White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/154159/

Version: Published Version

Article:
Lissovoi, A. and Oliveto, P.S. (2019) On the time and space complexity of genetic programming for evolving Boolean conjunctions. Journal of Artificial Intelligence Research, 66. pp. 655-689. ISSN: 1076-9757

https://doi.org/10.1613/jair.1.11821

© 2019 AI Access Foundation. Reproduced in accordance with the publisher's self-archiving policy.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Journal of Artificial Intelligence Research 66 (2019) 655-689 Submitted 09/2018; published 11/2019

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

Andrei Lissovoi A.LISSOVOI@SHEFFIELD.AC.UK
Pietro S. Oliveto P.OLIVETO@SHEFFIELD.AC.UK
Department of Computer Science, University of Sheffield
Sheffield S1 4DP, United Kingdom

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 progress the GP makes toward finding the target function. 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. If negations are allowed, 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 with access to negations or 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*. New programs are created by applying variation operators, and natural selection principles are used to evolve a population of programs with the aim of eventually identifying a program with the desired functionality. Although during the evolution the quality of programs is measured using a training set of inputs, 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 Koza, 2010; Schuh, Angryk, & Sheppard, 2012; Liu & Shao, 2013; Al-Sahaf, Al-Sahaf, Xue, Johnston, & Zhang, 2017 for some recent examples), there is a limited understanding

©2019 AI Access Foundation. All rights reserved.

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, Vanneschi, Gustafson, & Banzhaf, 2010; Poli, Langdon, & 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 (which would allow solution quality to be evaluated in polynomial time) generalise well. If the number of available variables n is small, it may be possible to evaluate the candidate solutions on all of its 2^n rows in every iteration. In that case, the complete training set could be used to evaluate program quality, and fitting the training set is sufficient for efficient optimisation. If n is larger, running the candidate programs on 2^n inputs in every iteration may be computationally infeasible, in which case the generalisation capabilities when using a polynomially-sized training set 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 particular structures rather than programs with a given functionality (Lissovoi & Oliveto, 2019). In such settings, the evolved tree structures would not accept variable inputs, and their fitness would be based on the structure of the produced tree rather than on its execution (Langdon & Poli, 2002; Durrett, Neumann, & O'Reilly, 2011; Wagner, Neumann, & Urli, 2015; Doerr, Kötzing, Lagodzinski, & Lengler, 2017; Kötzing, Lagodzinski, Lengler, & Melnichenko, 2018). The analysis of the MAX problem (Kötzing, Sutton, Neumann, & O'Reilly, 2014) is one exception to this, where trees are used to represent arithmetic expressions which are evaluated; however, this problem still includes no input variables. 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 a terminal set consisting of n input variables. 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) proved 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

conjunction of n variables, while it requires exponential time with overwhelming probability to evolve a parity function of n variables. While the latter result is a consequence of XOR not being PAC-learnable (Valiant, 2009), the analysis provided mathematical tools to prove exponential runtimes for GP in cases where the algorithm is inefficient.

Compared to previous runtime analyses, AND_n and XOR_n are indeed proper functions with input/output behaviour: they accept a number of variables as input and produce an output value based on the values of the inputs. 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 number of distinct variables in 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 neighbourhoods. 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 generalise well with arbitrarily 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. We first extend the function set to not only contain the required AND operators but also an unnecessary negation operator (i.e., NOT). This setting was already used to show that by adding an unnecessary operator the RLS-GP becomes inefficient with overwhelming probability[1] on AND_n (Mambrini & Oliveto, 2016). 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. We also extend the terminal set such that it contains both necessary and 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

1. We analyse this issue in the framework of the $(1+1)$ GP algorithms and HVL-Prime mutation by introducing negated literals into the training set. GP systems using a NOT *function* are similarly inefficient, as the issue stems from the way solution fitness is computed for some locally-optimal solutions that are representable in both settings.

(Valiant, 2009). Our results show that the non-strict selection algorithm RLS-GP is able to find the optimal solution efficiently when using the complete truth table as the training set, while the strict selection algorithm RLS-GP* may fail with high probability unless the substitution operator is switched off. When program quality is evaluated by sampling a polynomial number of inputs uniformly at random from the complete truth table in each iteration, we show that RLS-GP is able to construct solutions with polynomially-small generalisation errors in polynomial time if a limit on the maximum tree size is in place as is common in GP applications.

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, precisely define the learning problems and measures of generalisation ability, and introduce the mathematical tools that will be used in our analyses. 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 the $(1+1)$ 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 linear training sets which allow 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.

2. Preliminaries

The following sections present results for a total of four simple GP algorithms in various settings. All four algorithms evolve programs using a syntax tree representation, and use the HVL-Prime mutation operator defined in Figure 1 (first proposed by O'Reilly & Oppacher, 1996) to construct offspring solutions in each iteration (Durrett et al., 2011; Kötzing et al., 2014; Mambrini & Oliveto, 2016). Figure 3 illustrates the three possible HVL-Prime sub-operations. 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 by 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 emphasise 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 (Durrett et al., 2011). Our notation matches the conventions of runtime analysis of evolutionary algorithms (Oliveto & Yao, 2011; Jansen, 2013).

We also consider the $(1+1)$ GP and $(1+1)$ GP* algorithms, which perform $k = 1 +$ Poisson(1) iterative HVL-Prime mutations to produce an offspring solution in each iteration.

Choose sub-operation $op \in \{\texttt{INSERT}, \texttt{DELETE}, \texttt{SUBSTITUTE}\}$ uniformly at random
if X is an empty tree **then**
    Choose a literal $l \in L$ uniformly at random
    Set l to be the root of X.
else if $op = \texttt{INSERT}$ **then**
    Choose a node $x \in X$ uniformly at random
    Choose $f \in F, l \in L$ uniformly at random
    Replace x with f
    Set the children of f to be x and l, order chosen u.a.r.
else if $op = \texttt{DELETE}$ **then**
    Choose a leaf node $x \in X$ uniformly at random
    Replace x's parent in X with x's sibling in X
else if $op = \texttt{SUBSTITUTE}$ **then**
    Choose a leaf node $x \in X$ uniformly at random
    Choose a literal $l \in L$ uniformly at random
    Replace x with l.

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

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

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.

The extended mutation operator has been previously considered on other problems (Kötzing et al., 2014; Durrett et al., 2011).

Recall that the probability density function of a Poisson-distributed variable with parameter $\lambda = 1$ is:

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

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 instances based on the AND function: the objective of the GP systems will be to evolve a tree which computes the conjunction $x_1 \wedge \ldots \wedge x_n$ (the AND_n problem), or a conjunction of a subset of $m < n$ available variables (the $\text{AND}_{n,m}$ problem). For Boolean functions of n variables, the complete training set is

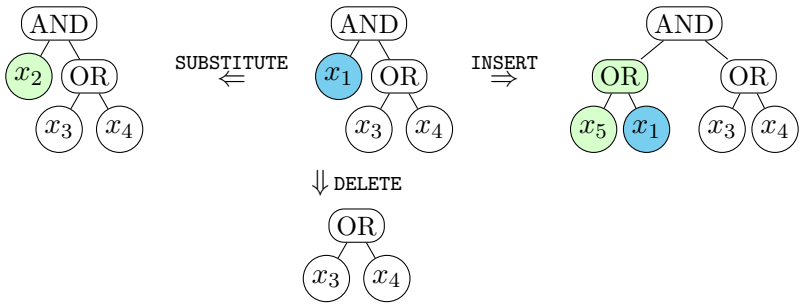

Figure 3: HVL-Prime sub-operations: substitution, insertion, and deletion.

a truth table with 2^n rows. Solution quality is measured by the number of inputs (either among all possible inputs, or among a subset that possibly depends on the iteration t) for which the output of the current solution and the target function differs.

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:*

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

Equivalently, error(h) can be thought of as the ratio of the rows of the complete truth table on which the solution h returns a different value from the target function $\hat{h}$.

When using the complete truth table as the training set, the fitness of a solution h is the number of inputs on which its output differs from the target function $\hat{h}$, i.e., $f_t(h) = 2^n \cdot \text{error}(h)$. In this formulation, lower $f_t(h)$ values are better, and an optimal solution has a fitness value of 0.

For AND_n and $\text{AND}_{n,m}$, when $h(x)$ is a conjunction of any subset of variables $x_1, \ldots, x_n$, it is possible to quickly calculate error(h) without evaluating the full 2^n rows of the truth table (per Propositions 2 and 18). 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 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.

We consider two ways of randomly selecting the training set of size s from the complete truth table of the target function: 1) *static training set*: a set of inputs is chosen at the beginning of the optimisation process and used for all fitness evaluations (thus making all $f_t(X)$ functions equivalent), or 2) *dynamic training set*: a new sample is taken at every iteration of the GP algorithm, and used for fitness evaluations within that iteration only (allowing $f_t(X)$ to change with t). This reflects the scenarios where only a limited amount of information about the target function is available (such as in medical applications, Archetti, Lanzeni, Messina, & Vanneschi, 2007), and where a prohibitively large amount of information is available to be used in every fitness evaluation (such as for problems involving large data sets, Song, Heywood, & Zincir-Heywood, 2005).

For some of our results, we additionally impose a limit on the maximum acceptable tree size $T_{\max}$, and reject any offspring that contain more than $T_{\max}$ leaf nodes. Tree size limits are typically applied in practice to avoid issues arising from bloat, a common problem where the size of the solution tends to increase without a corresponding increase in solution quality (Koza, 1992; Poli et al., 2008).

Note that the class of AND functions (in particular, $\text{AND}_{n,m}$) is evolvable under the 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. Ideally all 2^n rows should be sampled to evaluate the exact quality of a candidate solution. However, since this becomes computationally prohibitive for large problem sizes we also consider whether efficient evolution is possible when solution quality is evaluated by sampling a polynomial number of possible inputs instead.

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.*

Proof. The conjunction evaluates to true on the 2^{n-v} rows where all v present variables are true, while the target function evaluates to false on all but one of these truth table rows (the single row where all variables are true). □

2.1 Runtime Analysis Tools

The following theorems are useful mathematical tools that will be used in the analyses presented in this paper. The drift analysis results presented in this subsection are based on the excellent overview by Lengler (2019), which also includes proofs and application examples.

We use three drift analysis results in particular: the additive drift theorem (Theorem 3) concerns processes which progress toward their goal at a constant expected pace, the multiplicative drift theorem (Theorem 4) concerns processes which slow down their expected rate of progress as they approach the goal, and the negative drift theorem (Theorem 5) concerns processes which in expectation do not make progress toward the goal.

Theorem 3 (Additive drift, He & Yao, 2001, 2004)**.** *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S} \subseteq \mathbb{R}_0^+$ such that $0 \in \mathcal{S}$. Let $T := \inf\{t \geq 0 \mid X_t = 0\}$.*

If there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and for all $t \geq 0$, $\Delta_t(s) := E(X_t - X_{t+1} \mid X_t = s) \geq \delta$, then

$$E(T) \leq \frac{E(X_0)}{\delta}.$$

If there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and for all $t \geq 0$, $\Delta_t(s) := E(X_t - X_{t+1} \mid X_t = s) \leq \delta$, then

$$E(T) \geq \frac{E(X_0)}{\delta}.$$

Theorem 4 (Multiplicative drift, Doerr & Goldberg, 2010; Doerr, Johannsen, & Winzen, 2012)**.** *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables with a finite state space $\mathcal{S} \subseteq \mathbb{R}_0^+$ such that $0 \in \mathcal{S}$. Let $s_{\min} := \min(S \setminus \{0\})$, let $T := \inf\{t \geq 0 \mid X_t = 0\}$, and for $t \geq 0$ let $\Delta_t(s) := E(X_t - X_{t+1} \mid X_t = s)$.*

Suppose there exists $\delta > 0$ such that for all $s \in \mathcal{S} \setminus \{0\}$ and all $t \geq 0$ the drift is $\Delta_t(s) \geq \delta s$. Then,

$$E(T) \leq \frac{1 + E(\ln(X_0/s_{\min}))}{\delta}.$$

Moreover, for any $r > 0$, it holds that

$$\Pr(T > (\ln(X_0/s_{\min}) + r)/\delta) \leq e^{-r}.$$

Theorem 5 (Negative drift, Oliveto & Witt, 2011, 2012; Rowe & Sudholt, 2014)**.** *For all $a, b, \delta, \eta, r > 0$, with $a < b$, there is $c > 0, n_0 \in \mathbb{N}$ such that the following holds for all $n \geq n_0$. Suppose $(X_t)_{t\geq 0}$ is a sequence of random variables with a finite state space $\mathcal{S} \subseteq \mathbb{R}_0^+$, and with associated filtration $\mathcal{F}_t$. Assume $X_0 \geq bn$, and let $T_a := \min\{t \geq 0 \mid X_t \leq an\}$ be the hitting time of $S \cap [0, an]$. Assume further that for all $s \in \mathcal{S}$ with $s > an$, for all $j \in \mathbb{N}_0$, and for all $t \geq 0$, the following conditions hold:*

$$E(X_t - X_{t+1} \mid \mathcal{F}_t, X_t = s) \leq -\delta,$$
$$\Pr(|X_t - X_{t+1}| \geq j \mid \mathcal{F}_t, X_t = s) \leq \frac{r}{(1+\eta)^j}.$$

Then,

$$\Pr(T_a \leq e^{cn}) < e^{-cn}.$$

Additionally, results concerning the coupon collector process are useful when bounding the time for a GP system to insert a certain number of distinct variables into its solution. The theorem below is based on the formulation by Doerr (2019).

Theorem 6 (Coupon collector process, Feller, 1968; Doerr, 2019)**.** *Assume that there are n types of coupons available, and one coupon of a type chosen uniformly at random from the n types is acquired each time step. Let T_n be the expected number of time steps before at least one coupon of each type has been collected. Then,*

$$E(T_n) = \sum_{k=0}^{n-1} \frac{n}{n-k} = n \sum_{k=1}^{n} \frac{1}{k} = (1 + o(1))n \ln n,$$

and, for all $\varepsilon \geq 0$,

$$\Pr(T_n \leq (1-\varepsilon)(n-1)\ln n) \leq e^{-n^\varepsilon}.$$

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 by Mambrini and Oliveto (2016), who have 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 number of distinct variables present in the solution, and the time the algorithms are allowed to run. The proofs follow the techniques used by Jansen and Zarges (2014) to analyse Randomised Local Search for the OneMax problem.

Theorem 7. *For all budgets b, the expected number of distinct variables in the solution produced by the RLS-GP* using $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ on AND_n when initialised with an empty tree is: $E(v(x_b)) = n - n(1 - 1/(3n))^b$.*

Proof. We note that RLS-GP* does not accept insertions of variables which are already present in its current solution, as such mutations do not change the behaviour of that solution, and hence do not result in a fitness improvement. Therefore, the solution contains no duplicate variables, and thus neither deletions nor substitutions can be accepted, as reducing (or, in the best case for substitution, not affecting) the number of distinct variables in the current solution does not improve fitness.

A variable x_i is inserted into the current solution with probability $1/(3n)$ in each iteration; the probability that it has not been inserted within t iterations is $(1 - 1/(3n))^t$. Let $X_{i,t} \in \{0, 1\}$ be random variables such that $X_{i,t} = 0$ if and only if x_i is in the solution at time t; then, by linearity of expectation and symmetry, the expected number of variables still missing from the solution at time t is:

$$E(v(x_t)) = n - E\left(\sum_{i=1}^{n} X_{i,t}\right) = n - \sum_{i=1}^{n} E(X_{i,t}) = n - n(1 - 1/(3n))^t,$$

which proves the theorem. □

The RLS-GP will accept mutations inserting additional copies of variables already present in the tree (as these mutations do not affect the solution's fitness value). When the tree contains multiple copies of a single variable, the substitution sub-operator of HVL-Prime can also increase the number of distinct variables in the tree (by substituting a copy with a new variable). In this setting, we can provide upper and lower bounds on the performance of the RLS-GP: the result of Theorem 7 provides a lower bound on the number of distinct variables in the solution at time t, and Theorem 8 provides an upper bound based on the substitution operator always selecting a redundant variable for substitution.

Theorem 8. *For all budgets b, the expected number of distinct variables in the solution produced by the RLS-GP using $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ on AND_n when initialised with an empty tree is:*

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

Proof. We note that unlike the RLS-GP*, the RLS-GP algorithm accepts mutations which add further copies of variables already present in the current solution, which allow it to also accept deletions and substitutions targeting variables which appear in the current solution more than once. We disregard substitutions targeting variables which appear in the solution exactly once, as these would either be rejected (if the replacement variable already appears

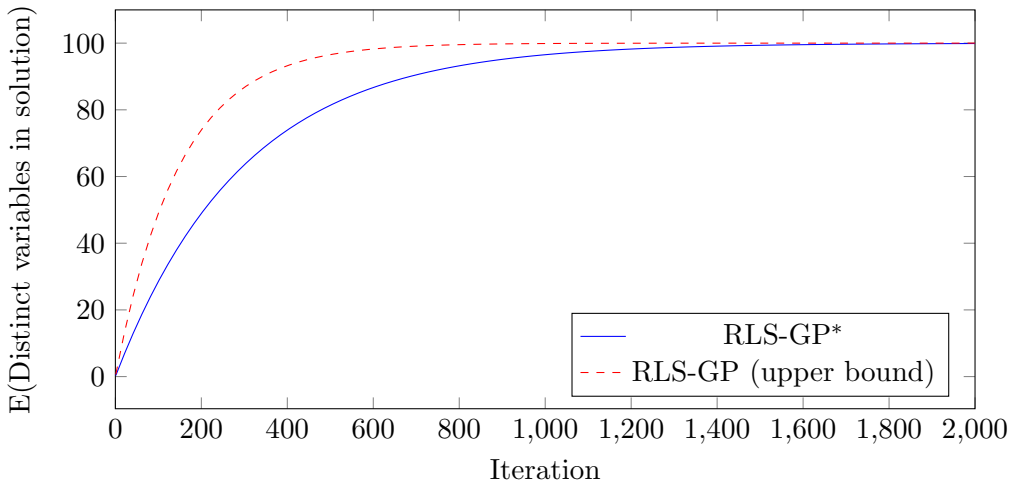

Figure 4: Expected number of distinct variables in the solutions constructed by the RLS-GP algorithms as a function of the number of iterations, for $n = 100$.

in the solution), or have no effect on the progress of the GP (if the replacement variable was not present in the current solution).

Thus, the probability p that a new variable x_i is added into the current solution depends on the probability that substitution manages to find a suitable terminal to replace, leading to:

$$(1 - 1/(3n))^b \leq p \leq (1 - 2/(3n))^b$$

and hence, in a manner similar to Theorem 7,

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

□

Figure 4 provides an illustration of the bounds from Theorem 7 and Theorem 8.

4. (1+1) GP on AND_n

In this section we present a runtime analysis of the $(1 + 1)$ GP and $(1 + 1)$ GP* algorithms, which apply the HVL-Prime mutation operator $k = 1 + \text{Poisson}(1)$ times before evaluating the quality of the offspring solution. We initially analyse these algorithms using the complete truth table as the training set, and then using training sets of polynomial size chosen uniformly at random either at the beginning of the optimisation process, or independently in each iteration.

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 9. *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 the training set in time* $\Theta(n \log n)$.

Proof. Neither algorithm will accept mutations which decrease the number of distinct variables in the current solution.

We divide the search space into fitness levels $A_1, \ldots, A_n$ such that each level A_i contains all the conjunctions with i distinct variables. To advance to fitness level A_{i+1}, a single-insertion mutation introducing a new distinct variable is sufficient, and occurs with probability $p_i \geq \frac{n-i}{3ne}$. By the artificial fitness level method (Oliveto & Yao, 2011), the expected runtime is $E(T) \leq \sum_{i=1}^{n} 1/p_i = 3en \sum_{i=1}^{n} \frac{1}{i} = O(n \log n)$.

For a lower bound on the expected runtime, using a lower tail bound on the coupon collector process (Theorem 6) yields that with probability at least $1 - e^{-\sqrt{n}}$, there is at least one variable which has never been added to the solution within $1/2 \cdot (n-1) \log n$ HVL-Prime sub-operations (pessimistically assuming that every sub-operation adds a variable chosen uniformly at random). The number of HVL-Prime sub-operations the $(1+1)$ GP performs within t iterations can be bounded by applying Feige's inequality (Doerr, 2019, Lemma 10.19) to bound the sum of t independent Poisson(1)-distributed random variables; doing so yields that with probability at least $1/13$, the $(1+1)$ GP will perform at most $1/2 \cdot (n-1) \log n$ HVL-Prime sub-operations within $1/4 \cdot (n-1) \log n - 1$ iterations. Thus, with at least constant probability, the $(1+1)$ GP requires more than this number of iterations to find the optimal solution, and by the law of total expectation, the expected number of iterations before an optimal solution is found is therefore at least $\Omega(n \log n)$. □

We additionally prove that the expected number of leaf nodes in the solutions produced is $\Theta(n)$.

Theorem 10. *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 the training set. The expected number of terminals in the produced solution is* $\Theta(n)$.

Proof. We first remark that any solution which fits the complete training set must contain all n variables at least once, and thus contains at least $\Omega(n)$ terminals. In the remainder of the proof, we focus on deriving upper bounds on the number of terminals in trees constructed by the $(1+1)$ GP* and $(1+1)$ GP algorithms.

To bound the size of the solution for the $(1+1)$ GP*, we note that only mutations introducing new distinct variables are accepted, and these can occur at most n times. The size of the resulting tree is the sum of the changes in the size of the tree over the at-most n accepted mutations. Let p_i be the probability that an HVL-Prime sub-operation inserts a missing variable, and $P_{i,k}$ be the probability that a mutation performs k sub-operations and at least one of these is an insertion of a missing variable:

$$P_{i,k} = \frac{1 - (1 - p_i)^k}{e(k-1)!},$$

where the numerator reflects the probability of inserting at least one missing variable in k HVL-Prime sub-operations, and the denominator reflects the probability of performing k sub-operations i.e. that k is chosen using the Poisson distribution.

Let X be a random variable denoting the number of HVL-Prime sub-operations in a mutation which inserts a missing variable. In expectation,

$$E(X) = \sum_{k=1}^{\infty} k \frac{P_{i,k}}{\sum_{j=1}^{\infty} P_{i,j}} = \frac{2e^{p_i} + 3p_i - p_i^2 - 2}{e^{p_i} + p_i - 1}$$

We note that $\frac{\mathrm{d}E(X)}{\mathrm{d}x} = \frac{-1+2p_i-{p_i}^2+e^{p_i}(1-3p_i+p_i^2)}{(e^{p_i}+p_i-1)^2} < 0$ for all $0 < p_i \leq 1$, and thus $E(X)$ is monotonically decreasing with respect to p_i. Furthermore, $\lim_{p_i \to 0} E(X) = 5/2$, and hence we can conclude that $E(X) \leq 2.5$. We use this as an upper bound on the expected change in the tree size caused by an accepted mutation, pessimistically assuming that all performed sub-operations are insertions. Thus, the expected size of the tree after all distinct variables have been added to it is at most $2.5n = O(n)$ by linearity of expectation.

To bound the size of the solution for the $(1+1)$ GP, we need a more complicated argument, as the algorithm is able to accept solutions which increase the size of the tree without a corresponding fitness improvement. To produce a bound, we will argue that within the time required to construct the optimal solution, the GP system will be able to use the deletion sub-operations to keep the size of the tree within a constant factor of n.

Let S be the number of terminals in the solution which fits the complete training set, and let M_z be the event that no single mutation inserts more than z copies of any variable before an optimal solution is found, where z is a sufficiently large constant. We apply the law of total expectation:

$$E(S) = \Pr(M_z)E(S \mid M_z) + (1 - \Pr(M_z)) \cdot E(S \mid \overline{M_z}),$$

and picking e.g. $z \geq 4$ to bound $(1 - \Pr(M_z)) = O(n^{-2})$, and $E(S \mid \overline{M_z}) = O(n \log n)$ (as the expected number of HVL-Prime sub-operations performed before finding the optimal solution is $O(n \log n)$, and each sub-operation can add at most one terminal), we bound the second term by $o(n)$. In the remainder of the proof, we focus on bounding $E(S \mid M_z) = O(n)$ to show $E(S) = O(n)$.

In order for the tree to contain more than $(2z+1)n$ terminals, a mutation must cause at least one variable to exceed $2z+1$ copies, and at the time this mutation occurred, the solution must already contain at least $z+1$ copies of the variable. In that case, both mutations which delete $i \leq z$ or insert $i \leq z$ copies of such variables will be accepted, as neither would affect the fitness value of the current solution. When a mutation is accepted, we consider the HVL-Prime sub-operations which affect variables with at least $2z+1$ copies in the tree. Suppose there are j such variables, so the probability that a sub-operation is an insertion of an additional copy is exactly $j/(3n)$, and the probability that a sub-operation is a deletion of an existing copy is at least $j/(3n)$ (as the remaining $n-j$ variables each have fewer than $2z+1$ copies in the tree, the probability that deletion selects one of the j variables with at least $2z+1$ copies is at least j/n); thus, each sub-operation is at least as likely to delete a terminal of a variable with at least $2z+1$ copies as it is to insert such a terminal.

The size of the solution at any time is therefore dominated by $(2z+1)n$ (i.e. assuming that each variable has $2z+1$ copies in the tree) plus a contribution from a fair random walk with a reflecting lower boundary, where every mutation accepted by the $(1+1)$ GP

causes the walk to perform an expected $O(1)$ steps (one for each performed HVL-Prime sub-operation affecting a variable with at least $2z+1$ copies in the tree at the time the sub-operation occurred; which for some mutations may be 0).

All that remains to be proven is that in the $O(n \log n)$ iterations required to add all distinct variables to the tree, the random walk contribution remains below n with probability $1 - o(1)$. This can be shown by applying Gambler's Ruin (Feller, 1968; Oliveto, He, & Yao, 2007) repeatedly: each time the contribution of the random walk becomes positive, we apply Gambler's Ruin to show that in expected n iterations, the game will end, and with probability $1 - 1/n$, the contribution from the walk will return to 0 (rather than reach n). In $O(n \log n)$ iterations, $O(\log n)$ Gambler's Ruin games will occur in expectation by an application of the additive drift theorem (Theorem 3), using $O(n \log n)$ as the initial distance, and $\Omega(n)$, i.e. the expected duration of a single game, as the drift to bound the expected number of games within the time budget. Markov's inequality can then be used to show that no more than $O(\log^2 n)$ games occur with probability $1 - 1/(\log n) = 1 - o(1)$. Finally, a union bound over the $O(\log^2 n)$ games each having a $1/n$ probability of reaching a contribution of n, and the Markov bound, yields the desired result: with probability at least $1 - \log(n)^2/n - 1/(\log n) = 1 - o(1)$, the random walk contribution never reaches n in the time required to find the optimal solution.

Thus, the expected number of terminals in the solution constructed by the $(1+1)$ GP is at most $(2z+2)n = O(n)$. □

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. In this subsection, we consider the training set to remain fixed throughout the process, to reflect a situation where only a limited amount of information about the target function's input/output behaviour is available. 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 prove 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 for the AND_n problem.

Theorem 11. *Let a training set of $s = \text{poly}(n)$ rows be drawn uniformly at random with replacement from the complete truth table of AND_n at the beginning of the run. The $(1+1)$ GP* and $(1+1)$ GP algorithms, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$, will fit the training set 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 $1 - n^{-c}$.*

Proof. When a row is drawn uniformly at random from the complete truth table of AND_n, variables in it are false independently with probability $1/2$, making the number of variables set to false within the row binomially distributed with parameters n and $1/2$. Using a Chernoff bound, we conclude that the probability that such a row sets fewer than $n/4$ variables to false is at most $e^{-n/8}$; and, using a union bound, the probability that all of the polynomially many rows in the training set contain at least $n/4$ variables set to false is at least $1 - \text{poly}(n)e^{-n/8} = 1 - e^{-\Omega(n)}$.

Suppose each row of the training set contains at least $n/4$ variables set to false. We can then apply the multiplicative drift theorem (Theorem 4) to bound the expected time it takes the $(1+1)$ GP* and $(1+1)$ GP algorithms to fit the training set.

Let X_t be the fitness value (i.e. the number of training set rows on which the current solution does not match the target function) at iteration t. Given that $X_t > 0$, there exists at least one row on which the current solution evaluates to true, while the target function evaluates to false. As each row sets at least $n/4$ variables to false, there must exist at least $n/4$ variables not yet present in the current solution, which, if added, would improve the solution's fitness value. As training set rows are sampled from the complete truth table uniformly at random with replacement, the probability that any particular variable is set to false in any particular row is $1/2$, and in expectation, when a new distinct variable is added to the current solution, the number of rows on which the solution does not match the target function is halved. Therefore, there exists a multiplicative drift toward $X_t = 0$ via single-variable additions:

$$E(X_t - X_{t+1} \mid X_t > 0) \geq \frac{X_t}{2 \cdot 4 \cdot 3 \cdot e}$$

where the multiplicative constants stem respectively from the probability that adding a new variable causes a row to fit the target function ($1/2$), the probability of inserting one of the at least $n/4$ still-missing variables ($(n/4)/n = 1/4$), the probability of HVL-Prime selecting an insertion operator ($1/3$), and the probability of the mutation operator performing a single HVL-Prime sub-operation ($1/e$). As neither the $(1+1)$ GP* nor the $(1+1)$ GP accept mutations which reduce the quality of the current solution, this serves as a lower bound on the multiplicative drift.

Initially, $X_0 \leq \text{poly}(n) \leq n^k$ for some sufficiently-large constant k. Applying Theorem 4, the expected first-hitting time $T := \inf\{t \geq 0 \mid X_t = 0\}$ is, for any starting point $X_0 \leq n^k$:

$$E(T) \leq \frac{1 + \log(n^k)}{1/(24e)} = 24e\,(1 + k\log(n)) = O(\log n).$$

If some training set row sets fewer than $n/4$ variables to false, we note that the hardest training set to fit has exactly one variable set to false in each row, requiring the solution to accumulate all of the up to n distinct variables. Applying Theorem 6, we conclude that the $(1+1)$ GP* and $(1+1)$ GP algorithms fit such a training set in time $O(n \log n)$.

Combining the two cases, the expected number of iterations required by the $(1+1)$ GP* and $(1+1)$ GP algorithms to fit a polynomially large training set drawn with replacement from the complete truth table of AND_n is:

$$\begin{aligned} E(T) &\leq (1 - e^{-\Omega(n)})O(\log n) + e^{-\Omega(n)}O(n \log n) \\ &= O(\log n). \end{aligned}$$

For the high-probability bound, we use the tail bounds of Theorem 4: the probability that the optimisation time exceeds $\delta^{-1}(\ln n + c \ln n) = 24e(\ln n + c \ln n) = O(\log n)$ is at most n^{-c} for any choice of $c > 0$. □

4.2.1 Generalisation Ability

Mambrini and Oliveto (2016) 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 13. However, our Theorem 11 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 12. *Let a training set of $s = \text{poly}(n)$ rows, $s \geq n^{c'}$, where $c' > 0$ is an arbitrary constant, be drawn uniformly at random with replacement from the complete truth table of AND_n at the beginning of the run. The solution produced by the $(1+1)$ GP* and $(1+1)$ GP algorithms, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ in expectation contains $\Omega(\log n)$ distinct literals.*

Proof. We will show that it is unlikely that the GP algorithms are able to find a solution that fits a training set of $n^{c'}$ rows, where $c' > 0$ is a constant, using fewer than $c' \log n$ distinct variables in the time it takes to find a solution that fits the training set. Thus the expected number of variables present in the solution produced by the GP algorithms is at least logarithmic.

Consider the probability that a solution with v distinct literals produces the correct solution on all s training set rows. If the target function evaluates to 0 on a truth table row (which holds for all but one complete truth table row, and hence with overwhelming probability for all rows of a polynomially large training set), a candidate solution matches the target function if at least one of its v distinct variables is set to 0 in this row, which occurs with probability $1 - 2^{-v}$. Thus, a solution with v distinct literals matches the target function on all s training set rows with probability $(1 - 2^{-v})^s$.

From Theorem 11, we know that the GP algorithms find a solution that fits the training set in $O(\log n)$ iterations with high probability. We can then use a union bound to upper-bound the probability that the GP finds a solution with $v \leq c' \log n$ distinct literals that fits the training set during the $c \log n$ iterations it takes to find a solution with high probability:

$$\begin{aligned} 1 - (1 - (1 - 2^{-v})^s)^{c \log n} &\leq c \log n (1 - 2^{-v})^s \\ \leq c \log(n) e^{-s/2^v} &\leq c \log(n) e^{-s n^{-c' \log 2}} \\ &\leq e^{-s n^{-c' \log 2 + \log(c \log n)}} \end{aligned}$$

which is at most $e^{-n^{\Omega(1)}}$ when $s \in \Omega(n^{c'})$.

Thus, the probability of a fitting solution with fewer than $c' \log n$ distinct variables being found is close to 0, and hence the expected number of distinct variables in a solution constructed by the GP algorithms is at least $\Omega(\log n)$. More formally, if S is the number of distinct variables in the solution that fits the entire training set, and T is the event that GP finds a solution within $c \log n$ iterations for some constant $c > 0$, we apply the law of

total expectation:

$$\begin{aligned} E(S) &\geq E(S \mid T) \cdot \Pr(T) \\ &\geq c' \log n \cdot (1 - \Pr(S \leq c' \log n \mid T)) \Pr(T) \\ &\geq c' \log n (1 - e^{-n^{\Omega(1)}})(1 - n^{-\Omega(1)}) \end{aligned}$$

which is in $\Omega(\log n)$ for $s \geq n^{c'}$. □

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 13. *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 at least $1 - n^{-c}$.*

Proof. Recall that, per Proposition 2, the fitness of such a conjunction of $v = c \log n$ distinct variables is $2^{n-v} - 1$, i.e. the conjunction does not match the target function on $2^{n - c \log n} - 1$ rows.

The probability that a row selected uniformly at random from the complete truth table of AND_n is a row on which the conjunction does not match the target function is

$$\frac{2^{n - c \log n} - 1}{2^n} \leq 2^{-c \log n} = n^{-c}$$

Thus, a conjunction of $c \log n$ variables is with polynomially high probability going to produce a correct output on a randomly sampled row of the complete truth table of AND_n. □

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. The use of small training sets reduces the computational cost of evaluating the quality of the candidate solutions when the number of available input/output examples describing the behaviour of the target function is too large to be practical. This models practical GP systems that periodically update a training set of reasonable size used to evaluate solution quality by sampling from a larger pool of examples. Examples of such systems are Random Subset Selection (Gathercole & Ross, 1994) where each example is selected to be in the current subset of training cases independently with equal probability in each iteration. Also more sophisticated GP systems exist that attempt to learn which subsets of the complete data set should be preferred for inclusion in the training set. Examples of this include Dynamic Subset Selection and Historical Subset Selection (Gathercole & Ross, 1994; Song et al., 2005; Curry, Lichodzijewski, & Heywood, 2007).

Theorem 14. *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. With overwhelming probability, 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.*

Proof. Let $S = n^k$ rows, where $k > 0$ is a sufficiently large constant depending on c, be sampled from the complete truth table in each iteration. A solution which contains $k \log_2 n$ distinct variables, achieves an error of 0 on a sampled training set of size n^k with at least constant probability, as in order for a row to be wrong, it must set all $k \log_2 n$ present variables to 1 and at least one other variable to 0. The probability that no such row is sampled from the complete truth table in a given iteration is at least:

$$(1 - 2^{-k \log_2 n})^S = (1 - n^{-k})^{n^k} > 1/(2e)$$

for $n \geq 2$. Thus, after an expected constant number of iterations with solutions containing at least $k \log_2 n$ distinct variables, a training set will be found on which a solution reports an error of 0.

We note that the probability that a mutation increasing the number of distinct variables in a solution is accepted is at least as high as the probability that a mutation decreasing the number of distinct variables is accepted, since there is a higher probability of sampling rows on which the increasing mutation is correct. Additionally, while the current solution contains at most $2k \log_2 n$ distinct variables, the probability that insertion or substitution sub-operations choose to insert a variable that is already present in the solution (leading to either failing to increase or decreasing the number of distinct variables respectively) is at most $2k \log_2(n)/n$. If this does not occur, insertion operations will increase the number of distinct variables, substitution operations will pessimistically not alter the number of distinct variables, and deletion operations will decrease the number of distinct variables in the current solution.

Conditioning on insertion and substitution not failing to insert a new variable each time, the number of distinct variables in the solution is at least as likely to increase as it is to decrease, and can thus be modelled as performing a fair random walk. For RLS-GP, the number of distinct variables in the solution always changes by 1; we also use this as a pessimistic upper bound on the changes by $(1+1)$ GP. Recall that the expected number of steps for a fair random walk on natural numbers with a reflecting boundary at 0 to reach a value of t is t^2 (Feller, 1968; Mitzenmacher & Upfal, 2005). The expected number of changes before the number of distinct variables in the solution increases from 0 to $k \log_2 n$ is thus $(k \log_2 n)^2$. A step of this process in expectation takes at most 6 iterations of the RLS-GP, e.g. by requiring that an insertion (occurs with probability 1/3) of a new distinct variable (conditioned on never inserting a non-distinct variable) is accepted (with probability at least 1/2 for non-strictly elitist GPs). For the $(1+1)$ GP, such a step of this process in expectation takes at most $6e$ iterations (as with probability $1/e$, $(1+1)$ GP performs a single HVL-Prime sub-operation in an iteration).

We note that the probability that insertion or substitution fail to insert a distinct variable at least once over $t = c'(k \log_2 n)^2$ mutations (where $c' > 0$ is a constant) is at most

$t \cdot (2k \log_2(n)/n) = O(\log^3 n/n) = o(1)$. Thus, after expected $O((k \log_2 n)^2)$ iterations, the RLS-GP has found a solution with $k \log_2 n$ distinct variables, and after an additional constant number of iterations, will also find a training set on which this solution (or one of its offspring) evaluates to an error of 0. □

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 might also occur when NOT is added to the function set.

In particular, we show that, similarly to the results for the RLS-GP algorithms with local-search mutation (Mambrini & Oliveto, 2016), the $(1+1)$ GP algorithms are also unable to find the optimal solution using the complete truth table in polynomial time, due to constructing a solution containing a contradiction (i.e., $x_i \wedge \overline{x_i} \wedge \ldots$ for some variable x_i). However, such solutions have an almost-perfect generalisation ability on AND_n, being wrong on only one input. Additionally, we will show in Corollary 17 that the generalisation performance of the $(1+1)$ GP when using a realistic training set of polynomial size is unaffected by the inclusion of the negated variables, and the GP system is still able to produce solutions with any polynomially-small error probability in polynomial time.

Theorem 15. *The* $(1+1)$ GP* *using* $F = \{AND\}$ *and* $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$, *using the complete truth table as the training set, does not fit the training set for* AND_n *in polynomial time with overwhelming probability.*

Proof. It is important to note that despite being incorrect with respect to the target function, adding negations to a non-optimal solution will generally improve the fitness of that solution. This occurs as a consequence of the complete truth table of AND_n and the use of conjunction operators within the tree: the target function evaluates to false on all but one of the 2^n rows of the truth table; adding a negated variable will cause the solution to not match the target function on the all-true row, but will cause it to match the target function on half of the previously-wrong rows (where the variable is set to 1). Thus, the $(1+1)$ GP will accept mutations adding negations unless the current solution is already optimal, and the $(1+1)$ GP* will accept mutations adding negations unless the current solution already contains $(n-1)$ distinct non-negations.

Suppose the tree contains $n/2$ distinct terminals; the probability that it does not contain a contradiction (that is, both x_i and $\overline{x_i}$ for some i) is then

$$\prod_{i=1}^{n/2-1} \frac{2n-2i}{2n-i} < \prod_{i=n/3}^{n/2-1} \frac{2n-2n/3}{2n-n/2} < (8/9)^{n/6-1}$$

as each time a new distinct terminal is added to the tree, both that terminal and its negation can no longer be inserted into the tree to increase the number of distinct variables without

introducing a contradiction. This means that with overwhelming probability, the tree does contain both x_i and $\overline{x_i}$ for some i upon reaching $n/2$ distinct terminals. The fitness of such a tree is exactly 1: it evaluates to false on all rows, while the target function evaluates to false on all but the one all-true row.

Once the tree contains a contradiction, the $(1+1)$ GP* will only accept mutations which produce an optimal solution. As the tree with $n/2$ or more distinct terminals is overwhelmingly likely to contain a contradiction, and the last mutation before the tree reaches $n/2$ or more distinct terminals is overwhelmingly unlikely to insert $n/4$ or more distinct literals, to produce the optimum, a mutation must simultaneously add all of the at least $n/4$ missing variables, and remove any negations present in the tree. The probability that a mutation with at least $n/4$ operations occurs is at most $\frac{2}{e(n/4)!} \leq 2^{-\Omega(n)}$; thus, the $(1+1)$ GP* requires an exponential amount of time to find the optimum after finding a contradiction. □

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

Proof. As in the proof of Theorem 15, the current solution will with overwhelming probability contain a contradiction before it contains $n/2$ distinct terminals.

After constructing a tree with a contradiction, the $(1+1)$ GP will accept any mutation which does not remove the last remaining contradiction, as well as any mutation which produces an optimal solution. We note that the drift on the size of the tree is at least 0 (as insertions are at least as likely as deletions, and deletions which remove the last contradiction while the tree does not contain all the positive literals are rejected), and so, using a Gambler's Ruin argument, it will reach size t in expected $O(t^2)$ iterations for any value of t.

While the tree is able to grow to an arbitrary size through a random walk, and thus will in expected polynomial time contain all positive literals with high probability, at least a constant fraction of the terminals in the tree at any time will be negations with high probability. In order to find the optimal solution, all negated literals must be removed from the tree, which becomes increasingly difficult as the fraction of negated literals in the tree decreases, providing a strong negative drift that cannot be overcome in polynomial time with high probability.

Suppose that at some point during the optimisation process, the tree contains at least $X \geq n$ not necessarily unique positive literals. As insertions of positive and negated literals occur and are accepted at the same rate, and deletions remove positive literals at least as often as negated literals while the tree contains at most $(1-1/n)$ negated literals for every positive literal in the tree (as a single positive literal may be exempt from deletion by being a part of the only remaining contradiction), the tree will also with high probability contain at least $Y > X(1-1/n)/2 > X/3$ negated literals. While $X \geq n$ positive literals remain in the tree, we apply the negative drift theorem (Theorem 5) to show that it takes an

exponential number of HVL-Prime sub-operations to reduce the number of negated literals from $Y \geq n/3$ to $Y \leq n/4$, as there is a strong negative drift to overcome:

$$\begin{aligned}
&E(Y_{t+1} - Y_t \mid n/3 \geq Y_t \geq n/4, X_t \geq n) \\
&= \frac{1}{3 \cdot 2} - \frac{Y_t}{3(Y_t + X_t)} + \frac{1}{3}\left(\frac{X_t}{2 \cdot (Y_t + X_t)} - \frac{Y_t}{2 \cdot (Y_t + X_t)}\right) \\
&> \frac{1}{6} - \frac{4}{45} - \frac{4}{90} = \frac{3}{90}
\end{aligned}$$

using $Y_t \leq n/3$ and $X_t + Y_t \geq 5n/4$ in negative terms.

The large jump condition is trivially satisfied as we are considering the drift in terms of HVL-Prime sub-operations rather than iterations of the algorithm, and each sub-operation can only change the number of negated literals present in the tree by at most 1.

If the number of positive literals in the tree drops below n, we know that the current solution is not optimal, and can apply the initial argument showing that more than $X/3$ negated literals exist in the tree the next time the tree contains $X \geq n$ positive literals. □

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 were never 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 becomes 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.

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 4.2.1 using minimal terminal and function sets, while the presence of negated variables in the current solution does not significantly affect the generalisation error on AND_n. Thus, the results of Theorems 12 (and the analysis of the generalisation ability of RLS-GP by Mambrini & Oliveto, 2016) and 14 can be adapted to yield the following corollary.

Corollary 17 (of Theorems 12 and 14)**.** *For any desired generalisation error n^{-c}, where $c > 0$ is a constant, there exists a polynomial training set size s such that if s rows are sampled with replacement from the complete truth table of AND_n either at the beginning of the run, or independently in every iteration, then the* $(1+1)$ GP *and the RLS-GP algorithms using $F = \{AND\}$ and $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ are able to find a solution with a generalisation error of at most n^{-c} in expectation after $O(\log n)$ (for a static training set) or $O(\log^2 n)$ iterations (for a dynamic training set).*

This follows from a symmetry argument: for AND_n, a non-contradiction conjunction containing at least one negated variable, and v distinct variables in total, is wrong on at most two more inputs than a solution containing v distinct variables in positive form (namely, the all-true input, and the input setting only the variables present in the solution as positive literals to true). This also implies that the same result would hold for any target function that is a conjunction of all n variables, each variable appearing in either positive or negative form.

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: $\text{AND}_{n,m}$

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, modelling 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 $\text{AND}_{n,m}$ 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 18. *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 $f_{a,b} = 2^{n-a-b} + 2^{n-m} - 2^{n+1-m-b}$ rows of the n-variable truth table for $\hat{h}$.*

Proof. A conjunction of $a \leq m$ distinct variables would differ from $\hat{h}$ when all a variables are set to 1, but at least one of $m - a$ variables is set to 0; this occurs in $2^{m-a} - 1$ rows of the truth table for the m variables in $\hat{h}$.

If a conjunction additionally contains b distinct variables not in $\hat{h}$, it differs from $\hat{h}$ both on $2^{m-a} - 1$ rows (where $a + b$ variables are set to 1, and at least one other variable in $\hat{h}$ to 0), and on $2^b - 1$ rows (where all m variables are set to 1, and at least one of b variables is set to 0) of the truth table for the $m + b$ variables.

Let $c = n - m - b$ be the number of distinct variables neither in the conjunction nor in $\hat{h}$: each such variable effectively doubles the number of truth table rows on which the conjunction and $\hat{h}$ differ.

Thus, a conjunction with $a \leq m$ distinct variables in $\hat{h}$, $b \leq n-m$ distinct variables not in $\hat{h}$, and $c = n-m-b$ variables in neither $\hat{h}$ nor the conjunction, would differ from $\hat{h}$ on

$$\begin{aligned} f_{a,b} &= 2^c \left(2^{m-a} - 1 + 2^b - 1\right) \\ &= 2^{n-m-b} \left(2^{m-a} + 2^b - 2\right) \\ &= 2^{n-a-b} + 2^{n-m} - 2^{n+1-m-b} \end{aligned}$$

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 19. *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 $a = m$.*

Proof. Substituting $a = m - i$ yields $f_{a,b} = 2^{n-m-b}(2^i - 2^1) + 2^{n-m}$. □

We will show that the HVL-Prime `SUBSTITUTE` operation prevents the RLS-GP* algorithm from finding an optimal solution when using the complete truth table as the 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 20. *The RLS-GP* algorithm with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$ will with high probability fail to find the optimum of $AND_{n,m}$ in finite time when $m = cn$ for any constant $0 < c < 1$ when using the complete truth table as the training set.*

Proof. While the current solution contains fewer than $m-1$ distinct variables of $\hat{h}$, mutations which add new distinct variables not in $\hat{h}$ will improve fitness (per Observation 19), and therefore be accepted by the RLS-GP*. Once all m $\hat{h}$ variables have been added to the tree, mutations which remove non-$\hat{h}$ variables (or substitute them with a variable already present in the current solution) will improve fitness, and therefore be accepted by the RLS-GP*. If a substitution replacing a non-$\hat{h}$ variable with another non-$\hat{h}$ variable, which is already present in the tree, is ever accepted, the RLS-GP* will not be able to reach the global optimum, as no HVL-Prime mutation removing or replacing a non-final copy of a non-$\hat{h}$ variable can improve fitness.

It remains to show that a problematic substitution is likely to occur during the optimisation process. In expectation, it takes $\sum_{i=0}^{m-2} \frac{n}{m-i} = O(n \log m)$ HVL-Prime insertions for the RLS-GP* to collect $m-1$ distinct $\hat{h}$ variables. During this time, mutations adding non-$\hat{h}$ variables may also be accepted. Let $w = (1-c)n$ be the number of distinct variables not in $\hat{h}$. While there are fewer than $h = w/2$ such variables in the solution, the probability that an addition selects a new non-$\hat{h}$ variable is at least a constant: $(w-h)/n = (1-c)/2$. Applying Chernoff bounds (Mitzenmacher & Upfal, 2005), the probability that after n additions occurring prior to the $m-1$'th $\hat{h}$-variable being added to the tree, fewer than $(1-c)n/4$ distinct non-$\hat{h}$ variables have been added is at most $e^{-\Omega(n)}$.

Thus, with overwhelming probability, the tree will contain at least $(1-c)n/4 = c'n$ (where $c' > 0$ is a constant) non-$\hat{h}$ variables upon adding the final $\hat{h}$-variable. Non-$\hat{h}$ variables can be removed either through deletion or through substitution. While at least $c'n/2 + 1$ distinct non-$\hat{h}$ variables remain in the tree, the probability that substitution choses one of the remaining non-$\hat{h}$ variables as the replacement is at least $c'/2$, i.e., a constant. Thus, the probability that when a non-$\hat{h}$ variable is removed, it is through substitution adding a copy of another non-$\hat{h}$ variable is at least a constant. The probability that substitution introduces a copy of a non-$\hat{h}$ variable over the course of $c'n/2$ removals is overwhelmingly high: $1-(1-c'/2)^{c'n/2}$. □

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 18, and setting $a = m = cn, b = (n-cn)/2$, 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 `SUBSTITUTE` 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 21. *The RLS-GP* algorithm, using the HVL-Prime mutation operator without the* `SUBSTITUTE` *operation, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$, and using the complete truth table as the training set, will find the optimum of $AND_{n,m}$ in expected $O(n \log n)$ iterations.*

Proof. Without substitution, HVL-Prime mutation can only introduce copies of variables into the tree through insertions, and as such mutations would not change the fitness value of the solution, they would not be accepted by the RLS-GP*.

Applying Theorem 6, we conclude that the expected number of iterations required for the RLS-GP* algorithm to add all m variables in $\hat{h}$ to the current solution is at most $O(n \log m)$. During this time, up to $n-m$ variables not in $\hat{h}$ may also be added, producing a tree of size at most n. Once all $\hat{h}$ variables have been added, it would take up to an expected $O(n \log n)$ iterations to remove the variables not found in $\hat{h}$ via HVL-Prime deletion mutations, since per Observation 19 mutations removing such variables improve fitness.

□

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 22. *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 $AND_{n,m}$, and using the complete truth table as the training set, remains below $3n$ during the first $O(n \log n)$ iterations with probability at least $1 - o(1)$.*

Proof. Let X_t be the number of terminals in the current solution tree. If the size of the tree is to exceed $2n$, some variables must appear in the tree more than twice; however, if the tree contains at least n terminals, an accepted mutation is at least as likely to delete an existing copy of such a variable as it is to insert an additional copy. Consider Δ_2, the expected change in the size of the tree due to addition or deletion of variables which appear at least twice in the solution at time t; and let X_t be the total number of terminals in the tree at time t, and Y_t and Z_t be the number of variables which appear in the tree at time t exactly once or exactly zero times respectively; then Δ_2 is:

$$\frac{n-Z_t-Y_t}{n}-\frac{X_t-Y_t}{X_t}\leq\frac{n-Y_t}{n}-\frac{X_t-Y_t}{X_t}-\frac{Z_t}{n}\leq 0$$

with the final inequality requiring that $X_t \geq n$.

We can upper-bound the number of terminals in the tree at time t as $2n$ plus a contribution from a fair random walk on $\mathbb{N}$ starting at 0. Every time the contribution from the fair random walk increases from 0 to 1, we apply a Gambler's Ruin argument (Feller, 1968), concluding that with probability $1-1/n$, the walk returns to 0 before reaching n; this return takes an expected $O(n)$ steps. By the additive drift theorem (Theorem 3) and Markov bounds (as in the proof of Theorem 10), no more than $O(\log^2 n)$ such games occur with high probability. Taking a union bound, the probability that all games result in returning to 0 is at least $(1-1/n)^{\log^2 n} \geq 1-(\log^2 n)/n \geq 1-o(1)$. Thus, with high probability, $X_t \leq 3n$ for the first $O(n\log n)$ iterations. □

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

Theorem 23. *The RLS-GP algorithm, using the HVL-Prime mutation operator with $F = \{AND\}$ and $L = \{x_1,\ldots,x_n\}$, and using the complete truth table as the training set, will find the optimum of $AND_{n,m}$ in $O(n\log n)$ iterations with probability $1-o(1)$.*

Proof. We will show that within the $O(n\log n)$ iterations during which, per Lemma 22, the current solution tree does not grow beyond $3n$ terminals, the RLS-GP constructs a solution containing all $\hat{h}$ variables and no non-$\hat{h}$ variables.

In $O(n\log m)$ iterations, with high probability, the current solution tree will contain at least one copy of every $\hat{h}$ variable. Once this occurs, the RLS-GP will accept removals of the last instance of non-$\hat{h}$ variables, and will not accept mutations which increase the number of distinct non-$\hat{h}$ variables in the current solution.

Without loss of generality, let $x_1 \notin \hat{h}$, C_t be the number of copies of x_1 in the tree at time t, and X_t be the total number of terminals in the tree at time t. Consider the expected change of C_t in a single iteration:

$$\begin{aligned}E(C_{t+1}-C_t \mid C_t) &= \frac{1}{3n}-\frac{C_t}{3X_t}+\frac{2}{3}\frac{X_t-C_t-C_t(n-1)}{nX_t}\\ &= \frac{1}{3n}-\frac{C_t}{3X_t}+\frac{2}{3n}-\frac{2C_t}{3X_t}=\frac{1}{n}-\frac{C_t}{X_t}\end{aligned}$$

Given that $X_t \leq 3n$, when $C_t > 3$, there is a negative drift on C_t:

$$E(C_{t+1}-C_t \mid C_t>3, X_t\leq 3n)\leq\frac{1}{n}-\frac{4}{3n}\leq-\frac{1}{3n}.$$

Thus, by applying Theorem 5, we can conclude that the probability that C_t first exceeds $c' \log n$, where $c' > 0$ is an appropriately-chosen constant, within $2^{\Omega(\log n)} = n^{\Omega(1)}$ iterations is at most $2^{-\Omega(\log n)} = n^{-\Omega(1)}$.

So $C_t \leq c' \log n$ when the m variables in $\hat{h}$ have been added to the tree. Applying the additive drift theorem (Theorem 3), C_t is reduced to 3 within at most $O(n \log n)$ iterations.

Consider the probabilities that the next operation to affect C_t will increase or decrease it, recalling that X_t is the number of terminals in the tree at time t:

$$\Pr(C_{t+1} > C_t) = \frac{1 + (n-1)/n}{3n} = \frac{2 - 1/n}{3n}$$
$$\Pr(C_{t+1} < C_t) = \frac{C_t}{3X_t} + \frac{C_t(n-1)}{3X_t n} = \frac{(2-1/n)C_t}{3X_t}$$

which can be used to bound the probability that the next change will be a decrease, with notation omitting the $C_t \neq C_{t+1}$ condition for formatting reasons:

$$\begin{aligned}\Pr(C_{t+1} < C_t \mid \ldots) &= \frac{\Pr(C_{t+1} < C_t)}{\Pr(C_{t+1} < C_t) + \Pr(C_{t+1} > C_t)} \\ &= \frac{(2-1/n)C_t}{(2-1/n)C_t + (2-1/n)(X_t/n)} \\ &= \frac{C_t}{C_t + X_t/n}\end{aligned}$$

noting that when $X_t \leq 3n$, this probability is at least $1/2$, $2/5$, and $1/4$, for $C_t = 3$, $C_t = 2$, and $C_t = 1$ respectively.

Assume $C_t = 3$. If the next three operations affecting C_t are all deletions, the variable is removed entirely. With combined probability $1/2 \cdot 2/5 \cdot 1/4 = 1/20$, the next three changes in C_t decrease it, and the sum of their waiting times is at most $2n + 3n + 6n = 11n$ (upper-bounded using $\Pr(C_{t+1} < C_t) \geq C_t/(2X_t)$, which holds for $n \geq 2$). If instead C_t increases beyond 3 then, by the additive drift theorem, it will return to 3 in expected cn steps for some constant $c > 0$. Thus, in expectation, the variable is removed $20(c+11)n$ iterations after first hitting $C_t = 3$. Doubling this and applying Markov's inequality, x_1 is removed completely after $40(c+11)n$ iterations with probability at least $1/2$; and in $c' \log(n)$ repetitions of $40(c+11)n$ iterations, it is removed with probability $1 - 2^{-c' \log n} = 1 - n^{-c'}$.

Applying a union bound, with probability $1 - n^{-c'+1}$, all non-$\hat{h}$ variables are removed from the tree in time $O(n \log n)$ with probability at least $1 - n^{-c'+1} = 1 - o(1)$.

Note that even if X_t exceeds $3n$, a random walk will return to $X_t = 2n$ in expected $O(n^2)$ iterations. □

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. For the analysis, 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 et al., 2008), and compare the fitness of two solutions by sampling s rows from the complete truth table independently at random in each iteration. We leave whether this tree size limit

is necessary at all, or whether it could be replaced by a bloat control mechanism as open questions.

Theorem 24. *For any desired generalisation error n^{-c}, where $c > 0$ is a constant, the RLS-GP* (without the substitution sub-operation of HVL-Prime) and RLS-GP algorithms, with $F = \{AND\}$ and $L = \{x_1, \ldots, x_n\}$, using a tree size limit $T_{\max} = 1.25c \log_2 n$, and sampling $s \in \Omega(n^{5c+1})$ rows from the complete truth table uniformly at random in each iteration will construct a solution with the desired generalisation error on the $AND_{n,m}$ problem within $O(cn \log(n) \log(m))$ iterations in expectation.*

Proof. For any $s \in \Omega(n^{c+\epsilon})$, and any positive constant ϵ, the expected number of rows on which a solution with a generalisation error of more than n^{-c} is wrong on in a training set of size s is $\Omega(n^{\epsilon})$. By a Chernoff bound, the probability that such a solution is wrong on at least $n^{\epsilon}/2$ rows is at least $e^{-\Omega(n^{\epsilon})}$, and by a union bound, this event does not occur in any polynomial number of iterations with overwhelming probability as ϵ is at least a constant greater than zero. Thus, if $s \in \Omega(n^{c+\epsilon})$, and the GP algorithms terminate in polynomial time, the generalisation error of the returned solution will be less than n^{-c}.

We set $T_{\max}$ such that a solution consisting of any $T_{\max}$ distinct variables will achieve the desired generalisation error when $m > T_{\max}$. For this purpose, $T_{\max} = 1.25 \log_2 n$ is sufficient: using Proposition 18, the generalisation error of a solution with $a + b = T_{\max}$ distinct variables is maximised when $b = T_{\max}$, and hence is at most $2n^{-1.25c} - 2n^{-2.5c} < n^{-c}$.

It remains to be shown that when $m \leq T_{\max}$, the GP algorithms can construct a solution with exactly m distinct correct variables in polynomial time. Typically, the algorithms will proceed by adding random variables to the tree until it is at the size limit, substituting incorrect variables for correct variables, and finally removing any excessive incorrect variables that remain. Throughout this, we need to show that the fitness function evaluation based on sampling s rows of the complete truth table independently uniformly at random in each iteration correctly identifies mutations following this pattern as beneficial, and mutations opposing this pattern as detrimental.

Suppose two solutions are being compared on a training set of size s. If one of the solutions is correct on D more complete truth table rows than the other, the expected result of the evaluation is the better solution will be favoured by $(D/2^n)s$ sampled rows. We can apply Azuma's inequality (Doerr, 2011) to bound the probability that the better solution is not identified correctly. Let X be the net number of rows favouring the better solution in the sampled training set of s rows; then, $E(X) = (D/2^n)s$ and letting $\lambda = (D/2^n)s/2$,

$$\begin{aligned}\Pr(X \leq E(X) - \lambda) &\leq \exp\left(-2\lambda^2 / \sum_{i=1}^{s} c_i^2\right) \\ &\leq \exp(-(D/2^n)^2 s/2)\end{aligned}$$

by Azuma's inequality. Hence, the better solution is identified correctly with high probability if $(D/2^n)^2 s \in \Omega(\log n)$.

We will show that by picking a large-enough s, the RLS-GP variants will with high probability accept mutations which increase the number of distinct correct variables in the solution, and will with high probably reject solutions which decrease this number. Recall that per Proposition 18 a solution with a distinct correct variables, and b distinct wrong

variables differs from the target function on $f_{a,b} = 2^n(2^{-a-b} + 2^{-m} - 2^{1-m-b})$ rows of the complete truth table.

There are two types of RLS-GP mutations which increase the number of distinct correct variables in the solution: a correct variable may be inserted or substituted for a duplicate copy of another variable, or it may be substituted for the last copy of an incorrect variable. Let $D_{+1,0}$ and $D_{+1,-1}$ be the difference in the number of correct complete truth table rows between the parent and the offspring affected by the two respective mutation types:

$$D_{+1,0} := f_{a,b} - f_{a+1,b} = 2^n \cdot 2^{-a-b-1}$$
$$D_{+1,-1} := f_{a,b} - f_{a+1,b-1} = 2^n \cdot 2^{1-m-b}$$

and using $(a+b) \leq T_{\max}$ and $m \leq T_{\max}$, we note that $\min(D_{+1,0}, D_{+1,-1}) > 2^n \cdot 2^{-2T_{\max}} \geq 2^n \cdot n^{-2.5c}$.

Similarly, decreasing the number of distinct correct variables might happen with or without simultaneously increasing the number of incorrect variables in the offspring. Let $D_{-1,0}$ and $D_{-1,+1}$ be the difference in the number of correct complete truth table rows between the parent and the offspring affected by a mutation which does not (and does, respectively) increase the number of distinct incorrect variables in the solution:

$$D_{-1,0} := f_{a-1,b} - f_{a,b} = 2^n \cdot 2^{-a-b}$$
$$D_{-1,+1} := f_{a-1,b+1} - f_{a,b} = 2^n \cdot 2^{-m-b}$$

and as before, we note that $\min(D_{-1,0}, D_{-1,+1}) \geq 2^n \cdot 2^{-2T_{\max}} \geq 2^n \cdot n^{-2.5c}$.

Thus, with $s \in \Omega(n^{5c+1})$, the RLS-GP algorithms will accept each mutation increasing the number of distinct correct variables, and reject each mutation decreasing the number of distinct correct variables with probability $\exp\left(-(D/2^n)^2 s/2\right) = e^{-\Omega(n)}$. By a straightforward union bound, the GP algorithms will with high probability not deviate from this behaviour in any polynomial number of fitness evaluations.

We note that while the current solution contains fewer than $\min(T_{\max}, m)$ distinct correct variables, there is always an RLS-GP mutation available which increases the number of distinct correct variables in the solution (either through insertion or substitution), and hence will be accepted with high probability by the above arguments. Pessimistically assuming that all such increases must be made by specific substitution operations, which occur with probability at least $1/3 \cdot (1/T_{\max}) \cdot (n-i)/n$, where i is the number of correct variables already present in the current solution, the expected number of iterations before RLS-GP algorithms collect the m distinct correct variables in the current solution is at most $3T_{\max} n \log(m)$ (similarly to the result of Theorem 6).

If $m < T_{\max}$, it may be the case that some wrong variables still remain in the solution when all m distinct correct variables have been inserted. Removing the last copy of each such variable improves fitness by $D^*_{0,-1}$ rows, and inserting a new distinct wrong variable decreases fitness by $D^*_{0,+1}$ rows; using that $a = m$, this yields:

$$D^*_{0,-1} := f_{a,b} - f_{a,b-1} = 2^n \cdot 2^{-b-m}$$
$$D^*_{0,+1} := f_{a,b+1} - f_{a,b} = 2^n \cdot 2^{-b-m}/2$$

and as $b \leq T_{\max}$, both of these are at least $2^n \cdot 2^{-2T_{\max}}/2 = 2^n \cdot n^{-2.5c}/2$.

To make sure that the RLS-GP correctly accepts mutations that reduce the number of distinct wrong variables, and rejects mutations that increase this number, once all correct variables have been collected in the solution, $s \in \Omega(n^{5c+1})$ is sufficient. As selecting these variables for deletion is much easier than inserting a new copy (at least $i/(3\,T_{\max})$ probability of deleting one of these variables while i copies remain, and only at most i/n probability of inserting or substituting another copy of one of these variables into the tree), by the additive drift theorem (Theorem 3), in expected $O({T_{\max}}^2) = o(n)$ iterations all wrong variables will be removed from the tree.

Thus, after an expected $O(cn\log(n)\log(m))$ iterations, each sampling $s \in \Omega(n^{5c+1})$ rows from the complete truth table, a solution with the desired generalisation error, i.e., at most n^{-c}, is constructed. □

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. The problem of identifying small training sets that allow for efficient evolution is considered in practical applications of GP (e.g., via the Dynamic or Historical Subset Selection mechanisms, Gathercole & Ross, 1994). Hence, identifying problems for which such subsets exist is an important research question. In this section, we show that for the conjunction problem, training sets of linear size which do allow efficient evolution exist. Evaluating whether the methods used in practice are capable of efficiently constructing such training sets is an interesting avenue for future work.

Consider a minimal training set M consisting of n rows, where the i'th row sets x_i to false and all other input variables 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 25. *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).*

Proof. On M, the fitness of a solution reflects the number of distinct literals in that solution: thus, mutations increasing the number of distinct literals are always accepted, and mutations reducing the number of distinct literals are never accepted. Using this observation, we can apply the same reasoning as used in the proof of Theorem 9.

With probability $\Omega(i/n)$, where i is the number of missing variables in the current solution, a mutation will increase the number of distinct literals in the current solution (by inserting a new distinct literal). As no mutation which decreases the number of distinct literals is ever accepted, all n variables will be present in the solution after an expected $O(n\log n)$ mutations by Theorem 6. □

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 $\text{AND}_{n,m}$ problem.

Theorem 26. *The RLS-GP and 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).*

Proof. On M, inserting a variable x_i not present in the target function $\hat{h}$ will increase the error: the row setting x_i to false will now be wrong, and no rows will be rendered correct (as no other row sets x_i to false). Neither the RLS-GP nor RLS-GP* will thus accept a solution with any wrong variables. Similarly, no mutations which decrease the number of distinct correct variables in the solution can be accepted.

As mutations which insert new distinct correct variables *are* accepted, applying Theorem 6 yields that the RLS-GP and RLS-GP* algorithms will produce the optimal solution in expected $O(n \log n)$ iterations. □

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 23.

As it does not appear to be possible for the (1+1) GP to evolve the exact conjunction of n variables if their negations are also present in the terminal set (even when using the complete truth table, per Theorems 15 and 16), we show how a more careful choice of the training set can be beneficial.

Theorem 27. *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, $F = \{AND\}$ 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 for AND_n in $O(n \log n)$ iterations.*

Proof. We note that any solution containing any negated variable will be wrong on at least $n + 1$ copies of the all-true truth table row, making it worse than any solution which does not contain a negation (and is therefore wrong on at most n rows). Thus, mutations which introduce negated literals are never accepted by the (1+1) GP algorithms.

If a solution and its offspring contain no negated literals, mutations which increase the number of distinct positive literals will improve the offspring's fitness. Conversely, decreasing the number of distinct positive literals will reduce the offspring's fitness.

Thus, the GP algorithms' behaviour can be modelled by the coupon collector problem: with at least constant probability, they will insert a single variable chosen uniformly at random from the terminal set, and after $O(n \log n)$ such insertions, will have inserted each positive terminal at least once. As the number of distinct positive terminals in the solution cannot decrease (without decreasing fitness), the optimal solution will have been found after at most $O(n \log n)$ iterations in expectation. □

A similar effect can be achieved by using a population with an explicit diversity mechanism (as employed in evolutionary algorithms to support global exploration of the search space, Friedrich, Oliveto, Sudholt, & Witt, 2009; Oliveto, Sudholt, & Zarges, 2019), 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. Note that this mechanism is different from the fitness diversity mechanism often considered in evolutionary computation literature (Friedrich et al., 2009; Oliveto & Zarges, 2015), which does not accept individuals with a fitness value already present in the population, even if their phenotypes differ.

Theorem 28. *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, $F = \{AND\}$ 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 for AND_n in expected $O(\mu n \log n)$ iterations.*

Proof. The phenotype diversity mechanism will reject any mutation which exactly replicates the training set behaviour of any solution already present in the population.

A solution containing a negation $\overline{x_i}$ and either the positive literal x_i or any other negation $\overline{x_j}$ $(j \neq i)$ will evaluate to 0 on all rows of the training set, and hence be wrong on only the all-true row.

A solution containing a single negation $\overline{x_i}$, and lacking the positive literal x_i, will evaluate to 0 on all but one of the training set rows, and is wrong on two rows: the all-true row, and the row that sets x_i to false while all remaining variables are true.

Hence, there are a total of $n + 1$ distinct evaluations of solutions containing negative literals on the considered training set. Once a solution evaluating to each of these evaluation profiles is present in the population, no offspring containing negative literals will be accepted. Thus, at least one individual in the population will never contain negative literals, and hence can increase in fitness by accumulating positive literals. Applying Theorem 6 using $i/(\mu n)$ as the probability of acquiring a new coupon in a single iteration, we conclude that after an expected $O(\mu n \log n)$ iterations, mutations which select this individual as an ancestor will have added all n distinct positive literals to it, producing the optimal solution. □

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. Our analysis shows how changing even slightly the parameters of the GP system may drastically affect its performance for evolving Boolean conjunctions. In particular, we have shown how the results are affected by modifying mutation and selection operators and the influence of different function and terminal sets.

We first 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

Problem	RLS-GP	$(1+1)$ GP
AND_n with $F = \{AND\}$, $L = \{x_1, \ldots, x_n\}$		
Complete	$\Theta(n \log n)^{\S}$	$\Theta(n \log n)$ (Th. 9)
Static	$O(\log n)^{\S}$	$O(\log n)$ (Th. 11)
Dynamic	$O(\log^2 n)$ (Th. 14)	$O(\log^2 n)$ (Th. 14)
AND_n with $F = \{AND\}$, $L = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$		
Complete	$\omega(\text{poly}(n))^{\S}$, $\infty^{*,\S}$	$\omega(\text{poly}(n))$, ∞^* (Th. 15, 16)
Static	$O(\log n)$ (Cor. 17)	$O(\log n)$ (Cor. 17)
Dynamic	$O(\log^2 n)$ (Cor. 17)	$O(\log^2 n)$ (Cor. 17)
$\text{AND}_{n,m}$ with $F = \{AND\}$, $L = \{x_1, \ldots, x_n\}$		
Complete	$O(n \log n)$ w.p. $1 - o(1)$ (Th. 23), ∞^*, $O(n \log n)^{\dagger}$ (Th. 20, 21)	?
Dynamic	$O(cn \log(n) \log(m))^{\ddagger}$ (Th. 24)	?

* Applies for RLS-GP* or $(1+1)$ GP* respectively.
† Applies for RLS-GP* (without `SUBSTITUTE`).
‡ Applies for RLS-GP with tree size limit $T_{\max} = c \log_2 n$.
§ Proved by Mambrini and Oliveto (2016).

Table 1: A summary of the known bounds on the expected number of iterations required by the $(1+1)$ GP and RLS-GP algorithms to produce a solution fitting the complete truth table (denoted Complete), or polynomially-sized training sets sampled from the complete truth table uniformly at random either at the beginning of the run (denoted Static) or independently in every iteration (denoted Dynamic).

analysis for AND_n provided a relationship between the number of variables in 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. Our analysis highlights the impact of parameters such as the mutation rate, function and terminal sets, and strict elitism of the selection operator on the performance of the GP system.

For AND_n with minimal function and terminal sets we show that the $(1+1)$ GP and $(1+1)$ GP* algorithms produce a solution that fits 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 not known (i.e. the $\text{AND}_{n,m}$ setting), we have demonstrated that the non-strictly elitist RLS-GP has an advan-

tage over the RLS-GP* when using the complete truth table as the training set. RLS-GP is also able to achieve polynomially-small generalisation errors in this setting when evaluating program fitness using a polynomially-large training set sampled in each iteration, while RLS-GP* may require the substitution sub-operation of HVL-Prime to be switched off to achieve this. For both algorithms, a limit on the size of the accepted trees may be required if the target conjunction consists of a (sub-)logarithmic number variables; whether this limit is necessary or whether it could be replaced by a bloat control mechanism remain open questions. Recent work has considered the more complicated setting of evolving conjunctions using $F = \{\text{AND}, \text{OR}\}$, albeit only for the AND_n problem (Doerr, Lissovoi, & Oliveto, 2019).

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, & Wegener, 2002; Oliveto & Yao, 2011). Hence, it is ideal as an easy benchmark function to evaluate the hillclimbing characteristics of GP systems.

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.). 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 by 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 population based GP systems, as is possible nowadays for standard genetic algorithms (Dang et al., 2018; Corus, Dang, Eremeev, & Lehre, 2018; Corus & Oliveto, 2018, 2019).

Acknowledgements

An extended abstract of this paper has been published at the 32nd AAAI Conference on Artificial Intelligence (Lissovoi & Oliveto, 2018). 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., & 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., & Vanneschi, L. (2007). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, *8*(4), 413–432.

Corus, D., Dang, D., Eremeev, A. V., & Lehre, P. K. (2018). Level-based analysis of genetic algorithms and other search processes. *IEEE Trans. Evolutionary Computation*, *22*(5), 707–719.

Corus, D., & Oliveto, P. S. (2018). Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *22*(5), 720–732.

Corus, D., & Oliveto, P. S. (2019). On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)*, pp. 1452–1460.

Curry, R., Lichodzijewski, P., & Heywood, M. I. (2007). Scaling genetic programming to large datasets using hierarchical dynamic subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, *37*(4), 1065–1073.

Dang, D., Friedrich, T., Kötzing, T., Krejca, M. S., Lehre, P. K., Oliveto, P. S., Sudholt, D., & Sutton, A. M. (2018). Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, *22*(3), 484–497.

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

Doerr, B. (2019). Probabilistic tools for the analysis of randomized optimization heuristics. In Doerr, B., & Neumann, F. (Eds.), *Theory of Evolutionary Computation (to appear)*. Springer. CoRR, abs/1801.06733.

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

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

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

Doerr, B., Lissovoi, A., & Oliveto, P. S. (2019). Evolving Boolean functions with conjunctions and disjunctions via genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2019)*, pp. 1003–1011.

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

Durrett, G., Neumann, F., & O'Reilly, U. M. (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)*, pp. 69–80.

Feldman, V. (2008). Evolvability from learning algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, pp. 619–628.

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

Friedrich, T., Oliveto, P. S., Sudholt, D., & Witt, C. (2009). Analysis of diversity-preserving mechanisms for global exploration. *Evolutionary Computation*, *17*(4), 455–476.

Gathercole, C., & Ross, P. (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Proceedings of the 3rd Parallel Problem Solving from Nature Conference (PPSN III)*, pp. 312–321.

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

He, J., & Yao, X. (2004). A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, *3*(1), 21–35.

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

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

Kötzing, T., Lagodzinski, J. A. G., Lengler, J., & Melnichenko, A. (2018). Destructiveness of lexicographic parsimony pressure and alleviation by a concatenation crossover in genetic programming. In *Proceedings of the 15th Parallel Problem Solving from Nature Conference (PPSN XV), Part II*, pp. 42–54.

Kötzing, T., Sutton, A. M., Neumann, F., & O'Reilly, U. M. (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., & Poli, R. (2002). *Foundations of genetic programming*. Springer.

Lengler, J. (2019). Drift analysis. In Doerr, B., & Neumann, F. (Eds.), *Theory of Evolutionary Computation (to appear)*. Springer. CoRR, abs/1712.00964.

Lissovoi, A., & Oliveto, P. (2019). Computational complexity analysis of genetic programming. In Doerr, B., & Neumann, F. (Eds.), *Theory of Evolutionary Computation (to appear)*. Springer. CoRR, abs/1811.04465.

Lissovoi, A., & Oliveto, P. S. (2018). On the time and space complexity of genetic programming for evolving Boolean conjunctions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 1363–1370. AAAI Press.

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

Mambrini, A., & 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)*, pp. 99–114.

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

Oliveto, P. S., He, J., & Yao, X. (2007). Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, *4*(3), 281–293.

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

Oliveto, P. S., & Witt, C. (2012). Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *CoRR*, *abs/1211.7184*.

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

Oliveto, P. S., Sudholt, D., & Zarges, C. (2019). On the benefits and risks of using fitness sharing for multimodal optimisation. *Theoretical Computer Science*, *773*, 53–70.

Oliveto, P. S., & Zarges, C. (2015). Analysis of diversity mechanisms for optimisation in dynamic environments with low frequencies of change. *Theoretical Computer Science*, *561*, 37–56.

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

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

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

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

Schuh, M. A., Angryk, R. A., & 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., & 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., & Urli, T. (2015). On the performance of different genetic programming approaches for the SORTING problem. *Evolutionary Computation*, *23*(4), 583–609.