem-specific algorithm design.

While there are many examples of successful applications of GP to practical problems (see e.g. (Koza 2010; Schuh, Angryk, and Sheppard 2012; Liu and Shao 2013; Al-Sahaf et al. 2017) for recent ones), there is a limited understanding of what problem characteristics make GP successful, how the multiple parameters should be set for optimal results, and how common failures could be avoided (O'Neill et al. 2010; Poli, Langdon, and McPhee 2008). A solid theoretical foundation is necessary to answer these questions and could be used to guide the design of future GP algorithms, as well as the choice of parameters in their applications.

A performance analysis of GP should focus on two different aspects of algorithmic behaviour: 1) the ability of the algorithm to fit a complete training set, and 2) whether the solutions evolved using a smaller training set generalise well. If the complete training set is not too large, then fitting the training set is sufficient for efficient optimisation. If, instead, its size is prohibitively large, then the generalisation capabilities are crucial. In any case, an analysis of the performance of a GP system using the complete set serves as a best-case model of its behaviour: if the system is unable to produce a reasonable solution while working with a complete training set, it is unlikely to perform well in practice.

Most previous theoretical work on the analysis of GP aimed at understanding the performance of simple GP systems for evolving trees with significant structures rather than programs with a given functionality (Langdon and Poli 2002; Durrett, Neumann, and O'Reilly 2011; Kötzing et al. 2014; Wagner, Neumann, and Urli 2015; Doerr et al. 2017). Only recently (Mambrini and Oliveto 2016) analysed the same simple GP algorithms for the AND_n and XOR_n problems that represent actual logical functions with proper inputs and logically-defined outputs. The goal of the problems is to evolve respectively conjunction and parity functions of n variables using a function set consisting solely of a binary AND (XOR, respectively) operator, and n input variables as terminals. These functions were chosen to highlight the performance and behaviour of the GP system respectively for

evolving an easy and a hard function since AND_n is known to be evolvable in the PAC learning framework while XOR_n is not (Valiant 2009). Note that there are learnable problems for which no GP system will be efficient because, compared to the more general concept of learnability, additional restrictions are imposed on the actions the GP is allowed to perform. In Valiant's Evolvability Theory, which is a restricted case of PAC-learning, the aim is to evaluate what functionalities can be acquired by an evolutionary process that learns from examples. A function is *evolvable* if there exists an evolution algorithm that evolves it. The aim of GP is to evolve a function automatically, without the need of a human-designed evolution algorithm. The question is what classes of functions may be evolved efficiently via GP, and which actually require a human-designed evolution algorithm (given that they are evolvable). (Mambrini and Oliveto 2016) prove that a simple GP system using a random local search operator which may either add, delete or substitute a node in the current solution, can efficiently evolve the conjunctions of n variables while it requires exponential time with overwhelming probability for the parity functions.

Compared to previous runtime analyses, AND_n and XOR_n are indeed proper functions with input/output behaviour. However, the settings considered in the paper are still far from realistic applications of GP. One drawback of the work is that the random local search operator used by the GP system is considerably different from the typical mutation operators used in evolutionary computation and GP, which allow to make larger changes to the current solution with some positive probability. More importantly, the function and terminal sets were limited to contain exactly the "right" ingredients, instead of analysing whether the GP system could learn to select the right functions and terminals from larger sets during the evolution.

In this paper, we analyse the performance and behaviour of GP for evolving Boolean conjunctions in these more realistic settings. The presented analysis reveals how and why small changes in the problem setting can hugely affect the performance of GP. We start by presenting a fixed budget computation analysis that provides a relationship between the number of iterations that the GP system is allowed to run and the expected error of the evolved program for the same random local search GP system (which we call RLS-GP) and settings considered by (Mambrini and Oliveto 2016).

Afterwards, we generalise the results to more realistic GP systems using more sophisticated mutation operations with larger neighborhoods. Apart from deriving time complexity results we also perform a space complexity analysis that delivers precise asymptotic bounds on the size of the evolved programs. The size of the produced tree allows us to derive precise statements on the generalisation ability of the produced solutions. In particular, for realistic training sets of polynomial size, the GP systems evolve programs of logarithmic size in the number of variables. These solutions, nevertheless, generalise well with high probability.

We then consider more challenging versions of the conjunction problem that allow to highlight how slight differences in the problem structure may lead to great differences in GP behaviour, hence of GP performance. On one hand, we extend the function set to not only contain the required AND operators but also an unnecessary negation operator (i.e. NOT)[1]. This setting was already considered in (Mambrini and Oliveto 2016) to show that by adding an unnecessary operator the RLS-GP becomes inefficient with overwhelming probability on AND_n. Our analysis shows that allowing mutation to insert, delete or substitute multiple variables at once does not help the GP. Nevertheless, the produced solutions generalise well. On the other hand, we extend the terminal set such that it also contains unnecessary variables (i.e., the target conjunction is a subset of the terminal set). This generalised version of the conjunction problem, which we call $\text{AND}_{n,m}$, is an interesting benchmark problem as it has been proven to be efficiently evolvable according to the PAC learning framework notion (Valiant 2009). Our results show that the non-strict selection RLS-GP fits the training set efficiently while the strict selection algorithm RLS-GP* may fail with high probability unless the substitution operator is switched off.

We conclude by presenting a problem hardness analysis that shows that for all the settings considered in the paper there exist training sets of linear size which allow the GP systems to efficiently produce solutions which generalise perfectly (i.e. are equivalent to the target function).

The rest of the paper is structured as follows. In the next section, we will introduce the RLS-GP and $(1+1)$ GP algorithms and precisely define the learning problems and measures of generalisation ability. In Section 3, we present the fixed budget analysis of the RLS-GP for the AND_n problem. In Section 4 we present the analysis of $(1+1)$ GP for the AND_n problem proving that the algorithms evolve the conjunctions efficiently and that they generalise well. In Section 5 we present the analysis of the GP systems for the more difficult problem where also negations of variables are allowed. In Section 6 we extend the terminal set to also contain unnecessary terminals. In Section 7 we prove the existence of a linear training set that allows the GP algorithms to evolve exact solutions to the AND_n and $\text{AND}_{n,m}$ problems, also if the negated variables are included in the terminal set. Finally, we conclude the paper by providing a summary of the results and discussing directions for future work.

Due to space constraints, the proofs of the theorems are omitted from this extended abstract. The proofs use standard randomised algorithm analysis techniques (Feller 1968; Mitzenmacher and Upfal 2005; Doerr 2011), including Additive (He and Yao 2001), Multiplicative (Doerr and Goldberg 2010; Doerr, Johannsen, and Winzen 2012), and Negative (Oliveto and Witt 2011; 2012; Rowe and Sudholt 2014) Drift Analysis.

2 Preliminaries

The following sections present results for a total of four simple GP algorithms in various settings. All four algorithms

[1]More precisely, to simplify technical issues due to the NOT function being unary, we introduce negated literals into the terminal set. While the behaviour of this GP system is not equivalent to the behaviour of a system using a NOT function, both systems share the same difficulties when optimising the AND_n function.

1: Choose $op \in \{\text{INS}, \text{DEL}, \text{SUB}\}$ uniformly at random
2: **if** X is an empty tree **then**
3: Choose a literal $l \in L$ uniformly at random
4: Set l to be the root of X.
5: **else if** $op = \text{INS}$ **then**
6: Choose a node $x \in X$ uniformly at random
7: Choose $f \in F, l \in L$ uniformly at random
8: Replace x with f
9: Set the children of f to be x and l, order chosen u.a.r.
10: **else if** $op = \text{DEL}$ **then**
11: Choose a leaf node $x \in X$ uniformly at random
12: Replace x's parent in X with x's sibling in X
13: **else if** $op = \text{SUB}$ **then**
14: Choose a leaf node $x \in X$ uniformly at random
15: Choose a literal $l \in L$ uniformly at random
16: Replace x with l.

Figure 1: The HVL-Prime mutation operator on a tree X.

evolve programs using a syntax tree representation, and use the HVL-Prime mutation operator shown in Figure 1 (first proposed in (O'Reilly and Oppacher 1996)) to construct offspring solutions in each iteration (Durrett, Neumann, and O'Reilly 2011; Kötzing et al. 2014; Mambrini and Oliveto 2016). Figure 2 shows the general pseudocode for all four algorithm variants.

The RLS-GP and RLS-GP* algorithms maintain a single solution, and apply a single instance of the HVL-Prime mutation operator to produce an offspring solution in each iteration. The performance of these algorithms on AND_n and XOR_n problems has been formally analysed in (Mambrini and Oliveto 2016). We note that previous work has referred to these algorithms as the $(1+1)$ GP; throughout this paper, we use the RLS-GP name to emphasize the local search nature of the mutation operator, and use $(1+1)$ GP to refer to an algorithm using a global mutation operator, which is also known as the (1+1) GP-multi. Our notation matches the conventions of runtime analysis of evolutionary algorithms (Oliveto and Yao 2011; Jansen 2013).

We also consider the $(1+1)$ GP and $(1+1)$ GP* algorithms, which perform $k = 1 + \text{Poisson}(1)$ iterative HVL-Prime mutations to produce an offspring solution in each iteration. The extended mutation operator has been previously considered on other problems in (Kötzing et al. 2014; Durrett, Neumann, and O'Reilly 2011), leading to the so-called $(1+1)$ GP-multi algorithms (in this paper, we will omit the "-multi" suffix).

Recall that the probability density function of a Poisson(1)-distributed variable is:

$$P(X = x) = \lambda^x e^{-\lambda}/(x!) = 1/(e\,x!),$$

while the $(1+1)$ GP and $(1+1)$ GP* algorithms perform $x+1$ HVL-Prime sub-operations. In each iteration, they perform a single HVL-Prime sub-operation with probability $1/e$, two sub-operations with probability $1/e$, three with probability $1/(2e)$, and so on. In expectation, two HVL-Prime sub-operations are performed in each iteration.

In the following sections we will consider problem in-

1: Initialise an empty tree X
2: **for** $t \leftarrow 1, 2, \ldots$ **do**
3: $X' \leftarrow X$
4: $k \leftarrow 1 + \text{Poisson}(1)$ ▷ $k \leftarrow 1$ in RLS-GP
5: **for** $i \leftarrow 1, \ldots, k$ **do**
6: $X' \leftarrow \text{HVL-Prime}(X')$
7: **if** $f(X') \leq f(X)$ **then** ▷ Strict $<$ in * variants
8: $X \leftarrow X'$

Figure 2: The $(1+1)$ GP and RLS-GP algorithms (in RLS-GP and RLS-GP*, k is always set to 1). In $(1+1)$ GP* and RLS-GP*, line 7 instead uses a strict comparison.

stances based on the AND function: the objective of the GP systems will be to evolve a tree which computes the conjunction $x_1 \wedge \ldots \wedge x_n$ (i.e., AND_n), or a conjunction of a subset of $m < n$ available variables (i.e., $\text{AND}_{n,m}$). For Boolean functions of n variables, the complete training set is a truth table with 2^n rows. Solution quality is measured by the number of rows on which the output of the current solution and the target function is different.

Definition 1. *The error ϵ of a solution h compared to $\hat{h}$ is defined as the probability that h and $\hat{h}$ differ on a truth table row r selected uniformly at random:*

$$\text{error}(h) = P(h(r) \neq \hat{h}(r)) = \epsilon.$$

For AND_n and $\text{AND}_{n,m}$, it is possible to quickly calculate $\text{error}(h)$ without evaluating the full 2^n rows of the truth table. This might not be possible when using GP for more practical problems. To model this difficulty, we also present analyses for GP algorithms using incomplete training sets, where the fitness function does not provide an accurate value of $\text{error}(h)$, but is instead based on a small sample of truth table rows. In such settings, we consider the generalisation ability of GP systems in terms of the expected error of the solution over the complete truth table, i.e. based on samples taken uniformly at random from the complete truth table, without using any problem-specific information.

We consider two ways of randomly selecting the training set of size s from the complete truth table of the target function: 1) it could be chosen at the beginning of the optimisation process and used for all fitness evaluations, or 2) a new sample could be taken at every iteration of the GP algorithm, and used for fitness evaluations within that iteration only. This reflects the scenarios where only a limited amount of information about the target function is available (such as in medical applications, e.g., (Archetti et al. 2007)), and where a prohibitively large amount of information is available to be used in every fitness evaluation (such as problems involving large data sets, e.g., (Song, Heywood, and Zincir-Heywood 2005)).

Note that the class of AND functions (in particular, $\text{AND}_{n,m}$) is evolvable under uniform distribution (Valiant 2009). However, it is not distribution-free evolvable using a Boolean loss function (Feldman 2008). This is of little concern to our aims, since all rows of the truth table are of equal importance for evaluating the correctness of the

evolved function. Furthermore, the GP is allowed to choose which rows to sample in order to evaluate the fitness of a solution. Our problem, instead, is that it is not possible to efficiently sample all the 2^n truth table rows to evaluate the exact quality of a candidate solution.

For our analysis, we will rely on the following observation concerning the fitness values of solutions of AND_n: as the number of distinct variables used by a candidate solution increases, its error relative to the target function decreases.

Proposition 2. *Every conjunction of v distinct variables differs from the target function $\text{AND}_n(x_1, \ldots, x_n)$ on $f_v = 2^{n-v} - 1$ rows.*

3 Fixed budget analysis of RLS-GP on AND_n

In this section we consider AND_n, the problem of evolving a conjunction of n variables, using minimal function and terminal sets. In this setting, the GP algorithms need to construct a tree which includes each variable at least once. This problem has been considered for the RLS-GP and RLS-GP* algorithms in (Mambrini and Oliveto 2016), where it has been shown that the algorithms find an exact solution in expected $O(n \log n)$ iterations. For a deeper understanding of the performance of RLS-GP, we present a fixed budget analysis, providing a relationship between the expected solution quality and the time the algorithms are allowed to run. The proofs follow the techniques used in (Jansen and Zarges 2014) to analyse Randomised Local Search on the ONEMAX problem.

Theorem 3. *For all budgets b, the expected fitness of the solution produced by the RLS-GP* on AND_n when initialised with an empty tree is: $E(f(x_b)) = 2^{n(1-1/(3n))^b} - 1$.*

The RLS-GP will accept mutations which insert copies of variables which are already present in the tree (as these mutations do not affect the solution's fitness value), which also allows it to accept mutations which apply the substitution sub-operator of HVL-Prime to replace a redundant variable in the tree with a potentially new variable. In this setting, we can provide upper and lower bounds on the performance of the RLS-GP, as illustrated in Figure 3: the result of Theorem 3 provides a lower bound on the quality of the solution at time t, and Theorem 4 provides an upper bound based on the substitution operator always selecting a redundant variable for substitution.

Theorem 4. *For all budgets b, the expected fitness of the solution produced by the RLS-GP on AND_n when initialised with an empty tree is:*

$$2^{n(1-1/(3n))^b} - 1 \leq E(f(x_b)) \leq 2^{n(1-2/(3n))^b} - 1.$$

4 (1+1) GP on AND_n

In this section we present a runtime analysis of the Poisson-mutation used by the $(1+1)$ GP and $(1+1)$ GP* algorithms on the AND_n problem, first using the complete truth table as the training set, and then using training sets of polynomial size, chosen uniformly at random at either the beginning of the optimisation process or independently in each iteration.

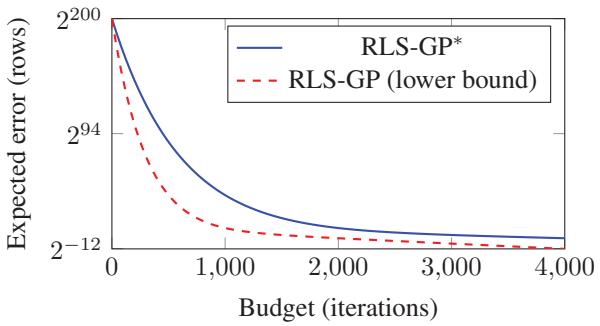

Figure 3: Fixed budget illustration for $n = 200$ variables. The expected error of the RLS-GP* also serves as an upper bound on that of the RLS-GP.

4.1 Complete training set

With minimal literal and function sets, both the $(1+1)$ GP and the $(1+1)$ GP* algorithms are able to fit the complete training set efficiently.

Theorem 5. *The $(1+1)$ GP and the $(1+1)$ GP* using $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ solve AND_n using the complete truth table as training set in time $\Theta(n \log n)$.*

We additionally prove that the solutions produced only contain $\Theta(n)$ leaf nodes in expectation.

Theorem 6. *The $(1+1)$ GP and the $(1+1)$ GP* using $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ solve AND_n using the complete truth table as training set produce a solution with an expected $\Theta(n)$ terminals.*

4.2 Static polynomial-size training sets

Since the efficient evaluation of all 2^n truth table rows for the target function is not possible without problem-specific insight, in practice GP algorithms are run on a smaller training set of rows selected from the complete truth table. The training set remains fixed throughout the process, which reflects a situation where only a limited amount of information about the target function's input/output behaviour is available. In this subsection, we show that the $(1+1)$ GP* and $(1+1)$ GP algorithms easily fit polynomially-sized training sets in polynomial time and provide solutions with good generalisation ability.

To begin with, we show that in $O(\log n)$ iterations, both of the $(1+1)$ GP algorithms are able to find a solution that fits a polynomially-sized training set.

Theorem 7. *The $(1+1)$ GP* and $(1+1)$ GP algorithms, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$, will fit the training set of $s = \text{poly}(n)$ rows drawn uniformly with replacement from the complete truth table of AND_n in expected $O(\log n)$ iterations. Additionally, for any $c > 0$, a solution fitting the training set is constructed within $O((c+1)\log n)$ iterations with probability at least $1 - n^{-c}$.*

Generalisation ability In (Mambrini and Oliveto 2016), it was proved that a solution with $O(\log n)$ variables will fit a polynomially-sized training set with high probability; this will be reflected in Observation 9. However, our Theorem 7

only provides an upper bound on the number of iterations it takes to construct a solution fitting the training set, which does not guarantee that this solution will have $\Omega(\log n)$ variables as in expectation the algorithms will attempt as many deletions as insertions. The following theorem shows that the constructed solution will indeed contain at least $\Omega(\log n)$ distinct variables.

Theorem 8. *The solution produced by the* $(1+1)$ GP* *and* $(1+1)$ GP *algorithms, with* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n\}$*, using a training set of size* $s = \text{poly}(n)$ *and* $s \geq n^{c'}$*, where* $c' > 0$ *is an arbitrary constant, drawn uniformly with replacement from the complete truth table of AND*$_n$*, in expectation contains* $\Omega(\log n)$ *distinct literals.*

Having shown that the expected number of distinct variables in the solution that fits the training set is $\Omega(\log n)$, we can now state the generalisation ability of the GP algorithms.

Observation 9. *A conjunction of* $c \log n$ *distinct variables, where* $c > 0$ *is a constant, matches the AND*$_n$ *function on a row drawn uniformly at random from the complete truth table of AND*$_n$ *with probability* $1 - n^{-c}$.

In Section 7, we will show that if the training set is chosen arbitrarily, using just n specific rows of the complete truth table is sufficient for any of the considered GP algorithms using minimal function and terminal sets to evolve a solution with a generalisation error of 0.

4.3 Dynamic polynomial-size training sets

We now consider the behaviour of the GP algorithms when a smaller training set is chosen independently at random from the complete truth table of the target function in each iteration. This approach would be used when the complete truth table describing the behaviour of the target function is known, but is prohibitively large to evaluate in its entirety for every fitness comparison performed by the GP algorithms.

Theorem 10. *Let* $n^{2c+\epsilon}$ *rows be sampled from the complete truth table in each iteration (where* $c > 0$ *and* $\epsilon > 0$ *are any constants) for AND*$_n$ *with* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n\}$*. The RLS-GP and the* $(1+1)$ GP *will terminate in expected* $O(\log^2 n)$ *iterations. W.o.p. the generalisation error will be at most* n^{-c}*. The RLS-GP** *and the* $(1+1)$ GP* *will find a solution with a generalisation error of at most* n^{-c} *within an expected* $O(\log n)$ *iterations.*

5 Extended function set: adding negations

In this section we consider the effect of allowing the GP algorithms to access additional functions when constructing solutions for AND$_n$. More specifically, we introduce the negation (unary NOT) operator. To avoid having to modify the HVL-Prime mutation operator to support unary functions, we introduce negation by extending the literal set rather than the function set, i.e. $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$. While this is not exactly equivalent in expressive power to having NOT in the function set, the issues encountered by the GP algorithms in the simplified setting are similarly possible when NOT is added to the function set. We show that similarly to the results for the RLS-GP algorithms with local-search mutation in (Mambrini and Oliveto 2016), the $(1+1)$ GP algorithms are also unable to find the optimal solution using the complete truth table in polynomial time.

Theorem 11. *The* $(1+1)$ GP* *using* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ *does not fit the complete training set for AND*$_n$ *in polynomial time with overwhelming probability.*

The analysis for the $(1+1)$ GP is more complex, as the algorithm allows the current solution to mutate almost freely after a contradiction has been obtained, since all solutions containing a contradiction are wrong on exactly one row (where all variables are set to true).

Theorem 12. *The* $(1+1)$ GP *using* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ *does not fit the complete training set for AND*$_n$ *in polynomial time with overwhelming probability.*

We note that a solution containing a contradiction matches the output of AND$_n$ on all but one row of the complete truth table, and thus has a generalisation error of just 2^{-n}. Hence, while introducing negations prevents the GP algorithms from constructing the optimal solution in polynomial time, it does not harm the generalisation ability of the current-best solution, including the cases where an incomplete training set is used: if the optimisation process ends by introducing a contradiction, then the resulting solution will achieve a much lower error than the solutions produced in section 4.2 using minimal terminal and function sets.

Contradictions are also not the only problem the GP systems face when negations can be added to the solution. Even if the solutions including a contradiction could never be accepted, non-optimal solutions which contain all n distinct variables in either positive or negative form have the same fitness value (i.e., they are wrong on two rows: the all-true row, and the row including setting all positive literals to true and all negative literals in the solution to false). Since inserting both negative and positive literals improves the fitness of the mutated offspring, it is unlikely that the first solution with all n distinct variables produced by the GP has significantly more positive literals than negative ones. As replacing negative literals with positive literals gets increasingly unlikely the fewer negative literals remain in the solution, it would be overwhelmingly unlikely that a solution with no negative literals is encountered in polynomial time.

In Section 7, we will show that there exists a training set of $2n + 1$ rows (or, if a population with a diversity mechanism is used, just $n + 1$ rows), that is sufficient to allow the GP algorithms to evolve a solution with a generalisation error of 0 even when negations are present in the terminal set.

6 Extended terminal set: AND$_{n,m}$

In general, when evolving a program it is not necessarily known in advance which distinct terminals will be required. (Valiant 2009) considered a setting where target functions are a conjunction of an unknown subset of n variables, modeling an uncertainty over which inputs are actually used in e.g. a classification problem. In this section we tackle a similar setting by considering the $AND_{n,m}$ problem, where the

target function is a conjunction of $m < n$ variables in the terminal set – and thus the GP algorithm has to contend with variables which are ultimately ignored by the target function. We point out that, differently from Valiant's work, the GP systems we consider are not especially designed to solve the problem. Throughout this section we consider local search mutation operators to simplify the analysis.

6.1 Complete training set

Similarly to Proposition 2, the error of a candidate solution on the $\text{AND}_{n,m}$ problem can also be calculated without explicitly evaluating all 2^n rows of the truth table.

Proposition 13. *Let $\hat{h}$ be a conjunction of m distinct variables, $m \leq n$. Any conjunction containing $a \leq m$ distinct variables in $\hat{h}$, and $b \leq n - m$ distinct variables not in $\hat{h}$ will differ from $\hat{h}$ on $f_{a,b} = 2^{n-a-b} + 2^{n-a} - 2^{n+1-m-b}$ rows of the n-variable truth table for $\hat{h}$.*

The following observation specifies exactly when adding and removing variables ignored by the target function improves the fitness value of a candidate solution.

Observation 14. *Suppose $\hat{h}$ is a conjunction of $m \leq n$ distinct variables. For a conjunction with $a \leq m$ distinct variables in $\hat{h}$, and $b \leq n-m$ distinct variables not in $\hat{h}$, adding a new distinct variable in $\hat{h}$ to the conjunction always decreases $f_{a,b}$, while adding a new distinct variable not in $\hat{h}$ to the conjunction decreases $f_{a,b}$ if and only if $a < m - 1$, and increases $f_{a,b}$ if and only if $a = m$.*

We will show that the HVL-Prime SUB operation prevents the RLS-GP* algorithm from fitting the complete training set, while the RLS-GP is able to do so in a polynomial number of iterations. To illustrate the former behaviour, we set m to be linear with respect to n in the following theorem.

Theorem 15. *The RLS-GP* algorithm with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ will with high probability fail to find the optimum of $\text{AND}_{n,m}$ in finite time when $m = cn$ for any constant $0 < c < 1$ when using the complete training set.*

We note that the quality of the solution produced by the RLS-GP* when it gets stuck with multiple copies of an undesired variable is not prohibitively bad: such a solution would still contain all m desired variables, and, in expectation, at most $(n - m)/2$ non-$\hat{h}$ variables (as in expectation at least half of the non-$\hat{h}$ variables would be removed rather than substituted out). Recalling Proposition 13, and setting $a = cn, b = (n - cn)/2$, we get an error on $f_{a,b} = 2^{n(1-c)} - 2^{n(1-c)/2}$ rows; and hence $\text{error}(h^*) < 2^{-cn}$.

Without the SUB sub-operation of HVL-Prime, the RLS-GP* is able to fit the complete training set in polynomial time, since mutations which insert additional copies of any variable into the tree would never improve the fitness of the current solution, and hence would never be accepted.

Theorem 16. *The RLS-GP* algorithm, using the HVL-Prime mutation operator without the SUB operation, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ will find the optimum of $\text{AND}_{n,m}$ in expected $O(n \log n)$ iterations when using the complete training set.*

We now show that the non-strictly elitist RLS-GP algorithm is able to fit the complete training set in polynomial time even with the full HVL-Prime mutation operator. Since the RLS-GP is able to accept solutions with identical fitness, it can reduce the number of copies of undesired variables via random walks, eventually allowing it to remove the last copy of each of the undesired variables. We begin by observing that the current solution does not increase beyond a certain size in the time required to fit the training set.

Lemma 17. *The number of terminals in the current solution of the RLS-GP algorithm, using the HVL-Prime mutation operator with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ on $\text{AND}_{n,m}$, remains below $3n$ during the first $O(n \log n)$ iterations with high probability.*

Having established a bound on the maximum size of the tree during the first $O(n \log n)$ iterations of the RLS-GP on $\text{AND}_{n,m}$, we can proceed to bound its runtime.

Theorem 18. *The RLS-GP algorithm, using the HVL-Prime mutation operator with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ will find the optimum of $\text{AND}_{n,m}$ in $O(n \log n)$ iterations with probability $1 - o(1)$.*

6.2 Dynamic polynomial-size training sets

A polynomially-sized training set can be used to produce a solution with a polynomially small generalisation error. In this case we restrict the maximum size of the tree the RLS-GP algorithm will accept (as is common in applications of GP to avoid the rapid increase of program size without significant return in fitness, i.e., bloat (Koza 1992; Poli, Langdon, and McPhee 2008)), and compare the fitness of two solutions by sampling S rows from the complete truth table independently at random in each iteration.

Theorem 19. *For any desired generalisation error n^{-c}, where $c > 0$ is a constant, the RLS-GP algorithms with a tree size limit $T_{\max} = c \log_2 n$ and sampling set size $S \in \Omega(n^{2c+1})$ can construct a solution with the desired generalisation error on the $\text{AND}_{n,m}$ problem in expectation in $O(cn \log(n) \log(m))$ iterations.*

We note that without the restriction on $T_{\max}$, it is possible for the GP's current solution to collect too many incorrect variables, which reduces the probability that mutations which increase/decrease the number of distinct correct variables are correctly identified as improving/reducing fitness, leaving the GP system to perform a random walk where it is easier to remove correct variables from the solution than to insert the missing correct variables.

7 Evolving exact solutions efficiently

For the AND family of problems analysed in the previous sections, we have shown that using randomly-selected polynomially-sized training sets yields solutions, which despite generalising well, are not equivalent to the target function. In this section we show the existence of small training sets which can be used to efficiently evolve a program that is exactly equivalent to the target function.

Consider a minimal training set M, consisting of n rows, where the i'th row sets x_i to false and all x_j (where $j \neq i$)

to true. We will show that using a static training set M (or a training set based on M) will allow the GP algorithms to construct an optimal solution efficiently.

Theorem 20. *The RLS-GP and* $(1+1)$ GP *algorithms using the training set* M *are able to find the optimal solution of AND*$_n$ *with* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n\}$ *in expected* $O(n \log n)$ *fitness evaluations (or* $O(n^2 \log n)$ *training set row evaluations)*

The minimal training set M can also be used for the RLS-GP when the target function is a conjunction of $m < n$ variables, as is the case in the AND$_{n,m}$ problem.

Theorem 21. *The RLS-GP algorithms using the training set* M *are able to find the optimal solution on AND*$_{n,m}$ *with* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n\}$ *in expected* $O(n \log n)$ *fitness evaluations (or* $O(n^2 \log n)$ *training set row evaluations).*

We expect that a similar result would also hold for the $(1+1)$ GP algorithms. However, since mutations which simultaneously insert both correct and undesired variables can occur and be accepted, and inserting copies of already-present undesired variables does not affect fitness, proving this would require a more complex random walk argument following the style of Theorem 18.

As it does not appear to be possible for the (1+1) GP to evolve a conjunction of n variables if their negations are also present in the terminal set (even when using the complete truth table, per Theorems 11 and 12), we show how a more careful choice of the training set can be beneficial.

Theorem 22. *Using a training set consisting of the* n *rows of the* M *training set and* $n+1$ *copies of the row setting all variables to true, and* $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$*, the RLS-GP and* $(1+1)$ GP *algorithms are able to construct the optimal solution to AND*$_n$ *in* $O(n \log n)$ *iterations.*

A similar effect can be achieved by using a population with an explicit diversity mechanism, rather than adding n additional copies of the all-true row. To the best of our knowledge this is the first time the benefits of using a population in GP systems have been rigorously proved. We consider the $(\mu+1)$ GP algorithm, which maintains a population of μ trees. In each iteration, an offspring is produced by selecting an ancestor uniformly at random from the current population and applying HVL-Prime $k = 1 + \text{Poisson}(1)$ times. If the fitness of the offspring is at least that of the worst individual in the population, it replaces that individual. With the phenotype diversity mechanism, offspring which exactly replicate the training set behaviour of any individual already present in the population are not accepted, even if their fitness is better than that of the worst individual in the population.

Theorem 23. *Using a training set consisting of the* n *rows of the* M *training set and a single copy of the row setting all variables to true, and* $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$*, the* $(\mu+1)$ *GP algorithm, with* $\mu \geq n+2$ *and using phenotype diversity, when initialised with* μ *empty solutions, construct the optimal solution in expected* $O(\mu n \log n)$ *iterations.*

Thus, we have shown that there exist training sets containing just $O(n)$ rows which allow the GP algorithms to construct the exact target function in polynomial time on the AND family of problems, even when the terminal set L contains negated literals or extra variables.

8 Conclusion

In this paper, a considerable step has been made towards the understanding of the working principles of general purpose GP systems for evolving programs with a given functionality. We presented a time and space complexity analysis of the RLS-GP and the $(1+1)$ GP algorithms for evolving Boolean conjunctions (i.e., the AND$_n$ problem). A fixed budget analysis for AND$_n$ provided a relationship between the expected error produced by the evolved program and the time the algorithm is allowed for the optimisation when local mutations are used. We made a considerable step forward towards the analysis of realistic GP systems by equipping the algorithms with more realistic mutation operators with large neighbourhoods, and by extending the function and terminal sets with more than just the minimal elements.

For AND$_n$ with minimal function and terminal sets we show that the $(1+1)$ GP and $(1+1)$ GP* algorithms produce a solution fitting the complete training set in $\Theta(n \log n)$ iterations and prove that this solution is of size $\Theta(n)$. When the size of the training set is limited to a polynomial of n, these GP systems produce logarithmically-sized solutions which generalise well.

Concerning the extended function set, when negated variables are also included in the terminal set for AND$_n$, the algorithms encounter great difficulties in fitting the complete training set. Nevertheless, the solutions have overwhelmingly good generalisation capabilities over the training set under uniform distribution. On the other hand, for extended terminal sets when the set of variables used by the target function is unknown to the GP algorithm (i.e. the AND$_{n,m}$ setting), we have demonstrated that the non-strictly elitist RLS-GP has an advantage over the RLS-GP*.

Overall, interesting characteristics of the considered benchmark function may be derived from the presented work. When using the minimal sets and the complete truth table, the problem is very similar to the ONEMAX coupon collecting benchmark problem used in evolutionary optimisation (Droste, Jansen, and Wegener 2002; Oliveto and Yao 2011). Hence, it is ideal as an easy benchmark function to evaluate the hillclimbing characteristics of GP systems.

However when smaller training sets are used, the problem characteristics change considerably. In a training set of polynomial size drawn uniformly at random from the complete truth table of AND$_n$, all the training set points return 0 in output with overwhelming probability (w.o.p.). This means that a constant function that always returns 0 will generalise well. We see this in our analysis where, with minimal terminal and function sets, a logarithmic number of conjunctions suffice to fit the polynomial training sets. These solutions are somewhat similar to those evolved in (Valiant 2009). However, when negated variables are also allowed, then constant functions that always return 0 are easily evolved. While these functions are trap points from which it is hard to escape on the complete training set, they fit a polynomial size training set (w.o.p.). In both cases the programs will return

the correct output (w.o.p.) over randomly drawn rows from the complete training set. It is fair, though, to assume that in many practical applications of AND_n circuits, the only input returning a true value occurs far more often than any other single input. An interesting future research direction is to consider such a scenario for the evolution of conjunctions.

Overall, defining an "easy" benchmark function to evaluate the generalisation capabilities of GP systems still remains an open problem. Benchmark functions where problems such as bloat and overfitting (Koza 1992) can be studied in further detail also need to be devised. Further directions for future work are to consider analyses of algorithms with more comprehensive terminal and function sets, along the way towards the analysis of more sophisticated and realistic population based GP systems.

Acknowledgements Financial support by the Engineering and Physical Sciences Research Council (EPSRC Grant No. EP/M004252/1) is gratefully acknowledged.

References

Al-Sahaf, H.; Al-Sahaf, A.; Xue, B.; Johnston, M.; and Zhang, M. 2017. Automatically evolving rotation-invariant texture image descriptors by genetic programming. *IEEE Transactions on Evolutionary Computation* 21(1):83–101.

Archetti, F.; Lanzeni, S.; Messina, E.; and Vanneschi, L. 2007. Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines* 8(4):413–432.

Doerr, B., and Goldberg, L. A. 2010. Drift analysis with tail bounds. In *Proceedings of the Parallel Problem Solving from Nature conference (PPSN XI)*, 174–183.

Doerr, B.; Kötzing, T.; Lagodzinski, J. A. G.; and Lengler, J. 2017. Bounding bloat in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017)*, 921–928.

Doerr, B.; Johannsen, D.; and Winzen, C. 2012. Multiplicative drift analysis. *Algorithmica* 64(4):673–697.

Doerr, B. 2011. Analyzing randomized search heuristics: Tools from probability theory. In Auger, A., and Doerr, B., eds., *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific. chapter 1, 1–20.

Droste, S.; Jansen, T.; and Wegener, I. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computuer Science* 276(1-2):51–81.

Durrett, G.; Neumann, F.; and O'Reilly, U. 2011. Computational complexity analysis of simple genetic programming on two problems modeling isolated program semantics. In *Proceedings of the Foundations of Genetic Algorithms workshop (FOGA 2011)*, 69–80.

Feldman, V. 2008. Evolvability from learning algorithms. In Dwork, C., ed., *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, 619–628. ACM.

Feller, W. 1968. *An introduction to probability theory and its applications*. Wiley.

He, J., and Yao, X. 2001. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence* 127(1):57–85.

Jansen, T., and Zarges, C. 2014. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science* 545:39–58.

Jansen, T. 2013. *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer.

Kötzing, T.; Sutton, A. M.; Neumann, F.; and O'Reilly, U. 2014. The MAX problem revisited: The importance of mutation in genetic programming. *Theoretical Computer Science* 545:94–107.

Koza, J. R. 1992. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press.

Koza, J. R. 2010. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines* 11(3-4):251–284.

Langdon, W. B., and Poli, R. 2002. *Foundations of genetic programming*. Springer.

Liu, L., and Shao, L. 2013. Learning discriminative representations from RGB-D video data. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 1493–1500.

Mambrini, A., and Oliveto, P. S. 2016. On the analysis of simple genetic programming for evolving boolean functions. In *Proceedings of Genetic Programming - 19th European Conference (EuroGP 2016)*, 99–114.

Mitzenmacher, M., and Upfal, E. 2005. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press.

Oliveto, P. S., and Witt, C. 2011. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica* 59(3):369–386.

Oliveto, P. S., and Witt, C. 2012. Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *arXiv:1211.7184*.

Oliveto, P. S., and Yao, X. 2011. Runtime analysis of evolutionary algorithms for discrete optimization. In Auger, A., and Doerr, B., eds., *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific. chapter 2, 21–52.

O'Neill, M.; Vanneschi, L.; Gustafson, S. M.; and Banzhaf, W. 2010. Open issues in genetic programming. *Genetic Programming and Evolvable Machines* 11(3-4):339–363.

O'Reilly, U.-M., and Oppacher, F. 1996. A comparative analysis of GP. In *Advances in Genetic Programming 2*. MIT Press. 23–44.

Poli, R.; Langdon, W. B.; and McPhee, N. F. 2008. *A Field Guide to Genetic Programming*. http://lulu.com.

Rowe, J. E., and Sudholt, D. 2014. The choice of the offspring population size in the (1, λ) evolutionary algorithm. *Theoretical Computer Science* 545:20–38.

Schuh, M. A.; Angryk, R. A.; and Sheppard, J. W. 2012. Evolving kernel functions with particle swarms and genetic programming. In *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2012)*.

Song, D.; Heywood, M. I.; and Zincir-Heywood, A. N. 2005. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Transactions on Evolutionary Computation* 9(3):225–239.

Valiant, L. G. 2009. Evolvability. *Journal of the ACM* 56(1).

Wagner, M.; Neumann, F.; and Urli, T. 2015. On the performance of different genetic programming approaches for the SORTING problem. *Evolutionary Computation* 23(4):583–609.